Generating Neural Archetypes to Instruct Fast and Interpretable Decisions

Pietro Barbiero[1(✉)], Gabriele Ciravegna[2], Giansalvo Cirrincione[3],
Alberto Tonda[4], and Giovanni Squillero[1]

[1] Politecnico di Torino, Torino, Italy
pietro.barbiero@studenti.polito.it, giovanni.squillero@polito.it
[2] University of Siena, Siena, Italy
gabriele.ciravegna@unifi.it
[3] University of South Pacific, Suva, Fiji
nimzoexin59@gmail.com
[4] Université Paris-Saclay, INRA, UMR 782 GMPA,
78850 Thiverval-Grignon, France
alberto.tonda@inra.fr

Abstract. In the field of artificial intelligence, agents learn how to take decisions by fitting their parameters on a set of samples called training set. Similarly, a *core set* is a subset of the training samples such that, if an agent exploits this set to fit its parameters instead of the whole training set, then the quality of the inferences does not change significantly. Relaxing the constraint that restricts the search for core sets to the available data, neural networks may be used to generate virtual samples, called *archetype set*, containing the same kind of information. This work illustrates the features of GH-ARCH, a recently proposed self-organizing hierarchical neural network for archetype discovery. Experiments show how the use of archetypes allows both ML agents to make fast and accurate predictions and human experts to make sense of such decisions by analyzing few important samples.

Keywords: Archetypes · Big data · Classification · Coresets · Explain AI · GH-ARCH · Hierarchical clustering · Machine learning · Neural networks · Self-organization · Semi-supervised learning

1 Introduction

In recent years, the development of modern technologies and infrastructures (such as internet of things, high-performance computing, and GPUs) as well as novel programming paradigms (e.g. parallelization) has led to a new era in knowledge discovery. As large sets of samples allow for estimating the parameters of very complex models, suddenly some intractable problems become easy to deal with. Among artificial intelligence disciplines, machine learning has captured most of the interest in scientific communities as it provides a set of models

E. Bucciarelli et al. (Eds.): DECON 2019, AISC 1009, pp. 45–52, 2020.
https://doi.org/10.1007/978-3-030-38227-8_6

with large capacity (i.e. number of parameters) suitable for almost any kind of data sets. On the other hand, some researchers are expressing doubts about the effective exploitation of machine learning in real contexts [1,2]. In fact, if on one side the capacity is the main reason of their success, on the other side the large number of parameters makes these models nearly impossible to be interpreted. In some real contexts, such as health-care and economics, it is important not only to take the right decisions, but also to explain the reason why they are better than others. Machine learning algorithms learn to make predictions by fitting their parameters using a set of samples (i.e. the training set). However, as for human beings, during the learning phase ML agents tend to give more weight to some training samples than others, as their features are more useful in estimating the model parameters. For highly complex models, the analysis of such key samples may be useful for human experts to have an insight on how agents are reasoning. As the size of databases is increasingly large, searching manually for fundamental samples has become an intractable problem. As a result, in the last few years, ML researchers and big data experts have proposed several algorithms for the extraction of fundamental sets of data, called *core sets*, from sizable databases. More concretely, a core set can be defined as the subset of the training samples of minimal size that is required for a given algorithm to provide good results, even as good as it would have if trained on the whole training set [3]. In other words, a core set summarizes the information contained in a data set, with the constraint that each core sample belongs to the training set. However, it is reasonable that better summaries can be discovered if such constraint is removed. If core samples are allowed to be outside the set of training data they are named *archetype samples*. Following this idea, the Growing Hierarchical Archetype (GH-ARCH) algorithm has been recently proposed for archetype discovery [4]. GH-ARCH is a neural network which builds an incremental and self-organized tree performing hierarchical clustering. For each layer, the final positions of neurons in the feature space represent the virtual set of points corresponding to the archetype set. The hierarchical structure of the neural network allows the user for the selection of archetype sets of different size. In the following, the GH-ARCH neural network is used to extract archetypes for different ML classifiers. Results of experiments described in Sect. 3 show how the use of archetypes allows both ML classifiers to make fast and accurate predictions and human experts to make sense of such decisions by analyzing few important samples.

