RESEARCH-ARTICLE

Evolutionary discovery of coresets for classification

PIETRO BARBIERO, Polytechnic of Turin, Turin, TO, Italy

GIOVANNI SQUILLERO, Polytechnic of Turin, Turin, TO, Italy

ALBERTO PAOLO TONDA, Paris-Saclay University, Gif-sur-Yvette, Ile-de-France, France

Published: 13 July 2019

Citation in BibTeX format

GECCO '19: Genetic and Evolutionary Computation Conference
July 13 - 17, 2019
Prague, Czech Republic

Conference Sponsors:
SIGEVO

Evolutionary Discovery of Coresets for Classification

Pietro Barbiero
Politecnico di Torino
Torino, Italy
pietro.barbiero@studenti.polito.it

Giovanni Squillero
Politecnico di Torino
Torino, Italy
giovanni.squillero@polito.it

Alberto Tonda
INRA, Université Paris-Saclay
Thiverval-Grignon, France
alberto.tonda@inra.fr

ABSTRACT

When a machine learning algorithm is able to obtain the same performance given a complete training set, and a small subset of samples from the same training set, the subset is termed *coreset*. As using a coreset improves training speed and allows human experts to gain a better understanding of the data, by reducing the number of samples to be examined, coreset discovery is an active line of research. Often in literature the problem of coreset discovery is framed as i. *single-objective*, attempting to find the candidate coreset that best represents the training set, and ii. *independent* from the machine learning algorithm used. In this work, an approach to evolutionary coreset discovery is presented. Building on preliminary results, the proposed approach uses a multi-objective evolutionary algorithm to find compromises between two conflicting objectives, i. minimizing the number of samples in a candidate coreset, and ii. maximizing the accuracy of a target classifier, trained with the coreset, on the whole original training set. Experimental results on popular classification benchmarks show that the proposed approach is able to identify candidate coresets with better accuracy and generality than state-of-the-art coreset discovery algorithms found in literature.

CCS CONCEPTS

• **Computing methodologies → Supervised learning**; **Genetic algorithms**;

KEYWORDS

Classification; Coreset discovery; Evolutionary algorithms; Explain AI; Machine learning; Multi-objective

ACM Reference format:
Pietro Barbiero, Giovanni Squillero, and Alberto Tonda. 2019. Evolutionary Discovery of Coresets for Classification. In *Proceedings of the Genetic and Evolutionary Computation Conference 2019, Prague, Czech Republic, July 13–17, 2019 (GECCO '19),* 8 pages.
https://doi.org/10.1145/3319619.3326846

1 INTRODUCTION

The field of machine learning (ML) redefined the concept of *coreset*, originally stemming from computational geometry, as the subset of input samples of minimal size, for which a ML algorithm can obtain a good approximation of its original behavior on the whole set of input samples. In other terms, 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 training set [2]. 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) [9], Frank-Wolfe (FW) [10], Forward Stagewise (FSW) [19], Least-angle regression (LAR) [13][4], Matching Pursuit (MP) [20], and Orthogonal Matching Pursuit (OMP) [21]. 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. It is important to notice, however, that the problem of finding a coreset can be intuitively framed as multi-objective, as the quality of the results is likely dependent on the number of points included in the coreset; and minimizing the number of selected points is probably detrimental to minimizing error. As ML algorithms employ different techniques to accomplish the same goals, it is also reasonable to assume that they would need coresets of different size and shape to operate at the best of their possibilities.

Starting from these two assumptions, and building on previous works [3], an evolutionary approach to coreset discovery for classification tasks is proposed. Starting from a training set, a state-of-the-art multi-objective evolutionary algorithm, NSGA-II [12], is set to find the coresets representing the best trade-offs between amount of points (to be minimized) and classifier error (to be minimized), for a specific classification algorithm. The resulting Pareto front includes different coresets, each one representing an optimal compromise between the objectives. A human expert would then be able to not only select the coreset more suited for their needs, but also obtain extra information on the ML algorithm's behavior, by observing its degradation in performance as the number of coreset points in Pareto-optimal candidate solutions decreases. Alternatively, a candidate coreset on the Pareto front can be automatically selected depending on its performance with respect to an unseen validation set.

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. A meta-analysis of the results on different classifiers shows that, while some of the data points selected are common to different algorithms, the choice of points in the coreset is heavily dependent on the classifier selected for the classification task.

2 BACKGROUND

The basic concepts of machine learning, coreset discovery, and multi-objective evolutionary algorithms, necessary to introduce the scope of this work, are briefly recalled in this section.

2.1 Machine learning and classification

ML algorithms are able to *improve their performance on a given task over time through experience* [23]. Such techniques automatically create models that, once trained on user-provided (training) data, can then provide predictions on unseen (test) data. In essence, ML consists in framing a learning task as an optimization task and finding a near-optimal solution for the optimization problem, exploiting the training data. Popular ML algorithms range from decision trees [7], to logistic regression [11], to artificial neural networks [16].

