Discovering Hierarchical Neural Archetype Sets

Ciravegna Gabriele[1][0000−0002−6799−1043], Pietro Barbiero[2][0000−0003−3155−2564], Giansalvo Cirrincione[3], Squillero Giovanni[2][0000−0001−5784−6435], and Tonda Alberto[4,5][0000−0001−5895−4809]

[1] University of Siena, Siena, Italy
[2] Politecnico di Torino, Torino, Italy
pietro.barbiero@polito.it
[3] University of the South Pacific, Suva, Fiji
nimzoexin59@gmail.com
[4] Université Paris-Saclay, Saclay, France
[5] UMR 782 INRA, Thiverval-Grignon, France
alberto.tonda@inra.fr

Abstract. In the field of machine learning, coresets are defined as subsets of the training set that can be used to obtain a good approximation of the behavior that a given algorithm would have on the whole training set. Advantages of using coresets instead of the training set include improving training speed and allowing for a better human understanding of the dataset. Not surprisingly, coreset discovery is an active research line, with several notable contributions in literature. Nevertheless, restricting the search for representative samples to the available data points might impair the final result. In this work, neural networks are used to create sets of virtual data points, named *archetypes*, with the objective to represent the information contained in a training set, in the same way a coreset does. Starting from a given training set, a hierarchical clustering neural network is trained and the weight vectors of the leaves are used as archetypes on which the classifiers are trained. Experimental results on several benchmarks show that the proposed approach is competitive with traditional coreset discovery techniques, delivering results with higher accuracy, and showing a greater ability to generalize to unseen test data.

Keywords: archetypes, big data, classification, coresets, explain AI, hierarchical clustering, machine learning, neural networks, XAI

1 Introduction

The concept of *coreset*, stemming from computational geometry, has been redefined in the field of machine learning (ML) as the subset of input samples of minimal size, for which a ML algorithm can obtain a good approximation of the behavior the algorithm itself would normally have on the whole set of input samples. Differently speaking, a coreset can be seen as a fundamental subset of a target training set that is sufficient for a given algorithm to deliver good results, or even the same results it would have if trained on the whole training set [1]. While this definition might appear generic, it must be noted that it encompasses significantly different tasks, ranging from classification, to regression, to clustering, whose performance is measured in entirely different ways. Practical applications of coresets include: obtaining a better understanding of the data, drastically reducing the number of data points a human expert has to analyze; and considerably speeding up training time of ML algorithms.

Coreset discovery is an active research line, and specialized ML literature reports a substantial number of approaches: Greedy Iterative Geodesic Ascent (GIGA) [2], Frank-Wolfe (FW) [3], Forward Stagewise (FSW) [4], Least-angle regression (LAR) [5] [6], Matching Pursuit (MP) [7], and Orthogonal Matching Pursuit (OMP) [8]. Often such algorithms require the user to specify the number N of desired points in the coreset; or assume, for simplicity, that a coreset is independent from the task and/or the algorithm selected for that task. As ML algorithms employ different techniques to accomplish the same goals, however, it is reasonable to assume that they would need coresets of different size and shape to operate at the best of their possibilities. Furthermore, restricting the search of coresets to the data points actually available in the given dataset might impair the final result: as coresets can be seen as a *summary* of the information contained in a dataset, it is conceivable that a set of virtual points might better represent the information in an even more concise way. In the following, we call these virtual data points *archetypes*.

Starting from these two considerations, a neural network approach to archetype set discovery for classification tasks is proposed. Starting from a training set, a GH-ARCH (Growing Hierarchical Archetypes) algorithm is used to find the archetype sets best representing the geometry of the dataset. GH-ARCH is a variant of the recently published GH-EXIN neural network [9] which performs hierarchical clustering by building a divisive hierarchical tree in an incremental and self-organized way. Indeed, in this work it will be shown how cluster centroids are the best virtual points to represent the information contained in a given dataset. Moreover, the hierarchical architecture of the network allows to select the best trade-off between the number of points used and the accuracy of the ML algorithm, by simply stopping the network when a desired resolution level is reached.