2 GH-ARCH

The Growing Hierarchical Archetype algorithm (GH-ARCH) is a recently proposed neural network for archetype discovery. Most of complex problems can be seen in a multi-resolution way where few key concepts are represented in higher levels and finer details are shown in deeper layers. For this reason, hierarchical algorithms, as GH-ARCH, are popular and effective techniques for extracting understandable information from data, as human experts can make sense of key aspects of the problem by having a look at few key concepts. Besides, GH-ARCH

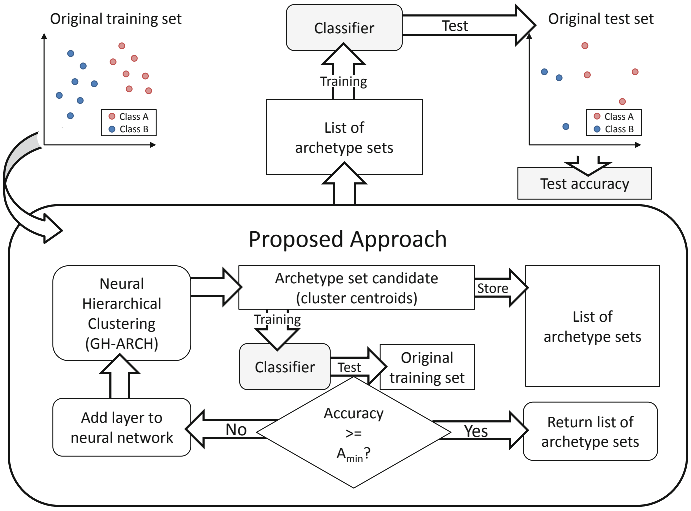

Fig. 1. Scheme of GH-ARCH for archetype extraction. A_{min} is a user-defined parameter.

is a data driven (self-organization) and incremental approach in the sense that both the number of neurons and their position in the feature space are automatically estimated from data. The technique described in this section directly derives from the Growing Hierarchical EXIN (GH-EXIN) algorithm [5], a neural network for hierarchical clustering. In both cases, neurons can be seen as representative prototypes of clusters as they are placed in such a way to provide the best topological representation of the data distribution. Therefore, once the positions of such prototypes is estimated from data, they can be interpreted as an *archetype set* of representative virtual samples. Figure 1 visually describes how GH-ARCH is used for archetype discovery. The major difference between the two algorithms consists in the final goal of the algorithm 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 explained in the following. For each father neuron, a neural network is trained on its corresponding Voronoi set (set of data represented by the father neuron). The children nodes are the neurons of the associated neural network, and determine a subdivision of the father Voronoi set. For each leaf, the procedure is repeated. The initial structure of the neural network is a seed, i.e. a pair of neurons, which are linked by an edge, whose age is set to zero. Multiple node creation and pruning determines the correct number of neurons of each network. For each epoch (presentation in a random way of the whole training set to the network) the basic iteration starts at the presentation of a new data point, say x_i. All neurons are ranked according to the Euclidean distances between x_i and their weights. The neuron with the shortest distance is the winner w_1. If its distance is larger than the scalar threshold of the neuron (novelty test), a new neuron is created with weight vector given by x_i. The initial weight vectors are heuristically defined as the average feature values of the points in the Voronoi set, and the neural thresholds are given by the mean distance among the same points. Otherwise, there is a weight adaptation and the creation of an edge The weight computation (training) is based on the Soft Competitive Learning (SCL) [6] paradigm, which requires a winner-takes-most strategy: at each iteration, both the winner and its neighbors change their weights but in different ways: w_1 and its direct topological neighbors are moved towards x_i by fractions α_1 and α_n (learning rates), respectively, of the vector connecting the weight vectors to the datum. This law requires the determination of a topology (neighbors) which is achieved by the Competitive Hebbian Learning (CHL) rule [6], used for creating the neuron connections: each time a neuron wins, an edge is created, linking it to the second nearest neuron, if the link does not exist yet. If there was an edge, its age is set to zero and the same age procedure as in [7] is used as follows. The age of all other links emanating from the winner is incremented by one; during this process if a link age is greater than the *agemax* scalar parameter, it is eliminated (pruning). The thresholds of the winner and second winner are recomputed as the distance to their farthest neighbor. At the end of each epoch, if a neuron remains unconnected (no neighbors), it is pruned, but the associated data points are analyzed by a new ranking of all the neurons of the network (i.e. also the neurons of the neural networks of the other leaves of the hierarchical tree). If it is outside the threshold of the new winner, it is labeled as an outlier and pruned. If, instead, it is inside, it is assigned to the winner Voronoi set. Each leaf neural network is controlled by the purity, calculated as