Classification, a classic ML task, consists in associating a single instance of measurements of several *features*, called *sample*, to one (or more) pre-defined *classes*, representing different groups. ML algorithms can position hyperplanes (often called *decision boundaries*) in the feature space, and later use them to decide the group a given sample belongs to. The placement of decision boundaries is set to maximize technique-specific metrics, whose values depend on the efficacy of the boundary with respect to the (labeled) training data. Decision boundaries inside a classifier can be represented explicitly, for example as a linear or non-linear combination of the features, or implicitly, for example as the outcome of a group of decision trees or other weak classifiers.

2.2 Coreset discovery

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 [2].

Finding coresets for ML problems is an active line of research, with applications ranging from speeding up training of algorithms on large datasets [25] 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: Frank-Wolfe (FW) [10], Greedy Iterative Geodesic Ascent (GIGA) [9], Forward Stagewise (FSW) [19], Least-angle regression (LAR) [13][4], Matching Pursuit (MP) [20] and Orthogonal Matching Pursuit (OMP) [21].

The original FW algorithm applies in the context of maximizing a concave function within a feasible polytope by means of a local linear approximation. In Section 4, we refer to the Bayesian implementation of the FW algorithm designed for core set discovery. This technique, described in [8], 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 [9], 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 [19], LAR [13][4], MP [20] and OMP [21] were all originally devised as greedy algorithms for dimensionality reduction, but have been later applied to coreset discovery, as this last problem represents the transpose of feature selection, choosing samples instead of features. The simplest of the group is FSW, which projects high-dimensional data in a lower dimensional space by selecting, one at a time, the features whose inclusion in the model gives the most statistically significant improvement. MP, on the other hand, 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 its correlations with the target signal.

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. Moreover, most of the coreset discovery solutions proposed in literature provide a unique result, representing the best coreset candidate. However, the problem of finding the coreset, given a specific dataset and an application, can be naturally expressed as multi-objective: on the one hand, the user wishes to identify a set of core points as small as possible; but on the other hand, the performance of the algorithm trained on the coreset should not differ from its starting performance, when trained on the original dataset. For this reason, multi-objective optimization algorithms could be well-suited to this task.

2.3 Multi-objective evolutionary algorithms

Optimization problems with contrasting objectives have no single optimal solution. Each candidate represents a different compromise between the multiple conflicting aims. Yet, it is still possible to search for *optimal trade-offs*, for which an objective cannot be improved without degrading the others. The set of such optimal compromises is called *Pareto front*. Multi-objective evolutionary algorithms (MOEA) currently represent the state of the art for problems with contradictory objectives, and are able to obtain good approximations of the true Pareto front in a reasonable amount of time. One of the most known MOEAs is the Non-Sorting Genetic Algorithm II (NSGA-II) [12], that makes use of a crowding mechanism to spread candidate solutions on the Pareto front as evenly as possible, with considerable efficiency for problems with 2-3 objectives.

3 PROPOSED APPROACH

Starting from the intuition that coreset discovery can be framed as a multi-objective problem, and that the results could be dependant on the target ML algorithm, a novel evolutionary approach to coreset discovery for classification is proposed, building on preliminary results presented in [1]. Given a training set **Tr** and a ML classifier, a candidate solution in the framework represents a coreset, a subset of the original training set. Candidate solutions are internally represented as bit strings, of length equal to the size of the training set, where a 1 in position i means that the corresponding sample s_i is retained in the coreset, while a 0 means that the sample is not considered. The classifier is then trained on the candidate coreset, and then an evaluation is performed on two conflicting objectives: number of samples in the coreset (to be minimized), and resulting error of the classifier on the original training set (to be minimized). NSGA-II is then set to optimize the coreset, finding a suitable Pareto front consisting of the best compromises with respect to the two objectives. In case the user wishes to obtain a single solution, the original training set **Tr** can be split into a training set to be used internally **Tr'** and a validation set **V**. At the end of the evolutionary optimization, each candidate coreset on the Pareto front is evaluated on the validation set **V** (unseen by the evolutionary procedure), to find the compromise with the best generality. A scheme of the proposed approach is presented in Figure 1.

When compared to the preliminary methodology presented in [1], the effectiveness of the approach has been considerably improved. In particular, the addition of the validation set **V** now makes it possible to obtain a fairer comparison with existing algorithms in literature. Furthermore, population size and activation probabilities of the evolutionary operators have been tuned to deliver a better performance on a wider variety of case studies.