Experimental results on classification benchmarks show that the proposed approach is able to best several state-of-the-art coreset discovery algorithms in literature, obtaining results that also allow the classifier to generalize better on an unseen test set of the same benchmark.

2 Background

In computational geometry, coresets are defined as a small set of points that approximates the shape of a larger point set. The concept of coreset in ML is extended to intend a subset of the (training) input samples, such that a good approximation to the original input can be obtained by solving the optimization problem directly on the coreset, rather than on the whole original set of input samples [1].

Finding coresets for ML problems is an active line of research, with applications ranging from speeding up training of algorithms on large datasets [10] to gaining a better understanding of the algorithm's behavior. Unsurprisingly, a considerable number of approaches to coreset discovery can be found in the specialized literature. In the following, a few of the main algorithms in the field, that will be used as a reference during the experiments, are briefly summarized: Greedy Iterative Geodesic Ascent (GIGA) [2], Frank-Wolfe (FW) [3], Forward Stagewise (FSW) [4], Least-angle regression (LAR) [5] [6], Matching Pursuit (MP) [7] and Orthogonal Matching Pursuit (OMP) [8].

BLR is based on the idea that finding the optimal coreset is too expensive. In order to overcome this issue, the authors use a k-clustering algorithm to obtain a compact representation of the data set. In particular, they claim that samples that are bunched together could be represented by a smaller set of points, while samples that are far from other data have a larger effect on inferences. Therefore, the BLR coreset is composed of few samples coming from tight clusters plus the outliers.

The original FW algorithm applies in the context of maximizing a concave function within a feasible polytope by means of a local linear approximation. Section 4 refers to the Bayesian implementation of the FW algorithm designed for core set discovery. This technique, described in [11], aims to find a linear combination of approximated likelihoods (which depends on the core set samples) that is similar to the full likelihood as much as possible.

GIGA is a greedy algorithm that further improves FW. In [2], the authors show that computing the residual error between the full and the approximated likelihoods by using a geodesic alignment guarantees a lower upper bound to the error at the same computational cost.

FSW [4], LAR [5] [6], MP [7] and OMP [8] were all originally devised as greedy algorithms for dimensionality reduction. The simplest is FSW which projects high-dimensional data in a lower dimensional space by selecting one at a time the feature whose inclusion in the model gives the most statistically significant improvement. MP instead includes features having the highest inner product with a target signal, while its improved version OMP at each step carries an orthogonal projection out. Similarly, LAR increases the weight of each feature in the direction equiangular to each one's correlations with the target signal. All these procedures could be applied to the transpose problem of feature selection, that is approximation of core sets. Often these algorithms start from the assumption that the coreset for a given dataset will be independent from the ML pipeline used, but this premise might not always be correct. For example, the optimization problem underlying a classification task might vary considerably depending on the ML algorithm used.

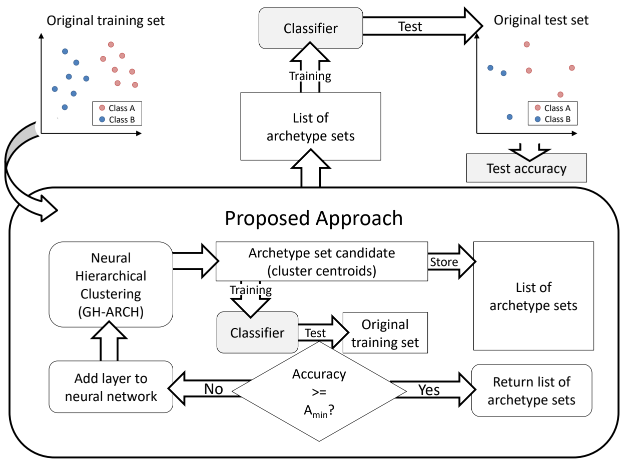


Fig. 1: Scheme of the proposed approach. A_{min} is a user-defined parameter.