$$P = \max_{i \in C} \{c_i\} \tag{1}$$

where c_i is the number of elements belonging the class i within the Voronoi set of the leaf and C is the number of classes; and by the heterogeneity, calculated as the sum of the Euclidean distances between the neuron (w_γ) and the N data composing its Voronoi set (x_i):

$$H = \sum_{i=1}^{N} ||w_\gamma - x_i||^2 \tag{2}$$

In particular, the training epochs are stopped when the estimated value of these parameters falls below a percentage of the value for the father leaf. This technique creates a vertical growth of the tree. The horizontal growth is generated by the neurons of each network. However, a simultaneous vertical and horizontal growth is possible. At the end of a training, the graphs created by the neuron edges are checked. If connected subgraphs are detected, each sub-graph is considered as a father, by estimating the centroid of the cluster (vertical growth) and the associated neurons as the corresponding sons (horizontal growth). This last step ends the neural clustering on the Voronoi set of one leaf to be expanded. The decision on whether to expand a leaf is again based on the purity and the heterogeneity of that leaf. In case either the purity of the leaf is lower than P_{min} or the heterogeneity is higher than H_{max} (user-dependent parameter), the node is labelled as *parent_node* and a further neural clustering is run on its Voronoi set. 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. 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 when the current purity and heterogeneity of all leaves are already high. Lastly, if 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.

3 Experimental Results

Both the experiments presented in this section can be reproduced using our code freely available on Bitbucket[1]. All the used classifiers are implemented in the *scikit-learn* [8] Python module and use default parameters. For the sake of reproducibility, a random seed is set for all the algorithms exploiting pseudo-random elements.

3.1 Understanding Archetypes

In order to understand at a glance the importance of archetypes, GH-ARCH is first applied to a synthetic data set called Blobs. It is composed of three isotropic 2-dimensional gaussian distributions, each one representing a different class. In Fig. 2, four archetype sets are shown at different resolution levels, corresponding to the 2nd and the 3rd layer of GH-ARCH. These virtual sets of samples are used to train *RandomForest* [9] and *Ridge* [10] classifiers in place of the whole training set. Observe how even using 3–4 samples the accuracy of predictions is comparable with the one obtained exploiting the whole training set. Besides, archetypes in deeper layers of GH-ARCH represent the data set distribution in more detail. Finally, notice how archetypes seem to have a regularization effect on tree-based classifiers like *RandomForest*, as the corresponding decision boundaries are smoother than the ones obtained with the whole training set.

3.2 Making Sense of Archetypes and Decisions in Economics

In order to show how to exploit GH-ARCH in a real setting, the proposed approach is applied to the *Car* data set [11], containing the information of 406 cars produced in USA or in Europe/Japan. Cars are characterized considering both architectural features (e.g. number of cylinders and weight) and

[1] https://bitbucket.org/neurocoreml/archetypical-neural-coresets/.