From a probabilistic point of view, the algorithm can be summarized as follows. Given sample $s_i = (\vec{x}^i, y^i)$, where $\vec{x}^i$ corresponds to the value of the features and y^i to the known class of s_i, respectively, the objective is to estimate the probability that it belongs to the core set C_j, given the information contained in the training data $\mathcal{D}_t = \{(\vec{x}^i_t, y^i_t)\}$ and the classifier parameters θ:

$$p((\vec{x}^i, y^i) \in C_j | \mathcal{D}_t, \theta) \tag{1}$$

The current estimate of the core set $\hat{C} = \{(\vec{x}^i, y^i) \in C_j\}$ is then used to fit the classifier parameters θ:

$$p(\theta | \hat{C}) \tag{2}$$

Finally, the trained classifier is used to make inferences on the training set:

$$p(y^i_t | \vec{x}^i_t, \theta(\hat{C})) \tag{3}$$

The proposed approach makes it possible to evolve several solutions C_j approximating the core set problem, reducing both the set size and the classification error:

$$\underset{C_j}{\arg\min} \begin{cases} |C_j| \\ \mathcal{L}(\theta(C_j), \mathcal{D}_t) \end{cases} \tag{4}$$

At the end of the evolution, the validation set $\mathcal{D}_v$ is used in order to evaluate the final solutions along the Pareto front. The maximum likelihood estimation for the evolved solutions is given by the core set providing the minimal error on the validation set:

$$\hat{C}^{MLE} = \underset{C_j}{\arg\min} \, \mathcal{L}(\theta(C_j), \mathcal{D}_v) \tag{5}$$

Finally, $\hat{C}^{MLE}$ is used to train the classifier:

$$p(\theta | \hat{C}^{MLE}) \tag{6}$$

to make inferences on an unseen test set $\mathcal{D}_u$:

$$p(y^i_u | \vec{x}^i_u, \theta(\hat{C}^{MLE})) \tag{7}$$

and to compute the accuracy of the model:

$$\mathcal{L}(\theta(\hat{C}^{MLE}), \mathcal{D}_u) \tag{8}$$

4 EXPERIMENTAL RESULTS

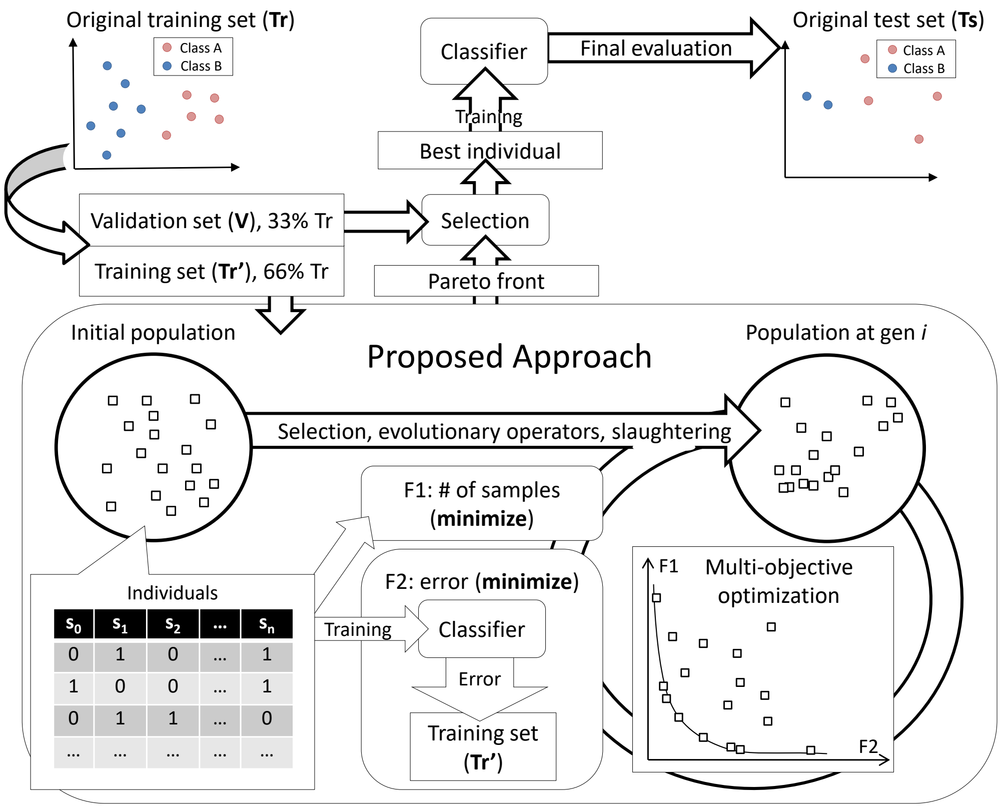
All the experiments presented in this section exploit four ML algorithms, representative of both classifiers with explicit hyperplanes (`Ridge` [24], `SVC` Support Vector Machines [17]) and ensemble, tree-based classifiers (`Bagging` [5], `RandomForest` [6]). All classifiers are implemented in the `scikit-learn`[1] [22] Python module and use default parameters. For the sake of comparison, it is important that the classifiers will follow the same training steps, albeit under different conditions. For this reason, a fixed seed has been set for all algorithms that exploit pseudo-random elements in their training process.