3 GH-ARCH

The proposed approach exploits neuron weight vectors of a clustering network as a set of virtual points, *archetypes*. These can better and more concisely represent the entire dataset rather than a single set of real data points. Neurons, in fact, are placed in such a way to provide the best topological representation of data and each neuron summarizes the information provided by its Voronoi set, the compact set of points assigned to it.

The use of a hierarchical divisive neural network makes it possible to provide at the same time multiple archetypes set at different level of resolution - each one corresponding to a level of the network. In fact, basic clustering algorithms are capable of producing only a single representation of the data. Furthermore, the proposed neural network, GH-ARCH, automatically suggests the best representation to be used by a given classifier. After finding an entire level of neurons (archetype set), GH-ARCH trains the classifier on the weight vectors of the leaf neurons: in case the accuracy overcomes A_{min}, the minimum accuracy archetype sets needs to satisfy (user-defined parameter), the algorithm returns the current list of archetypes sets; otherwise, the current set of archetypes is inserted in the list and a further layer is added to the network.

As previously introduced, GH-ARCH is a modified version of the GH-EXIN algorithm, devised ad-hoc for archetype set discovery. The major difference between the two algorithms consists in the final goal of the network and on the indices to be minimized: while GH-EXIN is a network which focuses on finding biclusters and minimize a biclustering quantization index, GH-ARCH attempts to minimize the heterogeneity and maximize the purity of the clusters, in order to group points which are both close to each other and belonging to the same class. The way in which data is divided at deeper and deeper levels, on the other hand, follows the same algorithm and it is briefly explained in the following. The overall proposed approach is summarized in Fig. 1.

3.1 Neural Clustering

Neural clustering algorithms build a hierarchical (divisive) tree whose vertices correspond to its neurons, as shown in Fig. 2. The architecture of the network is data-driven, the number of neurons and the number of layers are automatically determined according to the data. Each neuron is associated to a weight vector whose dimensionality corresponds to the input space. For each father neuron, a neural network is trained on its set of data (Voronoi set). The related cardinality is shown in Fig. 2, next to each vertex. The children nodes are the neurons of the associated neural network, and define a partition of the father Voronoi set. For each leaf, the procedure is repeated. The initial structure of the neural network is always a pair of neurons (seed), linked by an edge, with age set to zero. Multiple node creation and pruning determines the correct number of neurons of each network. For further details regarding GH-EXIN neuron creation and deletion we remind to [12] and [13] were it has been fully explained and compared with other SOA algorithms.

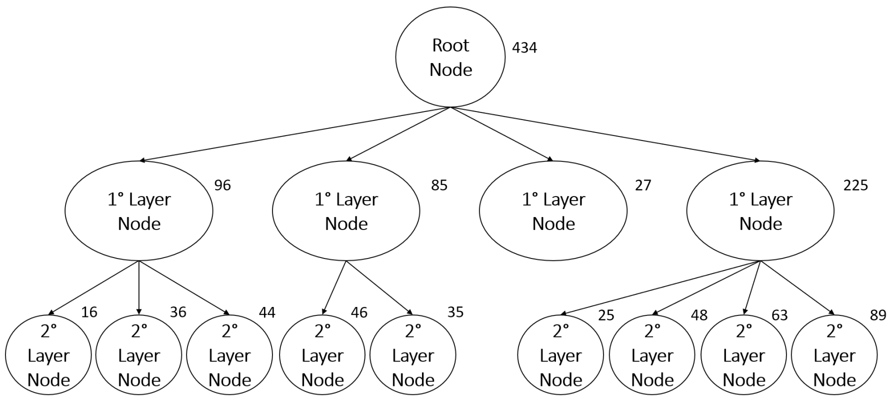
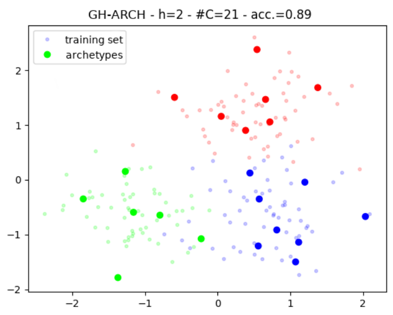
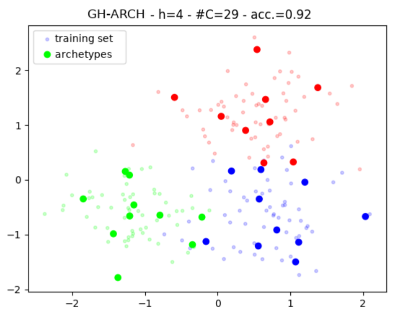


