Fundamental Flowers: Evolutionary Discovery of Coresets for Classification

Pietro Barbiero[1][0000−0003−3155−2564] and Alberto Tonda[2,3][0000−0001−5895−4809]

[1] Politecnico di Torino, Torino, Italy
pietro.barbiero@studenti.polito.it
[2] Université Paris-Saclay, Saclay, France
[3] UMR 782 INRA, Thiverval-Grignon, France
alberto.tonda@inra.fr

Abstract. In an optimization problem, a *coreset* can be defined as a subset of the input points, such that a good approximation to the optimization problem can be obtained by solving it directly on the coreset, instead of using the whole original input. In machine learning, coresets are exploited for applications ranging from speeding up training time, to helping humans understand the fundamental properties of a class, by considering only a few meaningful samples. The problem of discovering coresets, starting from a dataset and an application, can be defined as identifying the minimal amount of samples that do not significangly lower performance with respect to the performance on the whole dataset. Specialized literature offers several approaches to finding coresets, but such algorithms often disregard the application, or explicitly ask the user for the desired number of points. Starting from the consideration that finding coresets is an intuitively multi-objective problem, as minimizing the number of points goes against maintaining the original performance, in this paper we propose a multi-objective evolutionary approach to identifying coresets for classification. The proposed approach is tested on classical machine learning classification benchmarks, using 6 state-of-the-art classifiers, comparing against 7 algorithms for coreset discovery. Results show that not only the proposed approach is able to find coresets representing different compromises between compactness and performance, but that different coresets are identified for different classifiers, reinforcing the assumption that coresets might be closely linked to the specific application.

Keywords: Machine learning · Coresets · Evolutionary Computation · Multi-objective optimization.

1 Introduction

The concept of *coreset*, originally from computational geometry, has been redefined in the field of machine learning (ML) as the minimal number of input samples from which a ML algorithm can obtain a good approximation of the behavior it would have on the whole original dataset. In other words, a coreset

represents a fundamental subset of an available training set, that is sufficient for a given ML algorithm to obtain a good performance, or even the same performance it would have if trained on the whole training set [1]. This definition is intentionally generic, as it encompasses different tasks, ranging from classification, to regression, to clustering, that entail different measures of performance. The practical applications of coresets range from speeding up training time, to obtaining a better understanding of the data itself, making it possible for human experts to analyze only a few data points, instead of dealing with prohibitive amounts of samples.

Discovering coresets for a specific task is an open research line, and specialized ML literature reveals a considerable number of approaches, among which the most popular are Bayesian Logistic Regression (BLR, [17]), Greedy Iterative Geodesic Ascent (GIGA, [7]), Frank-Wolfe (FW, [8]), Forward Stagewise (FSW, [18]), Least-angle regression (LAR, [11][2]), Matching Pursuit (MP, [19]), and Orthogonal Matching Pursuit (OMP, [20]). Interestingly, very often such algorithms require the user to specify the desired number N of points in the coreset; or assume, for simplicity, that a good coreset is independent from the task and/or the ML algorithm selected for that task. However, the problem of finding a coreset can be intuitively framed as multi-objective, as the quality of the results is probably dependent on the number of points included in the coreset; and minimizing the number of selected points goes against maximizing performance. Moreover, it also seems likely that different ML algorithms would actually need coresets of different size and shape to operate at the best of their possibilities.

Starting from these two intuitions, we present an evolutionary approach to coreset discovery for classification tasks. Starting from a given training set, a state-of-the-art multi-objective evolutionary algorithm, NSGA-II [10], is set to find the coresets representing the best trade-offs between amount of points (to be minimized) and classifier accuracy (to be maximized), for a specific classification algorithm. The resulting Pareto front will then represent several coresets, each one a different optimal compromise beteween the objectives, and a human expert will 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 from observing its decrease in performance as the number of coreset points decreases.

Experimental results on two iconic 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, belonging to the same benchmark. Moreover, a meta-analysis of the results on different classifiers shows that indeed, while some of the 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, with similarities between classifiers belonging to the same families.

2 Background

In this section, we briefly recall the basics of machine learning, coreset discovery, and multi-objective evolutionary algorithms, that are necessary to introduce the scope of our work.

2.1 Machine learning and classification

Machine learning techniques are defined as algorithms able to *improve their performance on a given task over time through experience* [22]. In other words, such algorithms create models that are trained on user-provided (training) data, and are then able to provide predictions on unseen (test) data. The essence of ML is to frame a learning task as an optimization task, to then find a near-optimal solution for the optimization problem, relying upon the training data. Popular machine learning techniques range from decision trees [5], to logistic regression [9], to artificial neural networks [15].

A classic task for ML is *classification*, where a single instance of measurements of several *features*, called *sample*, is to be associated to one (or more) pre-defined *classes*, representing different groups. Usually ML techniques position hyperplanes (also called *decision boundaries*) in the feature space, and later use them to decide how to classify a given sample. Such decision boundaries are placed to maximize technique-specific metrics, whose value depends on the efficacy of the boundary with respect the (labeled) training data.