All the necessary code for the experiments has been implemented in Python, relying upon the `inspyred` module[2] [15]. The code is freely available in a BitBucket public repository[3]. NSGA-II uses default parameters of the `inspyred` module, with the exception of $\mu = 200$, $\lambda = 400$, stop condition 200 generations, and evolutionary operators bit-flip (probability $p_{bf} = 0.5$) and 1-point crossover (probability $p_c = 0.5$). Parameter values have been defined after a set of preliminary runs.

The experiments are performed on well-known data sets publicly available in the `scikit-learn` module: i. Blobs, three isotropic gaussian blobs (3 classes, 400 samples, 2 features); ii. Circles, a large circle containing a smaller one (2 classes, 400 samples, 2 features); iii. Moons, two interleaving half circles (2 classes, 400 samples, 2 features); iv. Iris [14] (3 classes, 150 samples, 4 features). For each case study, samples are randomly split between the original training set **Tr** (66%) and test set (33%). While all coreset discovery algorithms evaluated in the following exploit only **Tr** in their training procedure, it must be noted that the proposed approach further randomly splits **Tr** into **Tr'** (66% of **Tr**, 44% of the original dataset) and **V** (33% of **Tr**, 22% of the original dataset), to automatically select a single candidate solution on the Pareto front at the end of the evolutionary process. Features of the datasets are normalized using a normalization learned on the training set **Tr**, then eventually applied to the test set.

The results obtained by the proposed approach are then compared against the state-of-the-art coreset discovery algorithms GIGA [9], FW [10], MP [21], OMP [21], LAR [13][4], and FS [19],

[1]scikit-learn: Machine Learning in Python, http://scikit-learn.org/stable/
[2]inspyred: Bio-inspired Algorithms in Python, https://pythonhosted.org/inspyred/
[3]Evolutionary Discovery of Coresets, https://bitbucket.org/evomlteam/evolutionary-core-sets/src/master/



s_0	s_1	s_2	...	s_n
0	1	0	...	1
1	0	0	...	1
0	1	1	...	0
...	...	...	...	...

Figure 1: Scheme of the proposed approach. The original training set (Tr) is divided into a training set Tr' and a validation set V. The evolutionary approach creates candidates coresets, that are then evaluated on number of samples and error on the training set Tr' (both to be minimized). Once the evolution is over, a single candidate on the Pareto front is selected, evaluating its error on the validation set V. Finally, the best solution is evaluated on an unseen test set Ts, to evaluate its generality.

described in more detail in subsection 2.2. 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). Results of the comparison are presented in Tables 1, 2, 3, and 4, where the proposed approach is labeled *EvoCore*. Text in **bold** highlights the highest accuracy for each classifier on the test set.

With regards to test accuracy, the evolutionary approach not only outperforms by far the other techniques, but is often able to increase the performance obtained by training the same classifier with all the training samples available. This means that the decision boundaries generated using the evolved coresets may generalize even better than those generated using the whole training set. Figure 3 shows the decision boundaries obtained using the whole training set (left column) or the best candidate coreset (right column) to train classifiers, on different datasets. Among all the solutions provided by NSGA-II, the one having the smaller validation error has been selected, and, in case of the same validation error, the one having the smaller coreset. These results suggest that the performance of ML classifiers would not be a function of the *size* of the training set (as Big Data and Deep Learning often claim) but a function of the *mutual position* of the training samples in the feature space.

Figure 2 reports a meta-analysis of all the Pareto-optimal candidate coresets found by the proposed approach, divided by dataset, considering all classifiers. A few samples clearly appear very often among all candidate coresets, while others almost never do, but overall there is a considerable number of samples that are included with low but non-negligible frequency, indicating that different classifiers indeed exploit coresets of different shape.

A final experiment on the MNIST dataset [18], consisting in 70,000 images of handwritten digits, is performed to visually analyze the characteristics of the samples included in the coresets. The experiment uses the same settings as the previous ones, and it is carried out using the classifier `Ridge`. The results are summarized in Table 5. The best candidate coreset on the Pareto front features about 9,600 images, 20% of the original training set. The candidate coreset is manually inspected, and some of the more significant results are illustrated in Figure 4. In particular, points in the coreset for class *8* includes points that are still classified as '8', but they look similar to other classes (for example, *0*, *2*, or *6*), and are thus likely positioned near the corresponding decision boundaries separating the classes in the feature space. This evidence supports the claim