Fig. 2: GH-ARCH tree

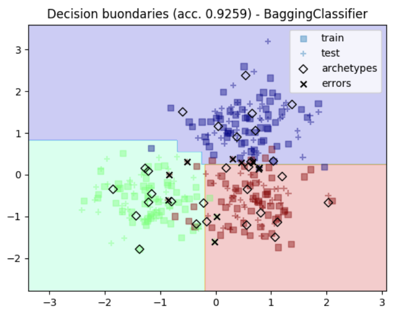
After repeating this procedure for all the leaves of a single layer, recalling Fig. 1, a given classifier is trained on the weight vectors of the leaves found by GH-ARCH so far. As previously stated, in case the accuracy obtained on a test set is higher than A_{min}, the current list of archetypes is returned. Nonetheless, the algorithm stops also in case there are no leaves to be expanded: this may occur in case the current purity and heterogeneity of all leaves are already high. Lastly, in case points grouped by a leaf do not belong to the same class, the label of the archetype of that leaf is assigned by means of a majority voting procedure.



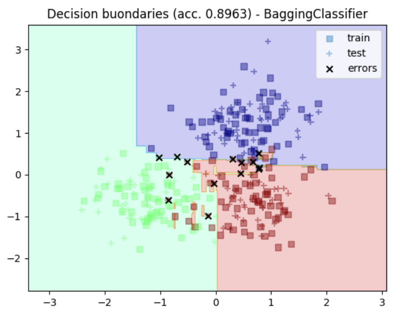
(a) Archetypes extracted by GH-ARCH at the 2^{nd} level of the network



(b) Archetypes extracted by GH-ARCH at the 4^{th} level of the network



(c) Decision boundaries using the archetypes.



(d) Decision boundaries using the whole training set.

Fig. 3: GH-ARCH on the Blobs dataset using the Bagging classifier.

4 Experimental results

All the experiments presented in this section exploit 4 classifiers, representative of both hyperplane-based and ensemble, tree-based classifiers: `Bagging` [14], `RandomForest` [15], `Ridge` [16], and `SVC` (Support Vector Machines) [17]. All classifiers are implemented in the `scikit-learn`[6] [18] Python module and use default parameters. For the sake of comparison, it is important that the classifier will follow the same training steps, albeit under different conditions. For this reason, a fixed seed has been set for all those that exploit pseudo-random elements in their training process.

The experiments are performed on popular benchmarks available in the `scikit-learn` package: i. Blobs, three isotropic gaussian blobs (3 classes, 400 samples, 2 features); ii. Circles, a large circle containing a smaller one (2 classes,

[6] scikit-learn: Machine Learning in Python, `http://scikit-learn.org/stable/`

Table 1: Blobs dataset. Coreset size, accuracy on test set and running time (seconds) of the considered classifiers and coreset algorithms.

		RandomForest			Bagging			SVC			Ridge		
algorithm	core type	size	accuracy	avg time	size	accuracy	avg time	size	accuracy	avg time	size	accuracy	avg time
all samples		265	**0.9185**		265	0.8963		265	**0.9407**		265	0.8963	
GH-ARCH	4 level	29	**0.9185**	0.9	29	**0.9259**	0.9	27	0.9259	1.0	30	0.8889	0.8
	3 level	27	**0.9185**		27	0.9185		25	0.9185		26	**0.9185**	
GIGA		3	0.6296	0.01	3	0.8889	0.01	3	0.8889	0.01	3	0.8519	0.01
FW		4	0.5185	3	4	0.6593	3	4	0.5852	3	4	0.8815	3
MP		5	0.4047	4	5	0.6296	4	5	0.3333	4	5	0.5778	4
FS		5	0.7481	4	5	0.7481	4	5	0.3333	4	5	0.6296	4
OP		4	0.4074	0.01	4	0.6148	0.01	4	0.3333	0.01	4	0.5852	0.01
LAR		3	0.8074	0.01	3	0.8074	0.01	3	0.5185	0.01	3	0.5185	0.01