2.2 Coresets

Coresets were originally studied in the context of computational geometry, and defined as a small set of points that approximates the shape of a larger point set. In ML the definition of coreset is extended to intend a subset of the 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 [24] 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: Bayesian Logistic Regression (BLR, [17]), Greedy Iterative Geodesic Ascent (GIGA, [7]), Frank-Wolfe (FW, [8]), Forward Stagewise (FSW, [18]), Least-angle regression (LAR, [11][2]), Matching Pursuit (MP, [19]) and Orthogonal Matching Pursuit (OMP, [20]).

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 region 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 [6], 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 [7], 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 [18], LAR [11][2], MP [19] and OMP [20] 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.

Very often these algorithms start from the assumption that the coreset for a given dataset will be independent from the ML pipeline used. This premise might not always be correct, as the optimization problem underlying, for example, a classification task, might vary considerably depending on the algorithm used. It is also important to notice that 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 evolutionary algorithms could be well-suited to this task.

2.3 Multi-objective evolutionary algorithms

When dealing with problems that feature contrasting objectives, there is no single optimal solution to be found. Each candidate solution, in fact, represents a different compromise between conflicting aims. Nevertheless, it is still possible to find *optimal trade-offs*, for which it is not possible to improve an objective without degrading the others. The set of such optimal compromises is called Pareto front. Multi-objective evolutionary algorithms (MOEA) have been particularly successful in tackling problems with contradictory objectives and obtaining good approximations of the Pareto front. One of the most known MOEAs is the Non-Sorting Genetic Algorithm II (NSGA-II) [10], that exploits a crowding mecha-

nism to evenly spread candidate solutions on the Pareto front, a procedure that is considerably efficient for problems with 2-3 objectives.

3 Proposed approach

We propose to exploit evolutionary computation to explore the space of all subsets of samples in a given training dataset in a classification task, searching for those subset that do not reduce classification accuracy. Such samples would then represent a coreset, a collection of points that is necessary and sufficient to correctly define the decision boundaries for a target classifier, given a target dataset.

A candidate solution in our problem is a set of samples to be kept from the training set. The genome of an individual is thus defined as a binary array of size equal to the training set, with a '1' in a given position meaning that the sample corresponding to that index will become part of the coreset, and a '0' meaning that the sample will be removed.

Intuitively, it is easier to maintain a classifier's accuracy while keeping a large number of samples from its training set, rather than just a small quantity. For this reason, it is more appropriate to frame the optimization problem as multi-objective, with the conflicting aims of *i. minimizing the number of samples in the coreset*, and *ii. maximizing classifier's accuracy on the whole dataset*. The first objective is measured by simply counting the number of '1's, i.e. the number of samples in the coreset, in each candidate solution. For the second objective, the target classifier is trained on the reduced training set, and its accuracy is then tested on the whole set of training points, including all samples removed from the training set by the candidate solution. A scheme of the individual evaluation is presented in Figure 1.

4 Experimental evaluation

In order to empirically validate the proposed approach, we evaluate the methodology with 6 algorithms, chosen to be representative of both hyperplane-based and ensemble, tree-based classifiers: `BaggingClassifier` [3], `GradientBoostingClassifier` [13], `LogisticRegression` [9], `RandomForestClassifier` [4], `RidgeClassifier` [23], `SVC` (Support Vector Machines) [16]. All classifiers are implemented in the `scikit-learn`[4] [21] Python module and use default parameters. A fixed seed has been set for all those that exploit pseudo-random elements in their training process, such as `RandomForestClassifier`, as for our objective it is important that the classifier will follow the same training steps, albeit with a reduced training set. A similar result could have been obtained by repeating multiple times the training process of classifiers containing random elements. In this first batch of experiments, that constitute our proof of concept, we selected the former option to reduce computational effort. The implementation of

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

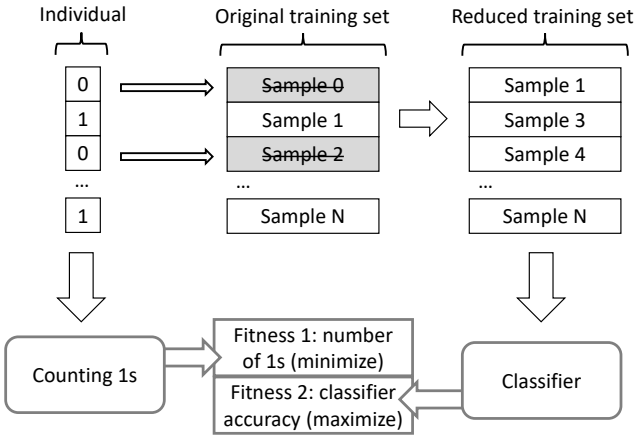

Fig. 1. Scheme of the proposed fitness evaluation. A given dataset is randomly split between training and test set. Candidate solutions, encoded as binary strings, are used to reduce the training set, by removing samples whose index corresponds to a '0' in the binary string. The target classifier is then trained on the reduced training set, while its accuracy is measured on all the original training samples.

NSGA-II, used during the evolutionary process, is taken from the `inspyred`[5] [14] Python module. NSGA-II uses a binary tournament selection, $\mu = 100$, $\lambda = 100$, a one-point crossover, a bit-flip mutation, and a stop condition set at 100 generations. Each individual can remove between 1 and 90 samples from the training dataset, so valid candidate genomes can contain between 1 and 90 '0's.

The results obtained by the proposed approach are then compared against the 7 coreset discovery algorithms BLR, [17], GIGA, [7], FW [8], MP [20], OMP [20], LAR [11][2], and FSW [18], described in more detail in subsection 2.2. Some of the algorithms require the user to specify the number N of desired points in the coreset: in order to provide a fair comparison, N is set to the size of the highest-accuracy coreset found by the proposed approach in the corresponding experiment.

All experiments are performed on the well-known Iris dataset [12], comprising 150 samples from 3 different classes. The samples are randomly split between a 99-sample training set and a 51-sample test set. As shown in Table 1, all considered classifiers perform reasonably well on the dataset, as Iris is a benchmark that presents no particular difficulty for most algorithms. In a first batch of experiments (Iris-2), we consider only two features of the dataset, in order to offer the reader a more intuitive visual assessment of the results.In a second set of

[5] inspyred: Bio-inspired Algorithms in Python, `https://pythonhosted.org/inspyred/`

tests (Iris-4), we consider all four features, and rely upon *Principal Component Analysis* (PCA) to support the visualization and discussion. The datasets are shown in Figure 2. The code used in this work is freely available in the BitBucket repository `https://bitbucket.org/evomlteam/evolutionary-core-sets`.

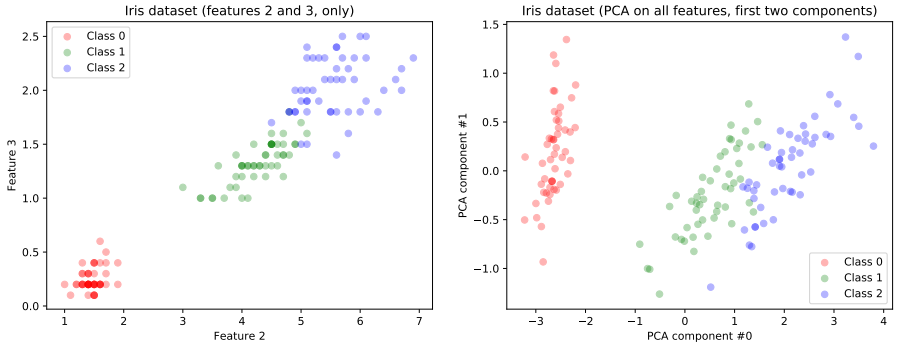

Fig. 2. Samples from the Iris dataset **(left)** visualized considering only features 2 and 3, **(right)** visualized considering the first two components of a PCA performed on all features.

Table 1. Initial performance (classification accuracy) of the considered classifiers on the Iris dataset (using two features and all, respectively).

	Iris (2 features)			**Iris (all features)**		
Classifier	**Overall**	**Training**	**Test**	**Overall**	**Training**	**Test**
GradientBoostingClassifier	0.9800	0.9899	0.9608	0.9867	1.0	0.9608
BaggingClassifier	0.9733	0.9899	0.9412	0.9867	1.0	0.9608
LogisticRegression	0.8533	0.8081	0.9412	0.9733	0.9596	1.0
RandomForestClassifier	0.9733	0.9899	0.9412	0.9867	1.0	0.9608
RidgeClassifier	0.8000	0.7677	0.8627	0.8667	0.8586	0.8824
SVC	0.9733	0.9697	0.9804	0.9733	0.9697	0.9804

4.1 Iris 2-feature validation

The first column of Figures 3 and 4 shows the Pareto front obtained by the proposed approach for all the classifiers on Iris-2. It is noticeable how the trade-offs found are different in both size and accuracy, depending on the considered algorithm. The second and third columns of the figures portray the frequency of appearance of samples in the training set inside the candidate solutions on the

Table 2. Test performance of the considered classifiers and coreset algorithms on the Iris-2 dataset.

	Training	EvoCore	BLR	GIGA	FW	MP	OMP	FSW	LAR
BaggingClassifier	0.9798	1.0000	0.8824	0.8824	0.8039	0.9412	0.9412	0.9412	0.9412
GradientBoostingClassifier	0.9596	0.9804	0.8627	0.8431	0.8824	0.9608	0.9608	0.9608	0.9608
LogisticRegression	0.9697	0.9804	0.6471	0.7059	0.6667	0.9804	0.9804	0.9804	0.9804
RandomForestClassifier	0.9798	0.9804	0.9020	0.8824	0.8039	0.9412	0.9412	0.9412	0.9412
RidgeClassifier	0.9798	0.9608	0.6667	0.6667	0.6667	0.9020	0.9020	0.9020	0.9020
SVC	0.9697	0.9804	0.6667	0.8824	0.8627	0.9412	0.9412	0.9412	0.9412
Average	**0.9731**	**0.9804**	**0.7712**	**0.8105**	**0.7810**	**0.9444**	**0.9444**	**0.9444**	**0.9444**

Table 3. Test performance of the considered classifiers and coreset algorithms on the Iris-4 dataset.

	Training	EvoCore	BLR	GIGA	FW	MP	OMP	FSW	LAR
BaggingClassifier	0.9596	0.9804	0.7059	0.9412	0.8824	0.9608	0.4118	0.8431	0.4118
GradientBoostingClassifier	0.9697	0.9804	0.7647	0.7255	0.8824	0.8824	0.6471	0.8824	0.6471
LogisticRegression	0.9798	1.0000	0.7843	0.8824	0.8039	0.8431	0.8039	0.7059	0.8039
RandomForestClassifier	0.9394	0.9608	0.7255	0.8627	0.9020	0.9804	0.7647	0.7843	0.7647
RidgeClassifier	0.9798	1.0000	0.7255	0.9020	0.8824	0.8824	0.7451	0.7451	0.7451
SVC	0.9697	0.9608	0.7843	0.9608	0.9608	0.9608	0.6275	0.6471	0.6275
Average	**0.9663**	**0.9804**	**0.7484**	**0.8791**	**0.8856**	**0.9183**	**0.6667**	**0.7680**	**0.6667**

Pareto fronts of the corresponding experiment. It is easy to observe that several points appear among all candidate coresets found in the same experiment.

Table 2 presents the accuracy of the most accurate coreset found by the proposed approach, against coresets discovered by state-of-the-art algorithms in ML literature. Not only the proposed approach outperforms the accuracy of all algorithms on the training set, but it also enables the classifiers to create decision boundaries that generalize better, obtaining improved results on the unseen test set as well.

4.2 Iris 4-feature validation

The first column of Figures 5 and 6 shows the Pareto front obtained by the proposed approach for all the classifiers on Iris-4. Again, the trade-offs found appear different in both size and accuracy, depending on the considered algorithm. As we observed for Iris-2, the second and third columns of the figures portraying the frequency of appearance of samples in the training set inside the candidate solutions on the Pareto fronts show several points appearing among all candidate coresets found in the same experiment.

Table 3 presents the accuracy of the most accurate coreset found by the proposed approach, against coresets discovered by state-of-the-art algorithms in ML literature. Again, the proposed approach outperforms the accuracy of all algorithms on both the training set and the unseen test set.

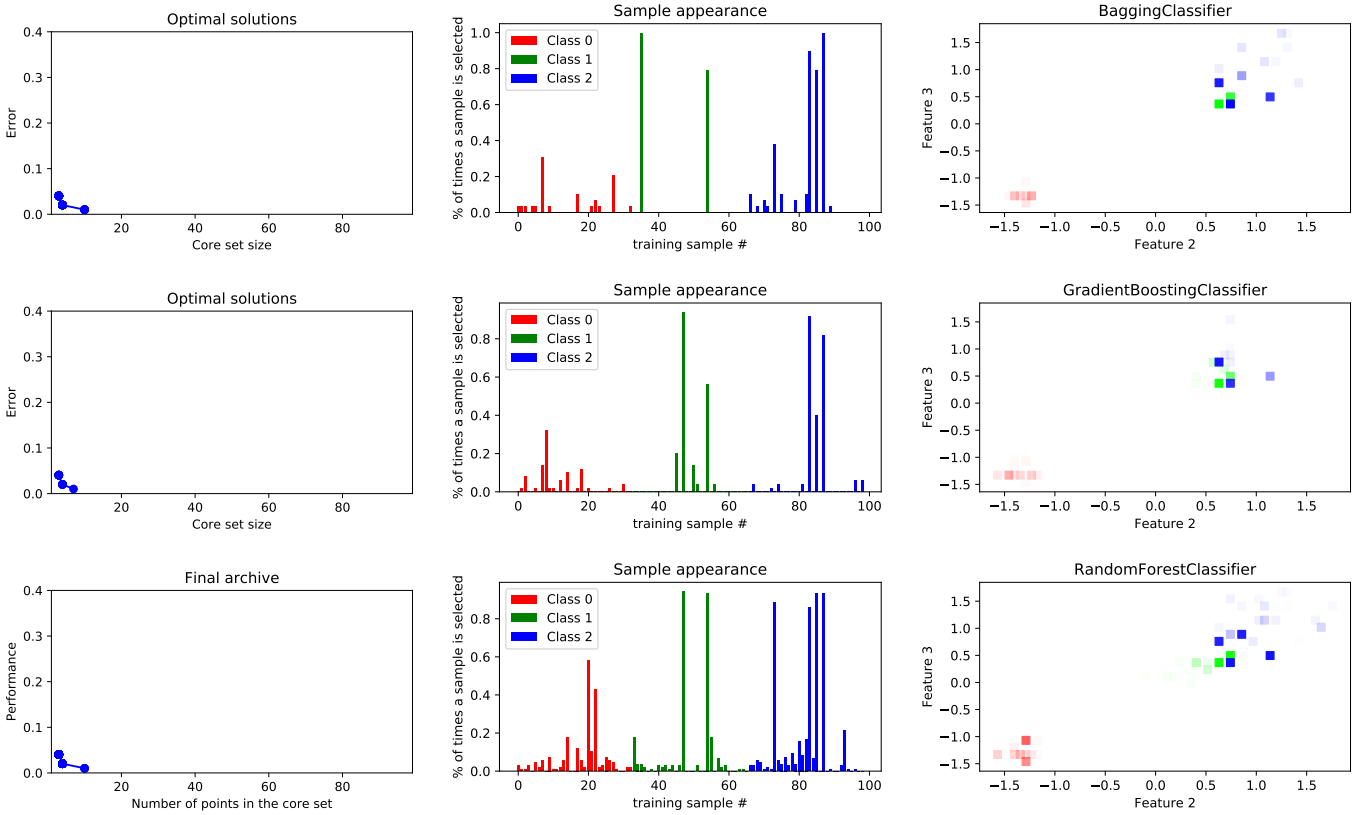

Fig. 3. For each experiment on dataset Iris-2, Pareto fronts showing optimal solutions (**left**), bar plots (**center**) and scatter plots (**right**) displaying the frequency of appearance of samples inside coresets. Bar height in bar plots and color saturation in scatter plots are directly proportional to the frequency of appearance of a specific sample. The first row refers to `BaggingClassifier`, the second one to `GradientBoostingClassifier`, and the third one to `RandomForestClassifier`.

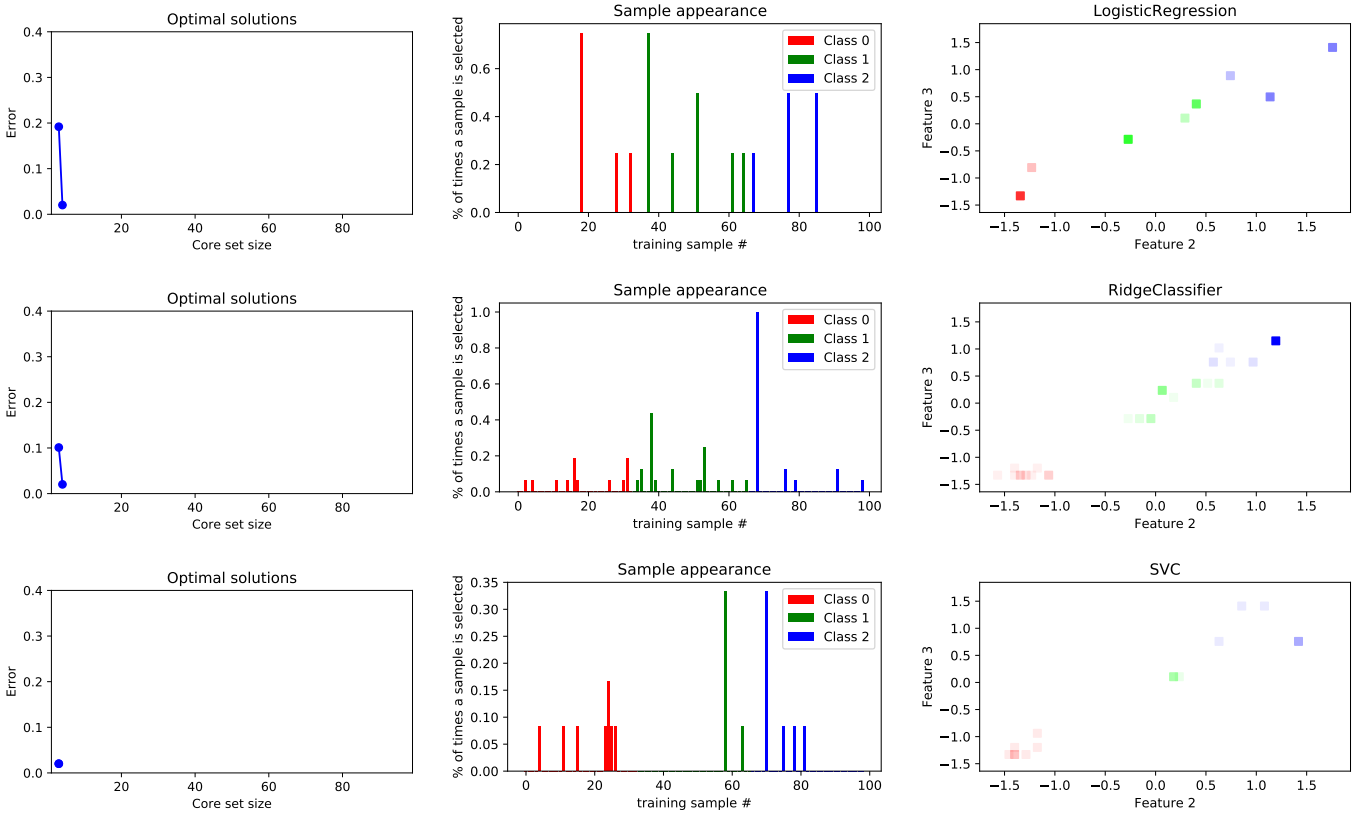

Fig. 4. For each experiment on dataset Iris-2, Pareto fronts showing optimal solutions (**left**), bar plots (**center**) and scatter plots (**right**) displaying the frequency of appearance of samples inside coresets. Bar height in bar plots and color saturation in scatter plots are directly proportional to the frequency of appearance of a specific sample. The first row refers to `LogisticRegression`, the second one to `RidgeClassifier`, and the third one to `SVC`.

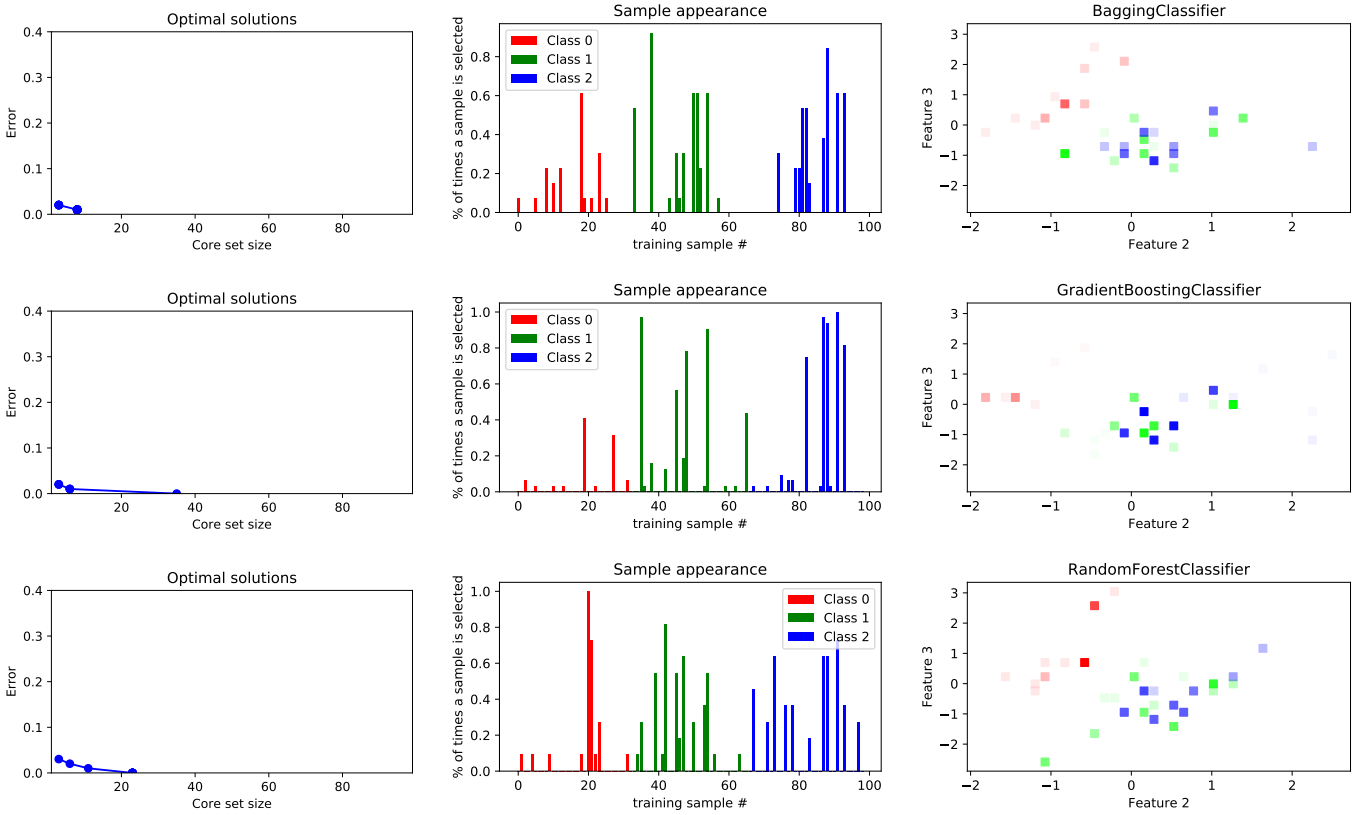

Fig. 5. For each experiment on dataset Iris-4, Pareto fronts showing optimal solutions (**left**), bar plots (**center**) and scatter plots (**right**) displaying the frequency of appearance of samples inside coresets. Bar height in bar plots and color saturation in scatter plots are directly proportional to the frequency of appearance of a specific sample. The first row refers to `BaggingClassifier`, the second one to `GradientBoostingClassifier`, and the third one to `RandomForestClassifier`.

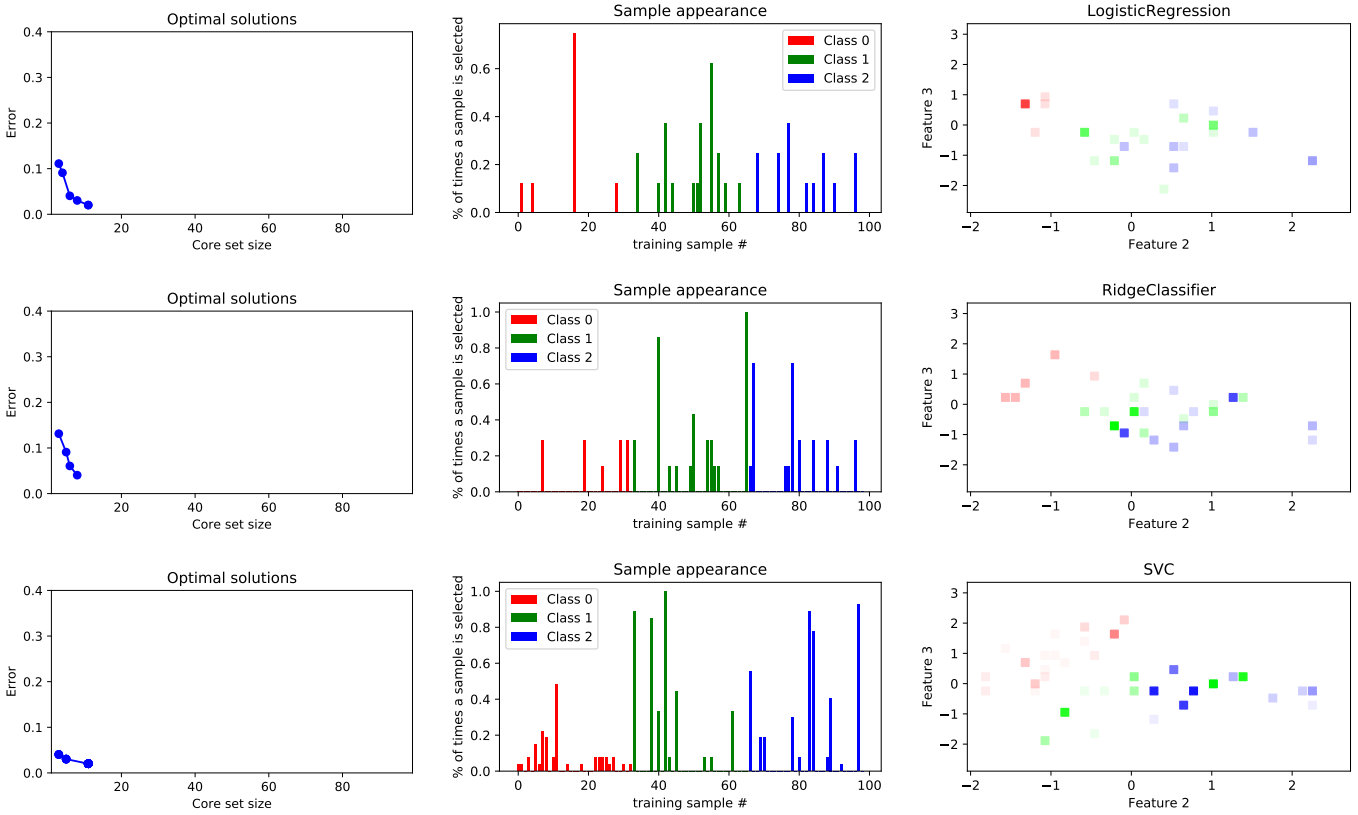

Fig. 6. For each experiment on dataset Iris-4, Pareto fronts showing optimal solutions (**left**), bar plots (**center**) and scatter plots (**right**) displaying the frequency of appearance of samples inside coresets. Bar height in bar plots and color saturation in scatter plots are directly proportional to the frequency of appearance of a specific sample. The first row refers to `LogisticRegression`, the second one to `RidgeClassifier`, and the third one to `SVC`.

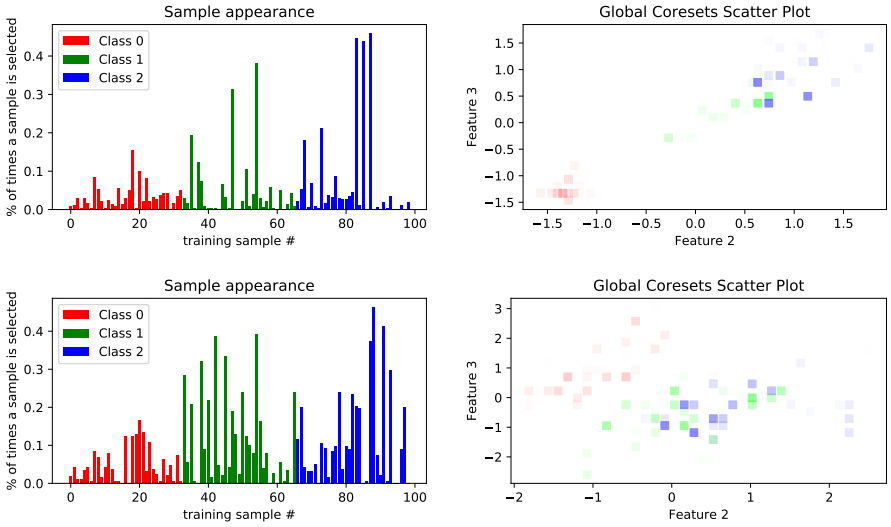

Fig. 7. Global results, taking into account all candidate solutions found during all experiments on Iris-2 **(top row)** and Iris-4 **(bottom row)**. Bar height in bar plots **(left)** and color saturation in scatter plots **(right)** are directly proportional to the frequency of appearance of a specific sample.

5 Discussion

The proposed framework proved to be extremely effective in discovering high-quality coresets for classification. Framing the problem as a function of both application and algorithm seems to be a better method than a more general approach, as highlighted by the differences in the coresets found using different classifiers. Furthermore, obtaining a Pareto front of different compromises, instead of a single solution, might not only provide the user with different viable alternatives, but also provide insight on the behavior of the classifier considered for the task.

Nevertheless, this methodology presents a few drawbacks. The main issue of the approach is computational time. All other coreset discovery algorithms in literature are able to return solutions in a few seconds on regular laptops, while the proposed method typically takes from minutes to tens of minutes. In fact, as the MOEA trains a classifier for each individual evaluation, everything depends on the classifier itself. Among the selected classifiers, a huge variance in training time is noticable between the fastest (`RandomForestClassifier`) and the slowest (`GradientBoostingClassifier`), with corresponding repercussions on the speed of the MOEA. While this problem can be mitigated by parallelizing evaluations, we argue that even the current performance is not a great obstacle,

as usually coresets are computed once, and then used multiple times. Given that not only the quality of the coresets discovered by the MOEA is higher, but that also several trade-offs are delivered in place of a single solution, we contend that the proposed approach is a preferable alternative to other solutions in literature.

From an overall analysis of the results on the two benchmarks Iris-2 and Iris-4, it is noticeable that, while points frequently appearing in candidate solutions on the Pareto front are generally different between classifiers, algorithms based on decision trees (`RandomForestClassifier`, `GradientBoostingClassifier`, `BaggingClassifier`) and algorithms based on linear hyperplanes (`LogisticRegression`, `RidgeClassifier`, `SVC`) tend to use similar points, see the second and third column of Figures 3, 4, 5, and 6. The two families seem to prefer points on the boundaries between classes (decision trees) or close to the class centroid (linear hyperplanes), respectively. Furthermore, taking into account all coresets found during all the experiments, Figure 7 shows how, despite differences between algorithms, a few samples are consistently selected among most of the candidate solutions found. Such samples might have a relevance going beyond the scope of the single technique, being instead excellent representatives of the class they belong to. Not surprisingly, the most common samples for each class are in part close to the class centroid, and in part close to the class boundaries, fully defining the shape of the training samples of the class.

6 Conclusions and future works

In this work, a novel methodology for coreset discovery in classification tasks is presented. The proposed approach relies on multi-objective evolutionary optimization to find the best compromises between size of the coreset (to be minimized) and classification accuracy (to be maximized). The approach is shown to outperform state-of-the-art coreset discovery algorithms in literature on classical classification benchmarks, and despite its considerably longer computational time, we argue that it is a preferable alternative as the quality of the coresets found is higher, and the algorithm is also more informative for the user.

Future works will analyze the behavior of evolutionary core set discovery for high-dimensional datasets, where visual expert validation is impractical, and explore the possibilities of the proposed approach for regression tasks.

References

1. Bachem, O., Lucic, M., Krause, A.: Practical coreset constructions for machine learning. arXiv preprint arXiv:1703.06476 (2017)
2. Boutsidis, C., Drineas, P., Magdon-Ismail, M.: Near-optimal Coresets For Least-Squares Regression. Tech. rep. (2013), `https://arxiv.org/pdf/1202.3505.pdf`
3. Breiman, L.: Pasting small votes for classification in large databases and on-line. Machine Learning **36**(1-2), 85–103 (1999)
4. Breiman, L.: Random forests. Machine learning **45**(1), 5–32 (2001)
5. Breiman, L., Friedman, J., Stone, C.J., Olshen, R.A.: Classification and regression trees. CRC press (1984)

6. Campbell, T., Broderick, T.: Automated Scalable Bayesian Inference via Hilbert Coresets (2017), http://arxiv.org/abs/1710.05053
7. Campbell, T., Broderick, T.: Bayesian Coreset Construction via Greedy Iterative Geodesic Ascent. In: International Conference on Machine Learning (ICML) (2018), https://arxiv.org/pdf/1802.01737.pdf
8. Clarkson, K.L.: Coresets, Sparse Greedy Approximation, and the Frank-Wolfe Algorithm. In: ACM Transactions on Algorithms (2010), http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.145.9299{\&}rep=rep1{\&}type=pdf
9. Cox, D.R.: The regression analysis of binary sequences. Journal of the Royal Statistical Society. Series B (Methodological) pp. 215–242 (1958)
10. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE transactions on evolutionary computation **6**(2), 182–197 (2002)
11. Efron, B., Hastie, T., Johnstone, I., Tibshirani, R.: Least Angle Regression. The Annals of Statistics **32**(2), 407–451 (2004). https://doi.org/10.1214/009053604000000067, https://arxiv.org/pdf/math/0406456.pdf
12. Fisher, R.A.: The use of multiple measurements in taxonomic problems. Annals of eugenics **7**(2), 179–188 (1936)
13. Friedman, J.H.: Greedy function approximation: a gradient boosting machine. Annals of statistics pp. 1189–1232 (2001)
14. Garrett, A.: inspyred (version 1.0.1) inspired intelligence. https://github.com/aarongarrett/inspyred (2012)
15. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. Mit press edn. (2016)
16. Hearst, M.A., Dumais, S.T., Osman, E., Platt, J., Scholkopf, B.: Support vector machines. IEEE Intelligent Systems and their Applications **13**(4), 18–28 (1998)
17. Huggins, J.H., Campbell, T., Broderick, T.: Coresets for Scalable Bayesian Logistic Regression. In: 30th Annual Conference on Neural Information Processing Systems (NIPS) (2016), https://arxiv.org/pdf/1605.06423.pdf
18. M. A., E.: Multiple Regression Analysis. Mathematical Methods for Digital Computers (1960)
19. Mallat, S., Zhang, Z.: Matching pursuits with time-frequency dictionaries. IEEE Transactions on Signal Processing **42**(12), 33973415 (1993)
20. Pati, Y., Rezaiifar, R., Krishnaprasad, P.: 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). https://doi.org/10.1109/ACSSC.1993.342465, http://ieeexplore.ieee.org/document/342465/
21. 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. Journal of Machine Learning Research **12**, 2825–2830 (2011)
22. Samuel, A.L.: Some studies in machine learning using the game of checkers. IBM Journal of research and development **3**(3), 210–229 (1959)
23. Tikhonov, A.N.: On the stability of inverse problems. In: Dokl. Akad. Nauk SSSR. vol. 39, pp. 195–198 (1943)
24. Tsang, I.W., Kwok, J.T., Cheung, P.M.: Core vector machines: Fast svm training on very large data sets. Journal of Machine Learning Research **6**(Apr), 363–392 (2005)