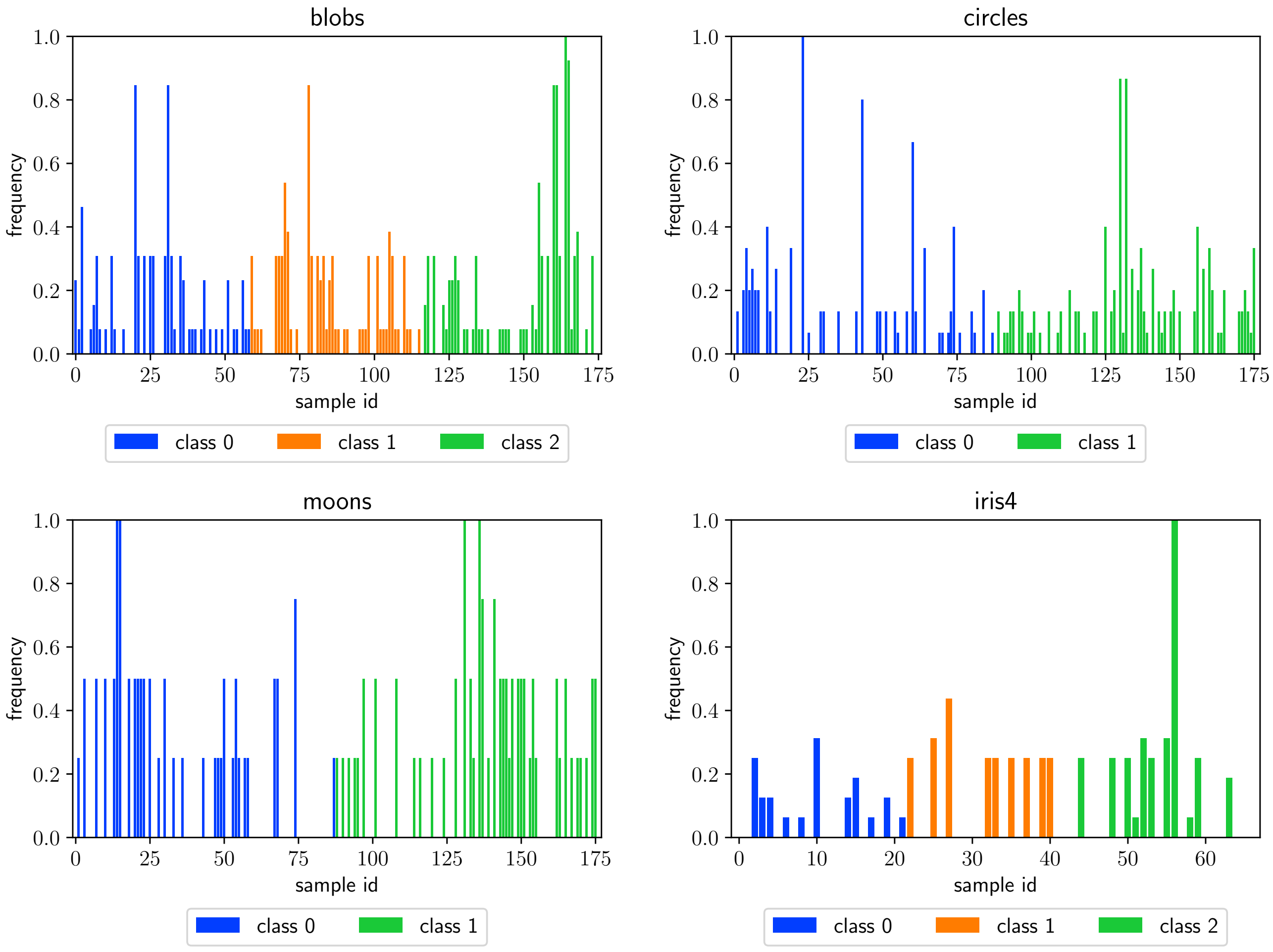


Figure 2: Frequency of appearance of samples in the Pareto front solutions of all the classifiers.

that the proposed approach is able to select meaningful samples to help the classification algorithm to correctly place its decision hyperplanes.

The main drawback of the evolutionary technique is represented by the computational time, which is typically 10 to 1,000 times the order of magnitude of other coreset discovery algorithms in literature, depending on the classifier. However, it can be argued that: i. this problem can be mitigated by parallelizing evaluations (experiments have been run on a consumer-end laptop[4] and it is still not parallelized); ii. the current performance is not an insurmountable obstacle, as usually coresets are computed once, off-line, and then used multiple times; iii. for datasets containing millions or billions of samples, a random subsampling could be used to extract a reduced dataset, to be later exploited during the evolution.

5 CONCLUSIONS

Coreset discovery is a problem of utmost practical importance for machine learning techniques. Most of the coreset discovery algorithms in literature either ask the user to specify the number of samples to be retained in the coreset, or assume the shape of the coreset to be independent from the task and the algorithm it is going to be used for. Starting from the intuition that coreset discovery is an inherently multi-objective problem, and that the structure of a coreset is likely to be dependent on the machine learning technique used for a specific task, an evolutionary approach to coreset discovery is proposed. The presented framework proves to be more efficient than comparable techniques, on several classical benchmarks.

Future works will extend the proposed approach to regression problems, where the aim is to obtain the approximate model of an unknown function, and clustering, where the objective is to obtain high-quality groups of data points, with no pre-existing information on the number or qualities of the groups.

[4]Intel® Core™ i7-8750H 2.20 GHz, 16 GB RAM.

REFERENCES

[1] An author. 2018. A publication. In *A conference*. 1–10.
[2] Olivier Bachem, Mario Lucic, and Andreas Krause. 2017. Practical coreset constructions for machine learning. *arXiv preprint arXiv:1703.06476* (2017).

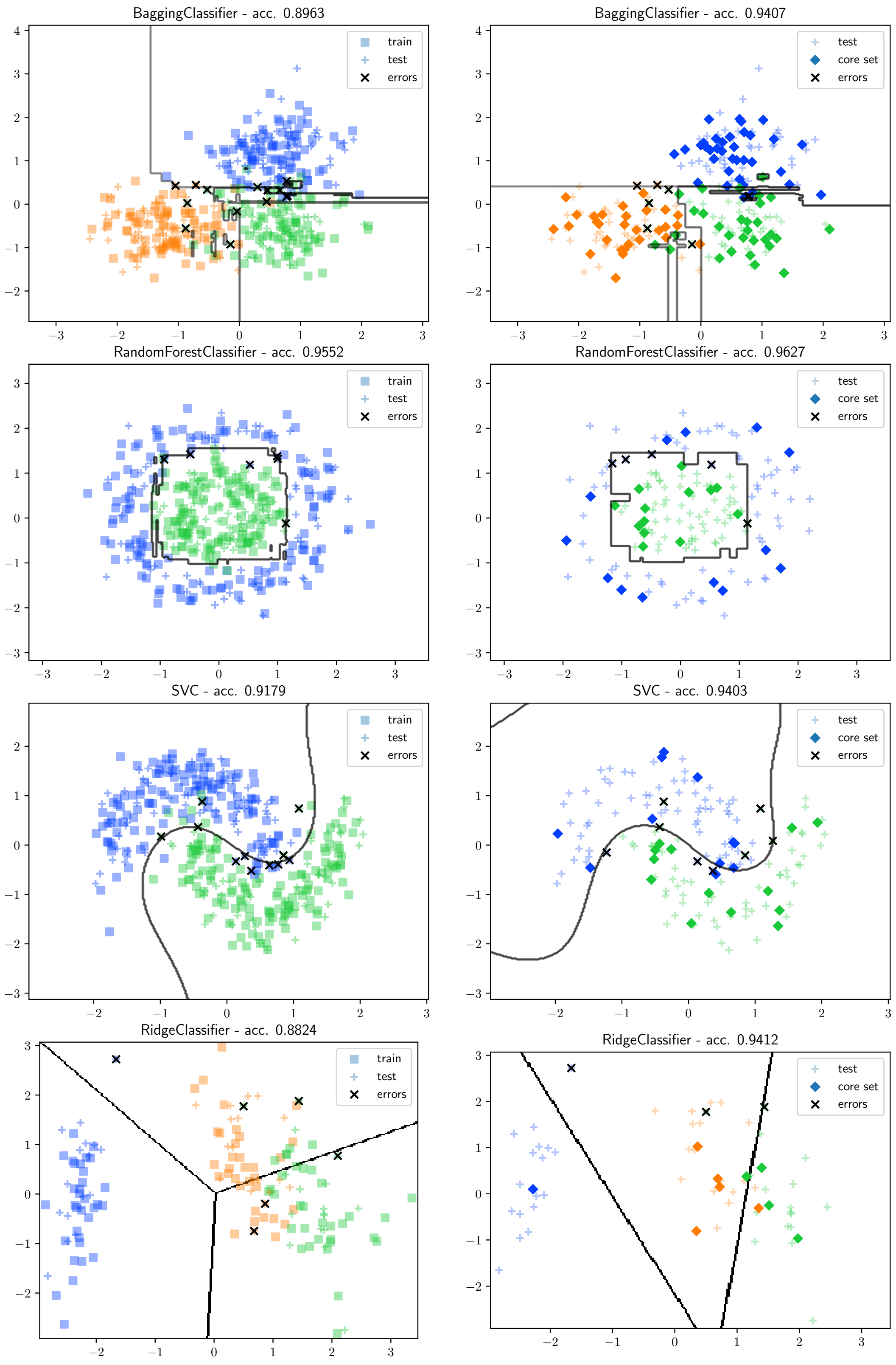


Figure 3: Decision boundaries using all the samples in the training set (Left) and only the coreset (Right) for training the classifier. Train samples are represented by squares, test samples by crosses, coresets by diamonds and test errors by 'x'-shapes. The represented datasets are Blobs (first row), Circles (second row), and Moons (third row), and Iris (fourth row, principal component space), respectively.

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	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	-
EvoCore	47	**0.9333**	518.1 s	94	**0.9407**	554.3 s	3	0.9111	232.8 s	3	**0.9185**	236.2 s
GIGA	3	0.6296	0.01 s	3	0.8889	0.01 s	3	0.8889	0.01 s	3	0.8519	0.01 s
FW	4	0.5185	3.2 s	4	0.6593	3.2 s	4	0.5852	3.2 s	4	0.8815	3.2 s
MP	5	0.4047	4.9 s	5	0.6296	4.9 s	5	0.3333	4.9 s	5	0.5778	4.9 s
FS	5	0.7481	4.6 s	5	0.7481	4.6 s	5	0.3333	4.6 s	5	0.6296	4.6 s
OP	4	0.4074	0.01 s	4	0.6148	0.01 s	4	0.3333	0.01 s	4	0.5852	0.01 s
LAR	3	0.8074	0.01 s	3	0.8074	0.01 s	3	0.5185	0.01 s	3	0.5185	0.01 s

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	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	-
EvoCore	26	**0.9627**	1,040.3 s	13	**0.9552**	878.8 s	6	0.9776	278.5 s	2	0.6343	355.7 s
GIGA	2	0.5970	0.01 s	2	0.5746	0.01 s	2	0.6343	0.01 s	2	0.6364	0.01 s
FW	5	0.5597	3.8 s	5	0.5000	3.8 s	5	0.5000	3.8 s	5	0.5000	3.8 s
MP	3	0.5000	4.1 s	3	0.5224	4.1 s	3	0.5000	4.1 s	3	**0.6567**	4.1 s
FS	4	0.6567	4.4 s	4	0.6194	4.4 s	4	0.6119	4.4 s	4	0.6269	4.4 s
OP	3	0.5000	0.01 s	3	0.4851	0.01 s	3	0.6418	0.01 s	3	0.5448	0.01 s
LAR	2	0.5522	0.01 s	2	0.6194	0.01 s	2	0.5970	0.01 s	2	0.5970	0.01 s

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	size	accuracy	avg time	size	accuracy	avg time	size	accuracy	avg time	size	accuracy	avg time
all samples	266	0.9328	-	2661	0.9254	-	266	0.9179	-	266	0.8134	-
EvoCore	10	**0.9403**	440.2 s	30	**0.9478**	415.9 s	24	**0.9403**	126.6 s	2	**0.8209**	139.8 s
GIGA	2	0.4254	0.01 s	2	0.2463	0.01 s	2	0.4701	0.01 s	2	0.4701	0.01 s
FW	6	0.6493	3.6 s	6	0.6493	3.6 s	6	0.5299	3.6 s	6	0.6866	3.6 s
MP	3	0.5149	4.6 s	3	0.5821	4.6 s	3	0.5896	4.6 s	3	0.6642	4.6 s
FS	2	0.5149	4.3 s	2	0.2313	4.3 s	2	0.6119	4.3 s	2	0.6119	4.3 s
OP	2	0.5149	0.01 s	2	0.2463	0.01 s	2	0.6493	0.01 s	2	0.6493	0.01 s
LAR	3	0.5149	24.2 s	3	0.2388	24.2 s	3	0.5224	24.2 s	3	0.5896	24.2 s

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	size	accuracy	avg time	size	accuracy	avg time	size	accuracy	avg time	size	accuracy	avg time
all samples	266	0.9328	-	2661	0.9254	-	266	0.9179	-	266	0.8134	-
EvoCore	15	**0.9412**	288.3 s	14	**0.9608**	303.7 s	5	0.9412	92.5 s	10	**0.9412**	90.9 s
GIGA	7	0.9216	0.7 s	7	0.6667	0.7 s	7	**0.9804**	0.7 s	7	0.8431	0.7 s
FW	15	0.8824	3.2 s	15	0.8627	3.2 s	15	0.9412	3.2 s	15	0.8235	3.2 s
MP	14	0.9412	4.5 s	14	0.8627	4.5 s	14	0.9216	4.5 s	14	0.7255	4.5 s
FS	7	0.6667	4.2 s	7	0.7059	4.2 s	7	0.6471	4.2 s	7	0.6275	4.2 s
OP	5	0.7059	0.1 s	5	0.5294	0.1 s	5	0.7843	0.1 s	5	0.8235	0.1 s
LAR	4	0.5294	22.2 s	4	0.6863	22.2 s	4	0.6471	22.2 s	4	0.7059	22.2 s

[3] Pietro Barbiero and Alberto Tonda. 2019. Fundamental Flowers: Finding Core Sets for Classification using Evolutionary Computation. In *EvoApplications 2019*.

[4] Christos Boutsidis, Petros Drineas, and Malik Magdon-Ismail. 2013. *Near-optimal Coresets For Least-Squares Regression.* Technical Report. arXiv:1202.3505v2