Table 2: Circles dataset. Coreset size, accuracy on test set and running time (seconds) of the considered classifiers and coreset algorithms.

		RandomForest			Bagging			SVC			Ridge		
algorithm	core type	size	accuracy	avg time	size	accuracy	avg time	size	accuracy	avg time	size	accuracy	avg time
all samples		266	**0.9552**		266	**0.9478**		266	**0.9851**		266	0.5000	
GH-ARCH	4 level	23	0.9254	0.8	23	0.9403	0.8	26	**0.9851**	1.0	26	0.5000	0.8
	3 level	15	0.6418		15	0.5970		24	0.9776		25	0.5000	
GIGA		2	0.5970	0.01	2	0.5746	0.01	2	0.6343	0.01	2	0.6364	0.01
FW		5	0.5597	3	5	0.5000	3	5	0.5000	3	5	0.5000	3
MP		3	0.5000	4	3	0.5224	4	3	0.5000	4	3	**0.6567**	4
FS		4	0.6567	4	4	0.6194	4	4	0.6119	4	4	0.6269	4
OP		3	0.5000	0.01	3	0.4851	0.01	3	0.6418	0.01	3	0.5448	0.01
LAR		2	0.5522	0.01	2	0.6194	0.01	2	0.5970	0.01	2	0.5970	0.01

400 samples, 2 features); iii. Moons, two interleaving half circles (2 classes, 400 samples, 2 features); iv. Iris [19] (3 classes, 150 samples, 4 features); v. Digits [20] (10 classes, 1797 samples, 64 features). The samples are randomly split between a 66%-sample training set and a 33%-sample test set. The code used in this work is freely available in the BitBucket repository `https://bitbucket.org/evomlteam/evolutionary-archetypes`.

The results obtained by the proposed approach are then compared against the 6 coreset discovery algorithms GIGA [2], FW [3], MP [8], OMP [8], LAR [5] [6], and FSW [4], described in more detail in section 2. The comparison is performed

Table 3: Moons dataset. Coreset size, accuracy on test set and running time (seconds) of the considered classifiers and coreset algorithms.

		RandomForest			Bagging			SVC			Ridge		
algorithm	core type	size	accuracy	avg time	size	accuracy	avg time	size	accuracy	avg time	size	accuracy	avg time
all samples		266	**0.9328**		266	**0.9254**		266	**0.9179**		266	**0.8134**	
GH-ARCH	3 level	28	0.8209	0.8	26	0.7687	1.0	23	0.8358	0.7	26	0.7687	0.9
	2 level	17	0.7313		17	0.8358		16	0.8433		17	0.7985	
GIGA		2	0.4254	0.01	2	0.2463	0.01	2	0.4701	0.01	2	0.4701	0.01
FW		6	0.6493	3	6	0.6493	3	6	0.5299	3	6	0.6866	3
MP		3	0.5149	4	3	0.5821	4	3	0.5896	4	3	0.6642	4
FS		2	0.5149	4	2	0.2313	4	2	0.6119	4	2	0.6119	4
OP		2	0.5149	0.01	2	0.2463	0.01	2	0.6493	0.01	2	0.6493	0.01
LAR		3	0.5149	24	3	0.2388	24	3	0.5224	24	3	0.5896	24

Table 4: Iris dataset. Coreset size, accuracy on test set and running time (seconds) of the considered classifiers and coreset algorithms.