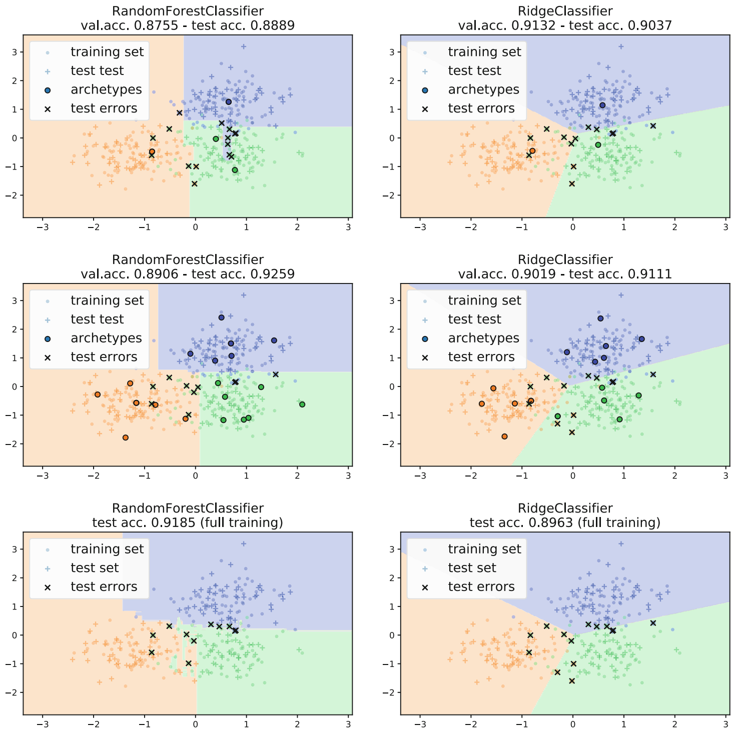

Fig. 2. GH-ARCH on the Blobs dataset using RandomForest (**left**) and Ridge (**right**) classifiers. Archetypes in the first and the second row corresponds to the ones of the 2nd and the 3rd hierarchical level of GH-ARCH, respectively. The last row show the decision boundaries obtained using all the training samples to fit model parameters.

Table 1. *Cars* data set. Training set size, classification accuracy on an unseen test set and running time (in seconds) for different classifiers exploiting both GH-ARCH and state-of-the-art algorithms for core set discovery.

Algorithm	RandomForest			Bagging			LogisticRegression			Ridge		
	Size	Accuracy	Avg time	Size	Accuracy	Avg time	Size	Accuracy	Avg time	Size	Accuracy	Avg time
All samples	270	0.8824		270	0.9338		270	0.8971		270	0.8750	
GH-ARCH (2L)	4	0.7868	0.06	4	0.7721	0.07	4	0.8235	0.03	4	0.8162	0.03
GH-ARCH (5L)	76	0.8971	0.20	76	0.8456	0.17	66	0.8309	0.15	76	0.8382	0.14
GIGA	27	0.6985	0.15	27	0.7426	0.15	27	0.6838	0.15	27	0.6250	0.15
FW	35	0.7206	0.63	35	0.7279	0.63	35	0.6912	0.63	35	0.6618	0.63
MP	26	0.6250	0.57	26	0.6250	0.57	26	0.6324	0.57	26	0.6324	0.57
FS	17	0.6250	0.62	17	0.6250	0.62	17	0.6250	0.62	17	0.6250	0.62
OP	6	0.6324	0.05	6	0.6176	0.05	6	0.6544	0.05	6	0.6471	0.05
LAR	8	0.6250	0.01	8	0.6250	0.01	8	0.6912	0.01	8	0.6324	0.01

the production year. Four ML classifiers are trained on GH-ARCH archetypes to predict the region of origin (USA vs not-USA): *Bagging* [12], *RandomForest* [9], *Ridge* [10], and *LogisticRegression* [13]. The results obtained exploiting

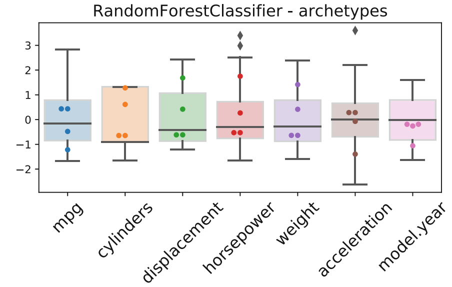

Fig. 3. Boxplots displaying the mean and the standard deviation of the training distribution for each feature. On top of boxplots, swarmplots show the archetype set extracted in the 2nd layer of GH-ARCH for *RandomForest*.

Table 2. Archetype set of extracted in the 2nd layer of GH-ARCH for *RandomForest*.

Features	A1	A2	A3	A4
mpg	26.4	19.7	14.3	26.4
Cylinders	1.4	3.0	3.9	1.4
Displacement	130.7	236.2	364.5	130.7
Horsepower	86.9	115.9	170.0	86.9
Weight	2438.5	3314.9	4144.8	2438.5
Acceleration	16.0	15.1	11.7	16.0
Model year	75.4	75.1	72.1	75.3
Target	EU/JP	USA	USA	EU/JP

GH-ARCH are then compared against the 6 coreset discovery algorithms GIGA [14], FW [15], MP [16], OMP [16], LAR [17,18], and FSW [19]. The comparison is performed 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). Table 1 summarizes the results obtained for each ML classifier. With regard to the accuracy on an unseen test set, classifiers trained using archetypes extracted in the 5th layer of GH-ARCH are comparable with the ones trained using the whole training set. In order to show how archetypes can be useful in interpreting model decisions, we manually analyzed the archetypes extracted in the 2nd hierarchical layer of GH-ARCH. Figure 3 and Table 2 show in two different ways (graphical and tabular) the archetype set found. Observe how the American archetypes A2 and A3 are very different from the European/Japanese. More in detail, EU/JP vehicles seem more ecological, as they have a better miles per gallon (mpg) ratio. American cars, instead, appear to be more powerful as they have higher values for number of cylinders, engine displacements (in cubic centimeters), horsepower, and weight (in lbs.). Summarizing, the use of archetypes allows ML agents, such *RandomForest*, to make fast and accurate predictions and allows human experts to make sense of such decisions by analyzing few important samples.

4 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. In this work, hierarchical clustering, based on a novel neural network architecture (GH-EXIN), is used to find meaningful archetype sets, virtual but representative data points. Results on a real economic dataset shows how archetypes may be useful in explaining decisions taken by machine learning classifiers.

References

1. Doshi-Velez, F., Kim, B.: Towards a rigorous science of interpretable machine learning. arXiv preprint arXiv:1702.08608 (2017)
2. Mitchell Waldrop, M.: News feature: what are the limits of deep learning? Proc. Natl. Acad. Sci. **116**(4), 1074–1077 (2019)
3. Bachem, O., Lucic, M., Krause, A.: Practical coreset constructions for machine learning. arXiv preprint arXiv:1703.06476 (2017)
4. Ciravegna, G., Barbiero, P., Cirrincione, G., Squillero, G., Tonda, A.: Discovering hierarchical neural archetype sets. In: The International Joint Conference on Neural Networks (IJCNN), July 2019
5. Cirrincione, G., Ciravegna, G., Barbiero, P., Randazzo, V., Pasero, E.: The GH-EXIN neural network for hierarchical clustering. Neural Networks (2019, under peer review)
6. Haykin, S.S.: Neural Networks and Learning Machines, vol. 3. Pearson, Upper Saddle River (2009)
7. Cirrincione, G., Randazzo, V., Pasero, E.: The growing curvilinear component analysis (GCCA) neural network. Neural Netw. **103**, 108–117 (2018)
8. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
9. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001)
10. Tikhonov, A.N.: On the stability of inverse problems. Dokl. Akad. Nauk SSSR **39**, 195–198 (1943)
11. Vanschoren, J., van Rijn, J.N., Bischl, B., Torgo, L.: OpenML: networked science in machine learning. SIGKDD Explorations **15**(2), 49–60 (2013)
12. Breiman, L.: Pasting small votes for classification in large databases and on-line. Mach. Learn. **36**(1–2), 85–103 (1999)
13. Cox, D.R.: The regression analysis of binary sequences. J. Roy. Stat. Soc.: Ser. B (Methodol.) **20**, 215–242 (1958)
14. Campbell, T., Broderick, T.: Bayesian coreset construction via greedy iterative geodesic ascent. In: International Conference on Machine Learning (ICML) (2018)
15. Clarkson, K.L.: Coresets, sparse greedy approximation, and the Frank-Wolfe algorithm. ACM Trans. Algorithms **6**, 63 (2010)
16. Pati, Y.C., Rezaiifar, R., Krishnaprasad, P.S.: Orthogonal matching pursuit: recursive function approximation with applications to wavelet decomposition. In: Proceedings of 27th Asilomar Conference on Signals, Systems and Computers, pp. 40–44 (1993)
17. Efron, B., Hastie, T., Johnstone, I., Tibshirani, R.: Least angle regression. Ann. Stat. **32**(2), 407–451 (2004)
18. Boutsidis, C., Drineas, P., Magdon-Ismail, M.: Near-optimal coresets for least-squares regression. Technical report (2013)
19. Efroymson, M.A.: Multiple regression analysis. Math. Methods Digital Comput. (1960)