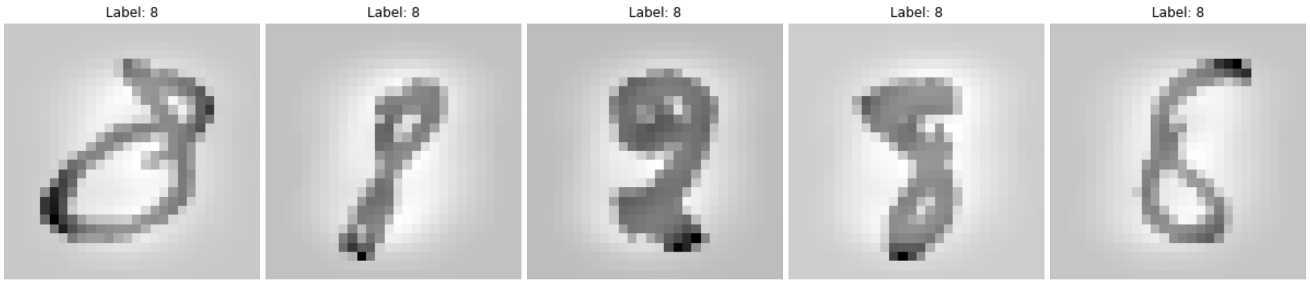


Figure 4: A few samples belonging to the best EvoCore candidate coreset for class 8. Visually, the selected samples appear to be ideal to separate class 8 from (left to right) 0, 1, 2, 7, and 6, respectively.

Table 5: MNIST dataset. Reporting coreset size, accuracy on test set and running time (seconds) of the coreset algorithms.

	Ridge		
algorithm	size	accuracy	avg time
all samples	46664	**0.9447**	-
EvoCore	9638	0.8466	11,915.3 s
GIGA	6134	0.8099	442.1 s
FW	7144	0.8187	2,609.1 s
MP	6585	0.8177	2,368.3 s
FS	583	0.4337	2,354.4 s
OP	98	0.5653	30.2 s
LAR	703	0.4743	532.6 s

https://arxiv.org/pdf/1202.3505.pdf

[5] Leo Breiman. 1999. Pasting small votes for classification in large databases and on-line. *Machine Learning* 36, 1-2 (1999), 85–103.

[6] Leo Breiman. 2001. Random forests. *Machine learning* 45, 1 (2001), 5–32.

[7] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. 1984. *Classification and regression trees.* CRC press.

[8] Trevor Campbell and Tamara Broderick. 2017. Automated Scalable Bayesian Inference via Hilbert Coresets. (2017). arXiv:1710.05053 http://arxiv.org/abs/1710.05053

[9] Trevor Campbell and Tamara Broderick. 2018. Bayesian Coreset Construction via Greedy Iterative Geodesic Ascent. In *International Conference on Machine Learning (ICML).* arXiv:arXiv:1802.01737v2 https://arxiv.org/pdf/1802.01737.pdf

[10] Kenneth L Clarkson. 2010. Coresets, Sparse Greedy Approximation, and the Frank-Wolfe Algorithm. In *ACM Transactions on Algorithms.* http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.145.9299

[11] David R Cox. 1958. The regression analysis of binary sequences. *Journal of the Royal Statistical Society. Series B (Methodological)* (1958), 215–242.

[12] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation* 6, 2 (2002), 182–197.

[13] Bradley Efron, Trevor Hastie, Iain Johnstone, and Robert Tibshirani. 2004. Least Angle Regression. *The Annals of Statistics* 32, 2 (2004), 407–451. https://doi.org/10.1214/009053604000000067

[14] Ronald A Fisher. 1936. The use of multiple measurements in taxonomic problems. *Annals of eugenics* 7, 2 (1936), 179–188.

[15] Aaron Garrett. 2012. inspyred (Version 1.0.1) Inspired Intelligence. https://github.com/aarongarrett/inspyred. (2012).

[16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning* (mit press ed.). arXiv:arXiv:1312.6184v5

[17] Marti A. Hearst, Susan T Dumais, Edgar Osman, John Platt, and Bernhard Scholkopf. 1998. Support vector machines. *IEEE Intelligent Systems and their Applications* 13, 4 (1998), 18–28.

[18] Yann LeCun and Corinna Cortes. 2010. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/. (2010). http://yann.lecun.com/exdb/mnist/

[19] Efroymson M. A. 1960. Multiple Regression Analysis. *Mathematical Methods for Digital Computers* (1960).

[20] StÃĺphane Mallat and Zhifeng Zhang. 1993. Matching pursuits with time-frequency dictionaries. *IEEE Transactions on Signal Processing* 42, 12 (1993), 3397âĂŞ3415.

[21] Y.C. Pati, R. Rezaiifar, and P.S. Krishnaprasad. 1993. Orthogonal matching pursuit: recursive function approximation with applications to wavelet decomposition. *Proceedings of 27th Asilomar Conference on Signals, Systems and Computers* (1993), 40–44. https://doi.org/10.1109/ACSSC.1993.342465 arXiv:1108.3326

[22] 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. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[23] Arthur L Samuel. 1959. Some studies in machine learning using the game of checkers. *IBM Journal of research and development* 3, 3 (1959), 210–229.

[24] Andrey Nikolayevich Tikhonov. 1943. On the stability of inverse problems. In *Dokl. Akad. Nauk SSSR*, Vol. 39. 195–198.

[25] Ivor W Tsang, James T Kwok, and Pak-Ming Cheung. 2005. Core vector machines: Fast SVM training on very large data sets. *Journal of Machine Learning Research* 6, Apr (2005), 363–392.