		RandomForest			Bagging			SVC			Ridge		
algorithm	core type	size	accuracy	avg time	size	accuracy	avg time	size	accuracy	avg time	size	accuracy	avg time
all samples		99	**0.9608**		99	**0.9608**		99	0.9412		99	**0.8824**	
GH-ARCH	3 level	23	0.9412	0.2	23	0.9412	0.2	23	0.9020	0.1	23	0.8431	0.1
	2 level	14	0.9412		14	0.9216		14	0.8824		14	0.8431	
GIGA		7	0.9216	0.01	7	0.6667	0.01	7	**0.9804**	0.01	7	0.8431	0.01
FW		15	0.8824	3	15	0.8627	3	15	0.9412	3	15	0.8235	3
MP		14	0.9412	4	14	0.8627	4	14	0.9216	4	14	0.7255	4
FS		7	0.6667	4	7	0.7059	4	7	0.6471	4	7	0.6275	4
OP		5	0.7059	0.01	5	0.5294	0.01	5	0.7843	0.01	5	0.8235	0.01
LAR		4	0.5294	22	4	0.6863	22	4	0.6471	22	4	0.7059	22

on three metrics: i. coreset size (lower is better); ii. classification accuracy on the test set (higher is better); iii. running time of the algorithm (lower is better).

Tables 1, 2, 3, and 4 present the performance of the proposed approach against state-of-the-art algorithms for coreset discovery in ML literature. Tables are divided in columns according to the classifier used. On the first row, the accuracy reached on whole original dataset is also reported.

With regard to test accuracy, GH-ARCH not only outperforms the other techniques by far, but it is sometimes able to increase the performance obtained training the same classifier with all the training samples available. This means that the decision boundaries generated by the classifier using the neural archetypes may generalize even better than those generated using the whole training set, as shown in figure 3. These results suggest that the performance of ML classifiers is not just a function of the *size* of the training set (as Big Data and Deep Learning often claim) but also a function of the *mutual position* of the training samples in the feature space. By exploiting the original training set and by relaxing the constraint of sample positions, GH-ARCH generates a new, smaller data set suited for each classifier in order to provide the best generalization ability.

Overall, the number of points used by GH-ARCH is generally higher compared to the other state-of-the-art algorithms. Still, the number of data points used is generally an order of magnitude smaller than the original training set. The time taken by all algorithms is also comparable, with GH-ARCH resulting among the fastest algorithms on all datasets but Digits. At last, sometimes the final archetype set may not be the best set found by the network. As shown in Table 3, the archetype set found at the 3^{rd} level performs worse than the one found at the 2^{nd} level for most classifier. For this reason, GH-ARCH returns the entire list of possible archetypes set produced by each layer, together with its accuracy, so that a human expert may choose the preferred configuration.

In Fig. 3, the process of creation of the archetypes can be observed at different levels of the network and how these points may also outperform the whole training set. GH-ARCH, after the second level of training (Fig 3a already distributes correctly the neurons - i.e. the archetypes - among the three classes. Nonetheless, two layers below (Fig. 3b), GH-ARCH increases the number of neurons ($21 \to 29$) in particular along the decision boundaries, thus improving the overall accuracy of the classifier ($90 \to 92$). Of notable importance is also the fact that GH-ARCH is capable of recognizing outliers, reducing the noise of the dataset present in Fig. 3d which causes a bad classification when using the whole training set.

5 Conclusions

Coreset discovery is a research line of utmost practical importance, and several techniques are available to find the most informative data points in a given training set. Limiting the search to existing points, however, might impair the final objective, that is, finding a set of points able to summarize the information contained in the original dataset. This work introduced the concept of *archetypes*, virtual data points that can be used in place of points of a coreset.

Hierarchical clustering, based on a novel neural network architecture (GH-EXIN), is used to find meaningful archetype sets, corresponding to the estimated levels; these sets can be exploited to train a target classifier for a given dataset.

Experimental results on popular benchmarks show that the proposed approach outperforms state-of-the-art core discovery techniques in literature on accuracy, generality, and computational time.

