Evolving Simple Models for Regression, Classification, and Clustering

Evan Lavender
edl43@drexel.edu

***Abstract*—This paper presents the use of evolutionary black-box optimization techniques to perform supervised linear and binary logistic regression and unsupervised data clustering. The algorithms used are a naive *Evolution Strategy* (ES), a *genetic algorithm* variant of ES, a *Natural Evolution Strategy* (NES), and *Differential Evolution* (DE). The `LinearRegression`, `LogisticRegression`, and `KMeans` models implemented by [1] are used for comparison. Toy data is generated by [1] and used to demonstrate regression, classification, and clustering. Real data from [2][1] is used for regression, and real data from [3][2] is used for binary classification. The experiments show metrics comparable to, and sometimes slightly better than the traditional techniques for linear and logistic regression and clustering.**

***Index Terms*—evolutionary algorithm, evolution strategy, natural evolution strategy, differential evolution, black-box optimization**

I. Introduction

Evolution Strategy and Differential Evolution algorithms belong to the *evolutionary algorithm* family. Both belong to the class of black box optimization algorithms that are heuristic search procedures inspired by natural evolution: At every iteration, a population of parameter vectors is perturbed and their objective function is evaluated [4]. The highest scoring (*fitness*) parameter vectors are then recombined to form the population for the next generation [4]. This procedure is repeated until the objective function is optimized [4]. The two types of algorithms differ in how they perform the perturbation (*mutation*) and recombination.

Black-box optimization algorithms are meant to operate on optimization problems that too difficult or complex to model directly. When the objective function is nonlinear and non-differentiable, direct search approaches are the methods of choice [5]. The objective function can be treated as a *black-box*; requiring no additional information besides fitness evaluations at certains points in parameter space [6]. This can be seen as estimating the gradient of the objective function by using the finite differences between solutions in the population.

The ES and DE algorithms differ in when they create the population, how they sample the solutions, and how they update parameters. ES algorithms typically create a population at the beginning of each iteration by sampling a normal distribution. The μ parameter of the distribution represents the best or *mean* solution found so far. The σ parameter may either be fixed, or updated during iteration. This random sampling serves to generate *mutations* of the best (μ) solution found. The DE algorithm has a hyperparameter of *bounds*, which are used to sample an initial population from a uniform distribution. This population is then evolved in place, with mutated solutions created using the difference between other solutions.

[1]https://www.kaggle.com/burakhmmtgl/energy-molecule

[2]https://www.kaggle.com/pavanraj159/predicting-a-pulsar-star

II. Related Work

The problem of black-box optimization has spawned a wide variety of approaches [6]. In addition to evolution strategies and differential evolution algorithms, these include the broad class of genetic algorithms, simulated annealing [7], particle swarm optimization [8], and others. Evolution strategies were designed to cope with high-dimensional continuous-valued domains and have remained an active field of research for more than four decades [9].

The naive ES variant is described in [10]. This implementation is very greedy in nature, and throws away all but the single best solution. The genetic algorithm variant is also described in [10]. The idea is to keep a percentage of the best solutions, and sample the new population from them. [4] describes a natural evolution strategy. NES algorithms maximize the average objective value over the *entire* population.

[5] introduces the differential evolution algorithm. DE generates new solutions by adding a weighted difference vector between two solutions to a third solution [5]. For each member of the population, a potential solution is generated and will replace the current member if it has a higher objective fitness value.

III. Methodology

A. Simple/Naive Evolution Strategy

The simplest evolution strategy samples from a normal distribution with a mean μ and a fixed standard deviation σ [10]. After the fitness results are evaluated, μ is set to the best solution in the population [10]. The next generation of samples is then centered around this new mean. The initial value for μ is a zero-vector.

Algorithm 1: Naive Evolution Strategy

Input: standard deviation σ
for $g = 0, 1, 2, ...$ **do**
  Sample $x_i, ..., x_n \sim \mathcal{N}(\mu_g, \sigma) \sim \mu_g + \sigma\mathcal{N}(0, I)$
  Compute returns $F_i = F(x_i)$ for $i = 1, ..., n$
  Set $\mu_{g+1} = \max F_i$
end

B. Simple Genetic Algorithm

Instead of only selecting the best solution, the genetic algorithm implementation will keep a percentage of the best solutions. Sampling a new solution involves adding Gaussian noise to a solution that is created by combining two solutions from the previous generation. For each solution in the population, two solutions of the previous generation are randomly selected to be combined. The recombination process takes a parameter from either solution with a 50% chance. Finally, this new solution has the sampled noise added to it.

Algorithm 2: Genetic Evolution Strategy

Input: standard deviation σ, number of parents λ
Initialize parents $p_g = p_i, ..., p_\lambda$
for $g = 0, 1, 2, ...$ **do**
  Sample $x_i, ..., x_n \sim \mathcal{N}(0, \sigma) \sim \sigma\mathcal{N}(0, I)$
  Recombine λ parents, add to $x_i, ..., x_n$
  Compute returns $F_i = F(x_i)$ for $i = 1, ..., n$
  Set $p_{g+1} = \max_{1...\lambda} F$
  Set $\mu_{g+1} = \frac{1}{\lambda} \sum p_{g+1}$
end

C. Natural Evolution Strategy

The natural evolution strategy algorithm described in [4] is very similar to the naive variation; they differ only in how the parameters are updated. This NES algorithm takes every member of the population into account when updating the μ parameter for the distribution.

Algorithm 3: Natural Evolution Strategy

Input: standard deviation σ, learning rate α
for $g = 0, 1, 2, ...$ **do**
  Sample $\epsilon_i, ..., \epsilon_n \sim \mathcal{N}(0, I)$
  Compute returns $F_i = F(\mu_g + \sigma\epsilon_i)$ for $i = 1, ..., n$
  Set $\mu_{g+1} = \mu_g + \alpha \frac{1}{n\sigma} \sum_{i=1}^{n} F_i \epsilon_i$
end

[4]

D. Differential Evolution

The Differential Evolution algorithm is not a variation of an evolution strategy, but is still within the family of evolutionary algorithms. An initial population is sampled from a uniform distribution with bounds set as a hyperparameter for the algorithm. The crucial idea behind DE is a scheme for generating trial parameter vectors [5]. DE generates new parameter vectors by adding a weighted difference (*differential*) vector between two population members for a third member [5]. This resulting vector is evaulated, and if its value is greater than the value of a predetermined population member, then it will replace that member for the following generation [5].

Trial vectors are generated according to

$$v = x + \lambda(x_{best} - x) + F(x_2 - x_3)$$

where x_2 and x_3 are randomly selected and $\neq x$, and $F > 0$ [5]. F is a parameter for controlling the amplification of the differential variation [5]. λ provides a means to enhance greediness by incorporating the current best solution x_{best} [5].

The final trial vector is created by setting each parameter to either v_i or x_i for $i = 1, ..., n$, according to a crossover probability hyperparameter. If the value of this new vector is greater than the value of x, x will be replaced with v.

Algorithm 4: Differential Evolution

Input: bounds b_{min}, b_{max}, parameter F, parameter λ
Initial population $x = \mathcal{U}(b_{min}, b_{max})$
Set $x_{best} = \max F(x)$
for $g = 0, 1, 2, ...$ **do**
  for x_i *in* $i = 1, ..., n$ **do**
    Sample vectors $x_2, x_3 = choice(x) \neq x_i$
    Create trial vector
      $v = x_i + \lambda(x_{best} - x_i) + F(x_2 - x_3)$
    Set $v_i = v_i$ if $rand() < p_{cx}$ else x_i
    Replace x_i if $F(v) > F(x_i)$
  end
  Set $x_{best} = \max F(x)$
end

E. Objective Functions

1) Regression: The objective function for the regression models is the residual sum of squares between the observed targets in the dataset, and the targets predicted by the model [1]. The algorithm will maximize the negative of this value; minimizing the function.

$$\|X\mu - y\|_2^2$$

2) Classification: The objective function for the binary classification models is the cross entropy function. The predictions are first passed through a sigmoid function. The algorithm will maximize this value.

$$\sum_{i=1}^{N} y \log X\mu + (1 - y) \log 1 - X\mu$$

3) Clustering: The objective function for the clustering models will compute the inertia value of the reference vectors. The algorithm will maximize the negative of this value; minimizing the function.

IV. Experiments and Results

A. Regression

1) Toy Problems: Each of the four algorithms was used to evolve a linear regression model of data created using [1]'s `make_regression` function. The data consists of 2500 samples with 100 features. The data is split into training and testing sets with a `test_size` of 0.3. The regression metrics calculated are R^2 score, explained variance score, and root-mean-square error (RMSE). A `LinearRegression` model from [1] is trained on the same data for comparison (see Table

I). Finally, cross-validation is performed on all five models to get a final R^2 score for each (see Table II). See Fig. 3 in the appendix for plots of the model score curves.

	R2 Score	Explained Variance Score	RMSE
Linear Regression	0.93038	0.93067	104.26580
Simple ES	0.92989	0.93018	104.62999
Genetic ES	0.92876	0.92899	105.47040
Natural ES	0.93054	0.93083	104.14309
DE	0.92932	0.92969	105.05684

TABLE I
TOY REGRESSION METRICS

	Cross Validated R2 Score
Linear Regression	0.93710 (+/- 0.00627)
Simple ES	0.93438 (+/- 0.00435)
Genetic ES	0.93192 (+/- 0.00824)
Natural ES	0.93710 (+/- 0.00634)
DE	0.93535 (+/- 0.00774)

TABLE II
TOY REGRESSION METRICS

We can see from the results that the models are all fairly close in performance with this data. Due to the inherent randomness of the evolutionary algorithms, they may produce a different set of parameters with a different random seed. The NES model may have outperformed standard Linear Regression after one trial, but we should not make conclusions from that alone. [1]'s `cross_val_score` was used to perform cross-validation. We see NES and Linear Regression have almost equal performance, and the other are slightly behind.

2) Ground State Energies of 16242 Molecules: Each model was tested on a real world dataset consisting of the ground state energies of 16242 molecules calculated by quantum mechanical simulations [2]. There are 1275 columns used that act as molecular features, with the final column being the atomization energy as calulated by simulate [2]. The inspiration behind this dataset stems from the problem that simulations of molecular properties are computationally expensive [2]. Being able to use a model to predict molecular properties could open up many possibilities in computational design and discovery of molecules, compounds and new drugs [2].

Initial results with `LinearRegression` gave an overfitted model with a R^2 score of -3685.75153. Using principal component analysis (PCA), it was determined that 50 components gave an acceptable amount of explained variance of 0.99448 (see Fig. 1). Kernel PCA (kPCA) was performed in addition to standard PCA for feature reduction. kPCA resulted in more variance being captured in the same number of components, so it was used for the final results. The first 16 components and their relation to the energy levels for both PCA and kPCA can be seen in Figures 4 and 5 in the appendix.

The trials remain the same from the toy example. Metrics can be seen in tables III and IV. Score curves can be seen in Figure 6 in the appendix. The Differential Evolution model slightly outperformed Linear Regression after cross-validation.

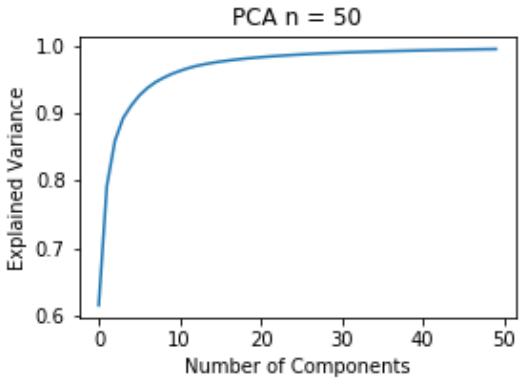

Fig. 1. Molecules - Explained Variance

	R2 Score	Explained Variance Score	RMSE
Linear Regression	0.97600	0.97601	0.56078
Simple ES	0.97305	0.97311	0.59432
Genetic ES	0.96871	0.96872	0.64042
Natural ES	0.94426	0.94866	0.85473
DE	0.97600	0.97600	0.56087

TABLE III
MOLECULE REGRESSION METRICS

B. Classification

1) Toy Problems: Each of the four algorithms was used to evolve a linear classification model of data created using [1]'s `make_classification` function. The data consists of 2500 samples with 100 features. The data is split into training and testing sets with a `test_size` of 0.3. A classification report and PR curve are generated for each model. A `LogisticRegression` model from [1] is trained on the same data for comparison (see Table V). Finally, cross-validation is performed on all five models to get a final accuracy score for each (see Table VI). See Fig. 7 in the appendix for plots of the model PR curves.

We can see that standard Logistic Regression has better

	Cross Validated R2 Score
Linear Regression	0.97651 (+/- 0.01405)
Simple ES	0.97382 (+/- 0.01514)
Genetic ES	0.96069 (+/- 0.01815)
Natural ES	0.93126 (+/- 0.03337)
DE	0.97654 (+/- 0.01404)

TABLE IV
MOLECULE REGRESSION METRICS

	Precision	Recall	F1-Score	Accuracy
Logistic Regression	0.96	0.96	0.96	0.96
Simple ES	0.94	0.94	0.94	0.94
Genetic ES	0.92	0.92	0.92	0.92
Natural ES	0.94	0.94	0.94	0.94
DE	0.89	0.89	0.89	0.89

TABLE V
TOY CLASSIFICATION METRICS

	Cross-Validated Accuracy
Logistic Regression	0.96080 (+/- 0.01229)
Simple ES	0.93000 (+/- 0.02400)
Genetic ES	0.93560 (+/- 0.00993)
Natural ES	0.95880 (+/- 0.01353)
DE	0.91080 (+/- 0.03825)

TABLE VI
TOY CLASSIFICATION METRICS

performance on this dataset.

2) Predicting a Pulsar Star: The HTRU2 is a dataset which describes a sample of pulsar candidates collected during the High Time Resolution Universe Survey [3]. Pulsars are a rare type of Neutron start that produce radio emission detectable here on Earth [3]. They are of considerable scientific interest as probes of space-time, the interstellar medium, and states of matter [3]. Each candidate is described by 8 continuous variables and a single binary class variable [3]. The experiments are the same as the toy problem; metrics can be seen in tables VII and VIII, and PR curves can be seen in Figure 8 in the appendix.

	Precision	Recall	F1-Score	Accuracy
Logistic Regression	0.96	0.91	0.93	0.98
Simple ES	0.96	0.92	0.94	0.98
Genetic ES	0.96	0.90	0.93	0.98
Natural ES	0.96	0.90	0.93	0.98
DE	0.96	0.91	0.93	0.98

TABLE VII
PULSAR CLASSIFICATION METRICS

	Cross-Validated Accuracy
Logistic Regression	0.97810 (+/- 0.00488)
Simple ES	0.97838 (+/- 0.00639)
Genetic ES	0.97709 (+/- 0.00469)
Natural ES	0.97620 (+/- 0.00546)
DE	0.97888 (+/- 0.00440)

TABLE VIII
PULSAR CLASSIFICATION METRICS

The pulsar dataset is unbalanced; with 16259 negative samples and 1639 negative samples. The single trials give every model equal precision and accuracy, but differing recall and F1-scores. The simle ES model found parameters that slightly outperformed Logistic Regression in recall and F1-score, with equal precision and accuracy. After cross-validation, the simple ES and DE models both have higher accuracy than Logistic Regression, with DE on top.

C. Clustering

Each of the four algorithms was used to evolve a set of reference vectors that cluster the data created using [1]'s `make_blobs` function. The data consists of 250 samples with 2 features and 10 clusters. Homogeneity, completeness, v-measure, and inertia are computed for each model, and scores are graphed (see Figure 9 in the appendix). A `KMeans` model from [1] is used on the same data for comparison (see Table IX). The clusters can be seen in Figure 2.

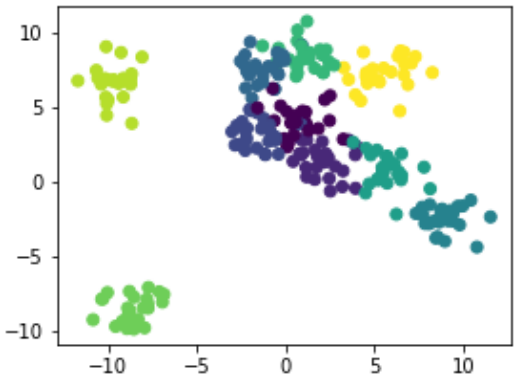

Fig. 2. Clusters

We can see the NES model created clusters with the highest homogeneity, completeness, and V-measure. The DE model has the lowest inertia, or within-cluster sum-of-squares. The DE model outperformed k-means in every metric computed with this dataset. An interesting thing to notice is that the simple ES model created a cluster that has nothing assigned to it. There were no constraints against this in the objective function, so the algorithm got stuck in a local optimum that includes that empty cluster.

	Homogeneity	Completeness	V-Measure	Inertia
K-Means	0.88091	0.88220	0.88156	432.20263
Simple ES	0.84160	0.89568	0.86780	539.64422
Genetic ES	0.85756	0.86349	0.86051	506.09878
Natural ES	0.90360	0.90441	0.90401	448.04099
DE	0.89660	0.89728	0.89694	431.09699

TABLE IX
TOY CLUSTERING METRICS

V. CONCLUSIONS

Two evolutionary optimization techniques were introduced: Evolution Strategies and Differential Evolution. Three variations of the Evolution Strategy technique were used: simple/naive ES, a genetic algorithm ES, and a natural ES. Normally used for black-box optimization, the algorithms were instead trialed against traditional regression, classification, and clustering techniques; namely linear regression, logistic regression, and k-means clustering. Experiments were ran on toy data generated by [1], and real datasets from [2] and [3]. The models generated were shown to have acceptable performance in most cases, and superior performance in others.

VI. FUTURE WORK

There is a large amount of future work that could improve upon these algorithms, and there are many variations on the evolution strategy that could be implemented. Hyperparameter optimization could be performed to fine-tune each model for a particular dataset. The effect each parameter has on the performance of the algorithm was not explored deeply.

An important property for an optimizer is to guarantee finding the global optimum. This constraint was not satisfied in thie work, as the algorithms would reliably settle in a local optimum. Many objective functions exist for the tasks of regression, classification, and clustering. This work only presented a single one for each.

Rather than evolving models directly, evolutionary algorithms are commonly used to evolve neural network parameters. [4] explored Evolution Strategies as an alternative to popular reinforcement-learning techniques. There are many interesting problems that can be explored.

REFERENCES

[1] 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.

[2] B. Himmetoglu, "Tree based machine learning framework for predicting ground state energies of molecules," *The Journal of Chemical Physics*, vol. 145, p. 134101, Oct 2016.

[3] S. C. J. M. B. J. D. K. R. J. Lyon, B. W. Stappers, "Fifty years of pulsar candidate selection: From simple filters to a new principled real-time classification approach," *Monthly Notices of the Royal Astronomical Society*, vol. 459, pp. 1104–1123.

[4] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," 2017.

[5] R. Storn and K. Price, "Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces," *J. of Global Optimization*, vol. 11, p. 341–359, Dec. 1997.

[6] D. Wierstra, T. Schaul, T. Glasmachers, Y. Sun, and J. Schmidhuber, "Natural evolution strategies," 2011.

[7] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing.," *Science*, vol. 220 4598, pp. 671–80, 1983.

[8] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95 - International Conference on Neural Networks*, vol. 4, pp. 1942–1948 vol.4, Nov 1995.

[9] H.-G. Beyer and H.-P. Schwefel, "Evolution strategies –a comprehensive introduction," *Natural Computing: An International Journal*, vol. 1, p. 3–52, May 2002.

[10] D. Ha, "A visual guide to evolution strategies," *blog.otoro.net*, 2017.

APPENDIX

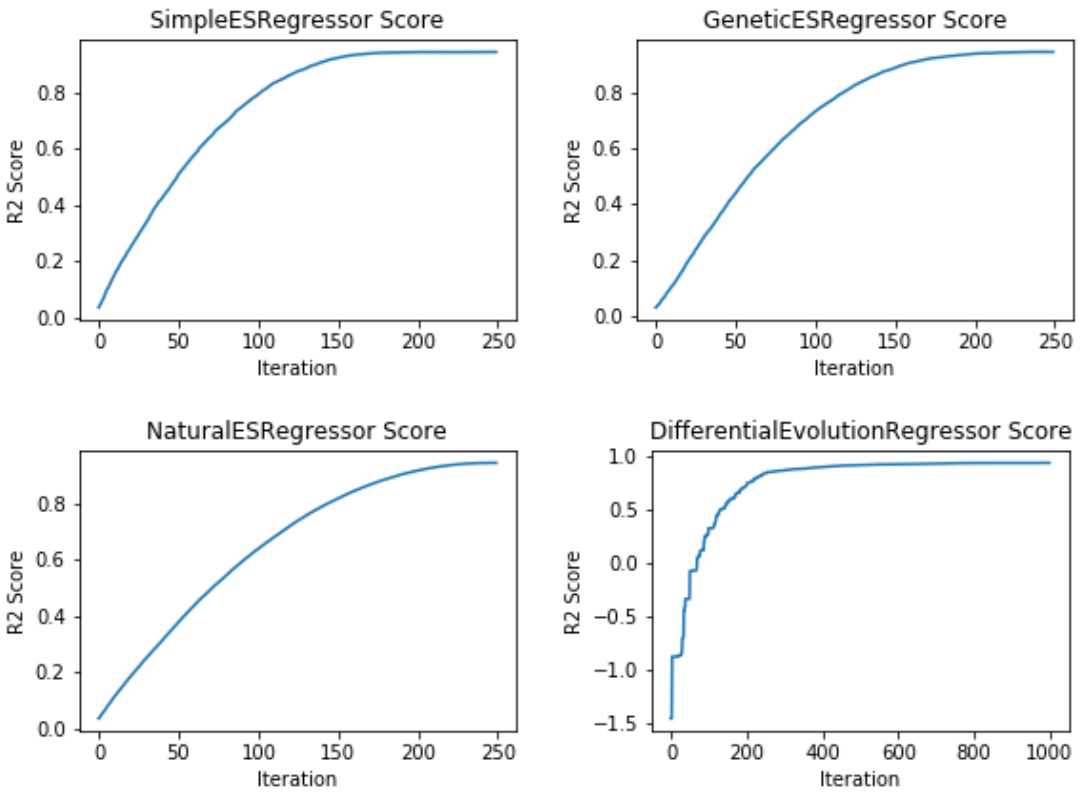

Fig. 3. Toy regression model score curves

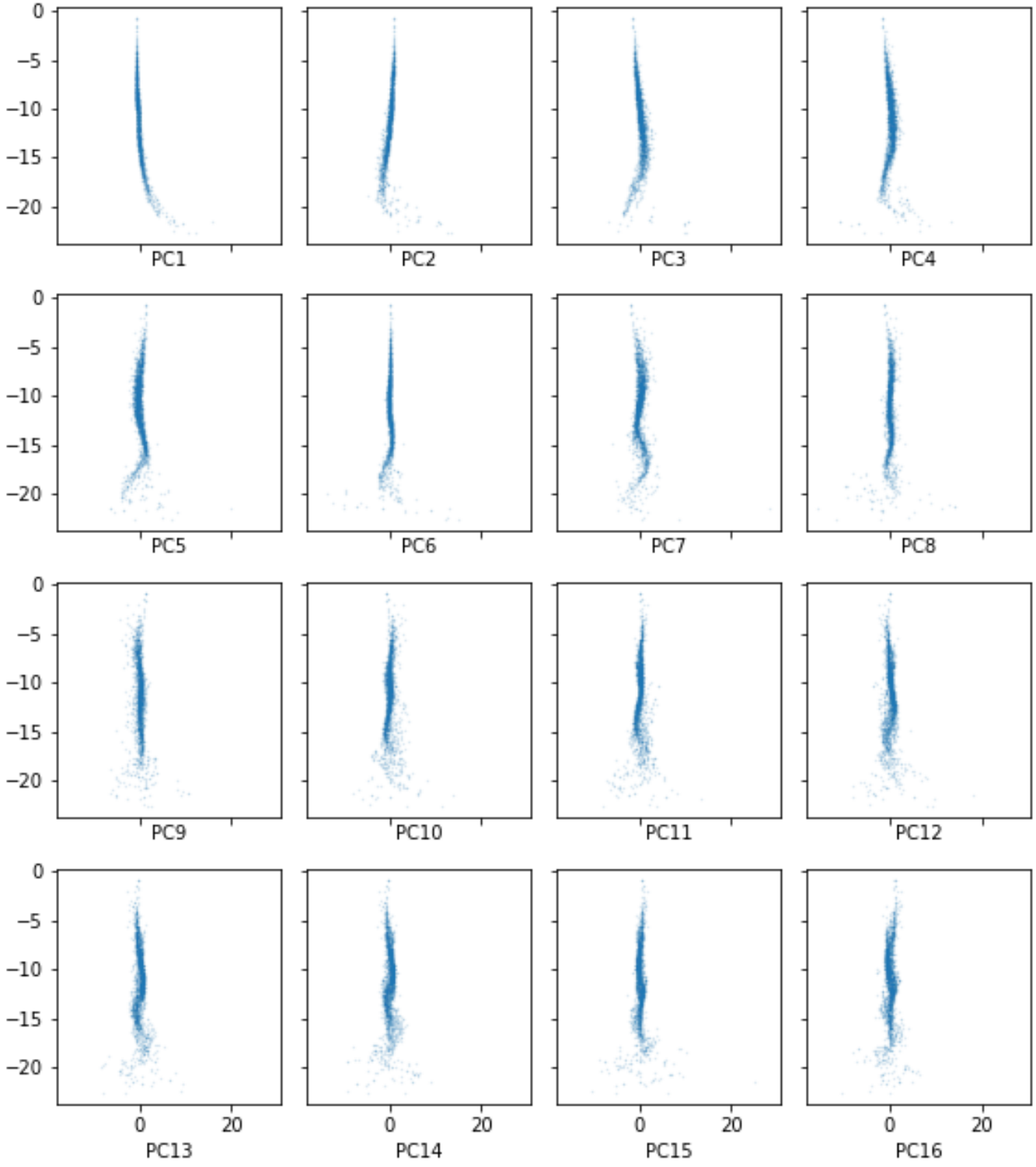

Fig. 4. Molecules - PCA

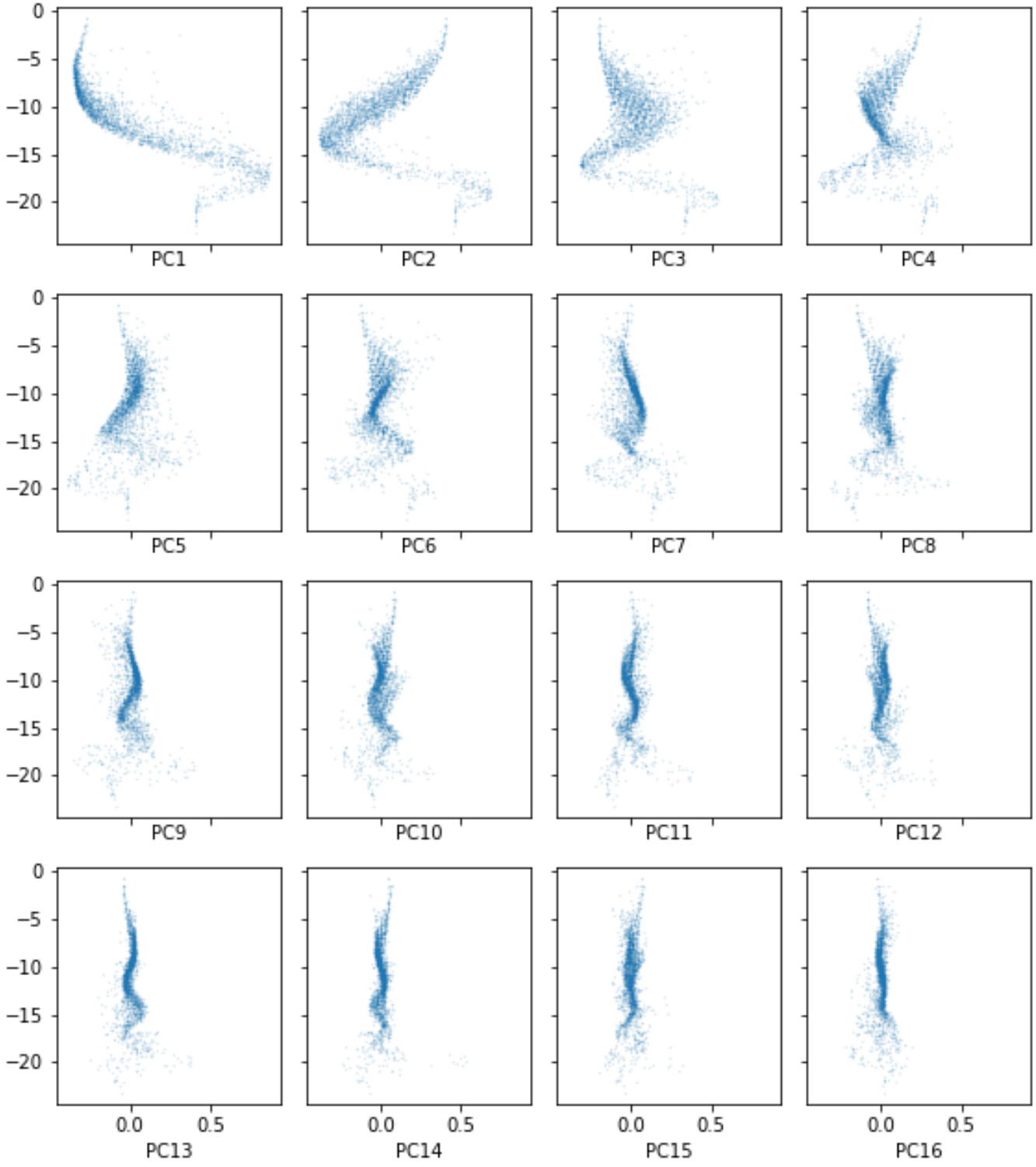

Fig. 5. Molecules - kPCA

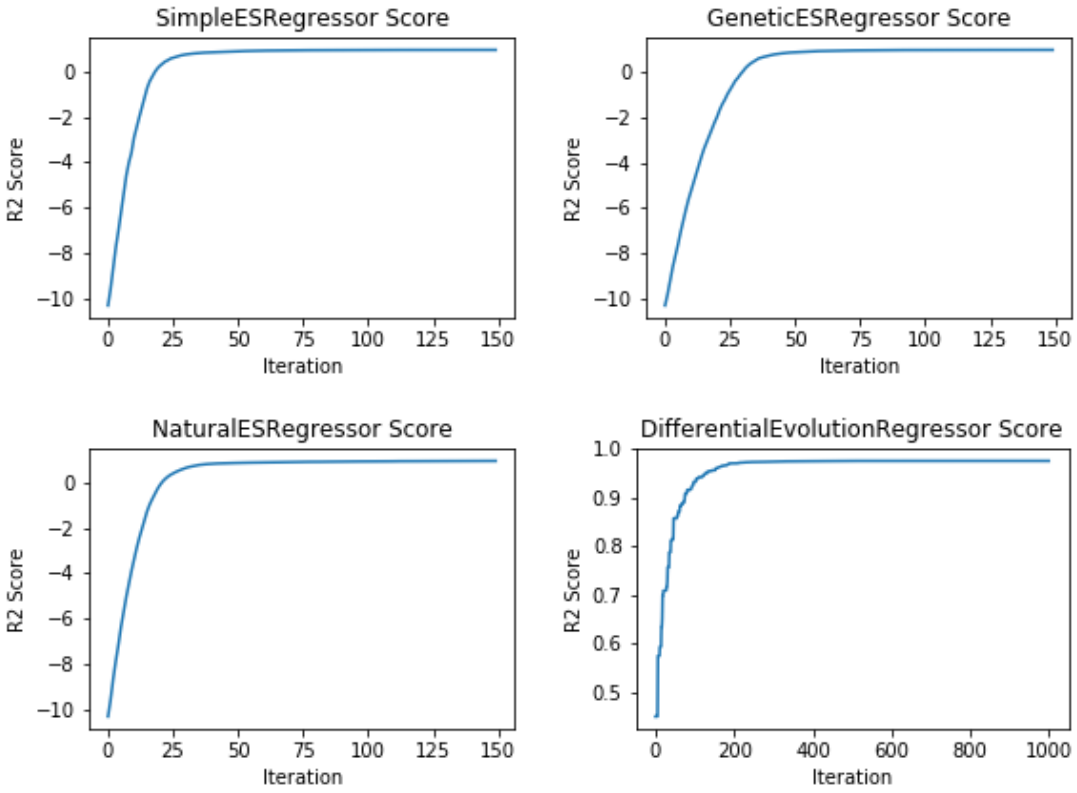

Fig. 6. Molecule regression model score curves

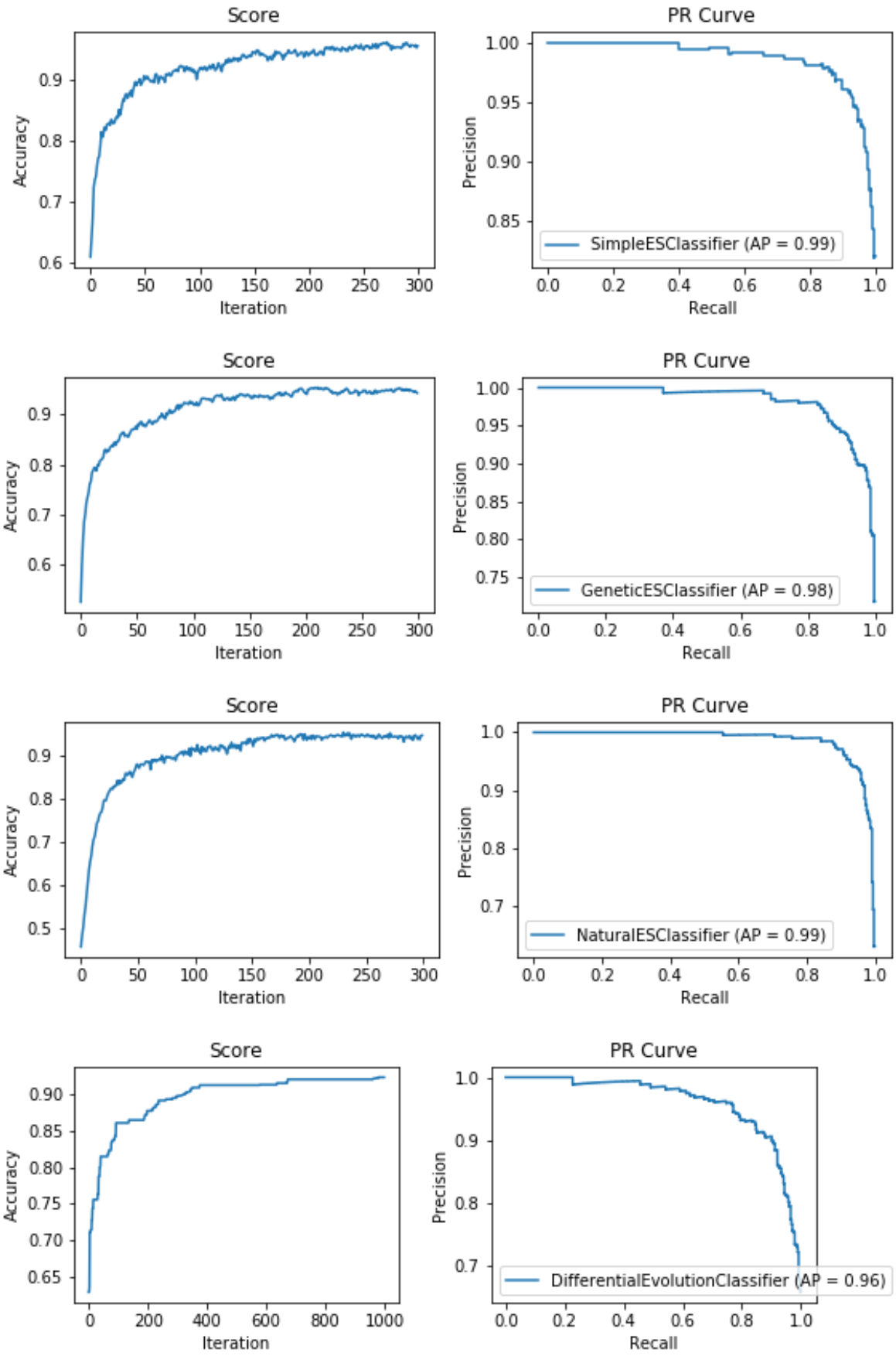

Fig. 7. Toy classification model score curves

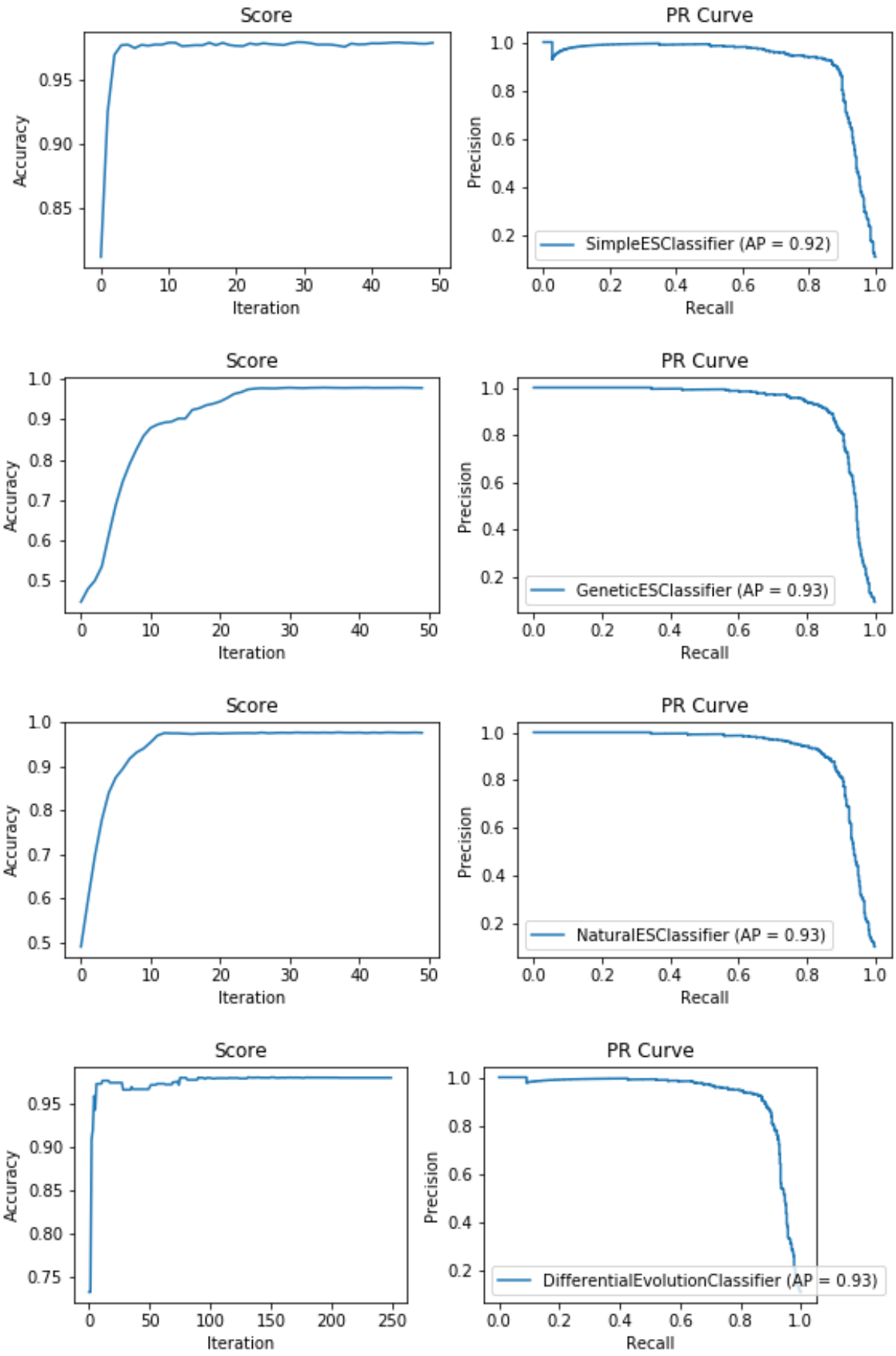

Fig. 8. Pulsar classification model score curves

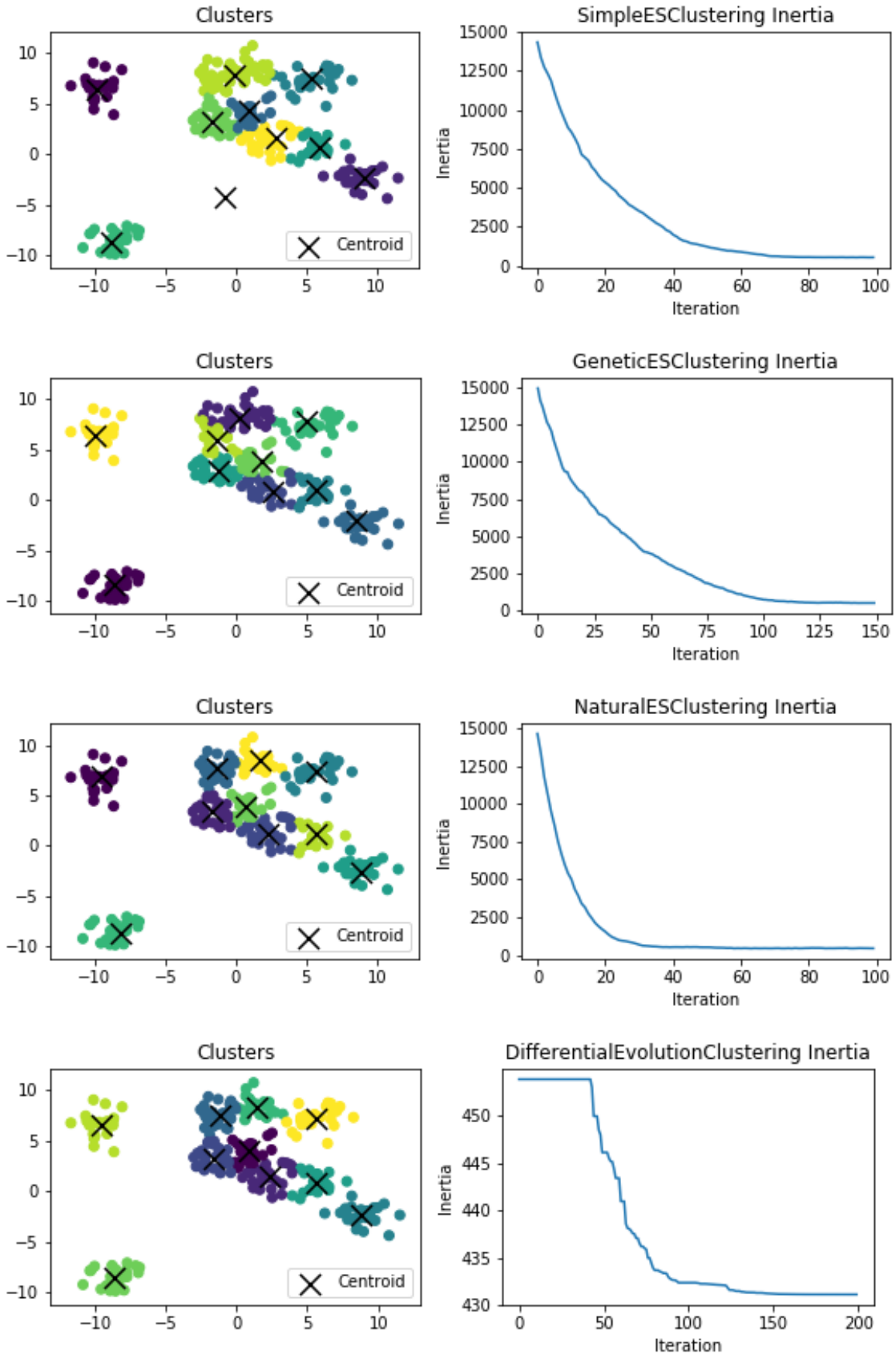

Fig. 9. Toy clustering