Future work will extend the current results to regression problems, and explore new methodologies to improve the archetype sets even further.

References

1. O. Bachem, M. Lucic, and A. Krause, “Practical coreset constructions for machine learning,” *arXiv preprint arXiv:1703.06476*, 2017.
2. T. Campbell and T. Broderick, “Bayesian Coreset Construction via Greedy Iterative Geodesic Ascent,” in *International Conference on Machine Learning (ICML)*, 2018. [Online]. Available: https://arxiv.org/pdf/1802.01737.pdf
3. K. L. Clarkson, “Coresets, Sparse Greedy Approximation, and the Frank-Wolfe Algorithm,” in *ACM Transactions on Algorithms*, 2010. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.145.9299{&}rep=rep1{&}type=pdf
4. E. M. A., “Multiple Regression Analysis,” *Mathematical Methods for Digital Computers*, 1960.
5. B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani, “Least Angle Regression,” *The Annals of Statistics*, vol. 32, no. 2, pp. 407–451, 2004. [Online]. Available: https://arxiv.org/pdf/math/0406456.pdf
6. C. Boutsidis, P. Drineas, and M. Magdon-Ismail, “Near-optimal Coresets For Least-Squares Regression,” Tech. Rep., 2013. [Online]. Available: https://arxiv.org/pdf/1202.3505.pdf
7. S. Mallat and Z. Zhang, “Matching pursuits with time-frequency dictionaries,” *IEEE Transactions on Signal Processing*, vol. 42, no. 12, p. 33973415, 1993.
8. Y. Pati, R. Rezaiifar, and P. Krishnaprasad, “Orthogonal matching pursuit: recursive function approximation with applications to wavelet decomposition,” *Proceedings of 27th Asilomar Conference on Signals, Systems and Computers*, pp. 40–44, 1993. [Online]. Available: http://ieeexplore.ieee.org/document/342465/

9. P. Barbiero, G. Ciravegna, E. Piccolo, G. Cirrincione, M. Cirrincione, and A. Bertotti, "Neural biclustering in gene expression analysis," in *2017 International Conference on Computational Science and Computational Intelligence (CSCI)*, Dec 2017, pp. 1238–1243.
10. I. W. Tsang, J. T. Kwok, and P.-M. Cheung, "Core vector machines: Fast svm training on very large data sets," *Journal of Machine Learning Research*, vol. 6, no. Apr, pp. 363–392, 2005.
11. T. Campbell and T. Broderick, "Automated Scalable Bayesian Inference via Hilbert Coresets," 2017. [Online]. Available: http://arxiv.org/abs/1710.05053
12. G. Cirrincione, G. Ciravegna, P. Barbiero, V. Randazzo, and E. Pasero, "The gh-exin neural network for hierarchical clustering," *Neural Networks*, vol. 121, pp. 57 – 73, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608019302060
13. G. Ciravegna, G. Cirrincione, F. Marcolin, P. Barbiero, N. Dagnes, and E. Piccolo, *Assessing Discriminating Capability ofGeometrical Descriptors for 3D Face Recognition by Using the GH-EXIN Neural Network.* Singapore: Springer Singapore, 2020, pp. 223–233. [Online]. Available: https://doi.org/10.1007/978-981-13-8950-4_21
14. L. Breiman, "Pasting small votes for classification in large databases and on-line," *Machine Learning*, vol. 36, no. 1-2, pp. 85–103, 1999.
15. ——, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.
16. A. N. Tikhonov, "On the stability of inverse problems," in *Dokl. Akad. Nauk SSSR*, vol. 39, no. 5, 1943, pp. 195–198.
17. M. A. Hearst, S. T. Dumais, E. Osman, J. Platt, and B. Scholkopf, "Support vector machines," *IEEE Intelligent Systems and their Applications*, vol. 13, no. 4, pp. 18–28, 1998.
18. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
19. R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Annals of eugenics*, vol. 7, no. 2, pp. 179–188, 1936.
20. D. Dheeru and E. Karra Taniskidou, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml