evoRF: an evolutionary approach to Random Forests

Diogo Ramos, Davide Carneiro and Paulo Novais

Abstract Machine Learning is a field in which significant steps forward have been taken in the last years, resulting in a wide variety of available algorithms, for many different problems. Nonetheless, most of these algorithms focus on the training of static models, in the sense that the model stops evolving after the training phase. This is increasingly becoming a limitation, especially in an era in which datasets are increasingly larger and may even arrive as sequential streams of data. Frequently re-training a model, in these scenarios, is not realistic. In this paper we propose evoRF: a combination of a Random Forest with an evolutionary approach. Its key innovative aspect is the evolution of the weights of the Random Forest over time, as new data arrives, thus making the forest's voting scheme adapt to the new data. Older trees can also be replaced by newly trained ones, according to their accuracy, ensuring that the ensemble remains up to date without requiring a whole retraining.

Key words: Random Forest, Optimization, Genetic Algorithms, Online Learning

1 Introduction

Decision Trees[1] are among the most popular predictive models used in Machine Learning nowadays. On the one hand, the models that result from the training of a Decision Tree are generally simple to interpret and understand[2, 3] as they can be represented as boolean conditions, as opposed to other approaches such as Neural

Diogo Ramos, Davide Carneiro
CIICESI/ESTG - Polytechnic Institute of Porto, Felgueiras, Portugal, e-mail: {8150101,dcarneiro}@estg.ipp.pt

Paulo Novais
Algoritmi Centre/Department of Informatics, Universidade do Minho, Portugal e-mail: pjon@di.uminho.pt

Networks. On the other hand, Decision Tree algorithms tend to behave fairly well with large datasets and can be used for both classification and regression [4, 5].

A Decision Tree model can be defined as an acyclic graph upon which decisions can be made by following the branches of the tree. The tree is constituted by nodes (in which branching takes place) and leafs, which contain the answer to the problem. When searching for an answer, a feature of the feature vector is considered in each node of the tree. If the value of the feature is below a given threshold, the path follows the left branch. Otherwise, the right branch is followed.

The search for an answer ends when a leaf is reached. The leaf may contain the label of a class in the case of a classification problem, or a value in the case of a regression problem[3].

In the last years, Decision Trees have gained renewed interest in the context of meta-models[6]. Meta-models, also known as Ensembles, are Machine Learning models that are based on the combination of a number of weak models, generally trained with a sub-set of the data of the feature vector. The underlying assumption is that a large group of simple and efficient models will have higher accuracy than one large heavy model.

One of the most widely used ensemble algorithms is the Random Forest[7], in which multiple Decision Trees are used as weak learners. These trees are purposely made simple during training, namely by limiting its depth (branching is stopped early) or by limiting the amount of data or the feature vector used in each tree. Each tree is thus trained on a different set of data, a process known as Bagging.

In this paper we propose a meta-model that improves the result of the training of a Random Forest through the use of an evolutionary approach. After the training of the ensemble, an evolutionary process takes place with the goal to optimize the weight of each tree in the voting scheme of the forest, driven by the goal to minimize error measures such as the RMSE or accuracy. The output is a vector of weights that is then used by the Random Forest to make a prediction. As opposed to other tree-based ensembles, the proposed meta-model is especially interesting in the domain of online learning, in which data changes gradually, such as in streams of data from social media, as well as in very large datasets whose size might be a challenge for traditional approaches. In these scenarios, the weights of the models in the ensemble can be adapted dynamically through the proposed evolutionary process, for a new set of test data. Moreover, as data flows, new trees can be trained and added to the model pool, eventually replacing older trees according to their weight in the ensemble. This ensures that the ensemble remains up-to-date without the need to re-train it with all the data.

2 Methodology

As put forward in Section 1, in this paper we propose an evolutionary approach to improve the result of an ensemble by optimizing the weights of each model on the voting scheme. The methodology followed is described in this section.

The process starts with the random splitting of the dataset into three different sets: *training*, *validation* and *test*. A Random Forest with t trees is trained using the training and validation sets, with the sklearn 0.20.3 package. In order to make each tree in the ensemble a weak learner, each is trained with a subset of the data and of the feature vector. These hyperparameters are deemed, respectively, sr (sample rate) and csr (column sample rate). As for the remaining parameters, the default ones in the package are used.

After the training of the ensemble finishes, the evolutionary process takes place. This process has as main goal to find the optimum weight of each tree in the ensemble. This evolutionary process is often time-consuming. However, this is not a drawback in this case as the ensemble can make predictions as soon as all the trees are trained, like a traditional Random Forest. The evolutionary process starts and the new weights of the trees are updated in real-time and used as soon as better solutions are found.

The evolutionary process starts with the creation of w random solutions for the problem, that is, w weights vectors. This first group of solutions constitutes the first generation g_1 of a group of G generations that will be created throughout the process. Each weight $w_{g,s,i}, g \in [1..G], s \in [1..w], i \in [1..t]$ represents the weight of the tree i in the solution s of generation g ($\sum_{i=1}^{t} w_{g,s,i} = 1, \forall s \in [1..w], \forall g \in [1..G]$).

The fitness of each solution in the current generation is then calculated. This depends on the type of the ensemble. In a classification problem, the fitness is given by the accuracy of the predictions while in a regression problem the fitness is given by the RMSE. Both are calculated using the *test* dataset, which acts as holdout. This results in a fitness vector f of size t, in which each fitness value $f_{g,s}, g \in [1..G], s \in [1..w]$ represents the fitness of a given solution s in generation g.

After the fitness is calculated, the b solutions with the highest fitness are selected to be reproduced, thus originating the first solutions of a new generation. A group of m new solutions are added to this new generation, based on the mutation operator. This operator takes as input a solution (selected randomly from the b best solutions), generates a random number $r, r \in [0..mr]$ (in which mr denotes the mutation rate), and randomly selects two positions in the weights vector i and j. The solution is thus mutated according to $w_{g,s,i} \leftarrow w_{g,s,i} + mr$ and $w_{g,s,j} \leftarrow w_{g,s,j} - mr$. This is repeated for the same chromosome until the sum of the weights of the vector equals 1, i.e., until the solution is a valid one ($\sum_{i=1}^{t} w_{g,s} = 1$). Another group of r random solutions are also added to this new generation, created as detailed before. The aim is to maintain a certain level of diversity in each generation. Another common operator is crossover. This is not used in this work as it would potentially violate the validity of the solutions crossed, with a very low probability of obtaining valid solutions.

Once the new generation is complete, including not only the b best solutions but also the m solutions generated by mutation and the r solutions generated randomly ($b + m + r = w$), the process returns to the fitness evaluation stage and repeats.

The process stops when a limit of generations l is reached, or when the value of the best fitness found so far has not improved by δ over the last n generations. The output of the process is the weights vector $w_{g,s}$ with the highest fitness value in all generations ($g \in [1..G], s \in [1..w]$).

The main hyperparameters that can be tuned when using this algorithm are thus:

- sr - sample rate
- csr - column sample rate
- w - number of solutions in each generation
- t - number of trees in the ensemble
- b - number of best solutions to consider in each generation
- mr - mutation rate, defining how much each solution should mutate
- m - number of new solutions to generate through mutation, in each generation
- r - number of new random solutions to generate, in each generation
- l - number of iterations (stopping criterion)
- δ and n - define the reaching of convergence (stopping criterion)

3 Results

In order to validate evoRF, it was implemented and tested with the well-known dataset MNIST of handwritten digits. It has a total of 70.000 examples of handwritten digits. The digits have been size-normalized and centered in a fixed-size image, in a 28x28 grid in grayscale. Researchers using this dataset generally carry out some pre-processing tasks which result in lower error rates, such as deskewing, noise removal, blurring or pixel shifts. Since the goal of this paper is to validate evoRF and not to achieve the highest accuracy possible with this dataset, the data were not pre-processed and was used as is.

This dataset was thus used to train a classification ensemble, following the methodology detailed in Section 2. An ensemble with 25 trees was trained using 66% of the data, with 29% used for validation and 5% used for testing and for making the solutions evolve. A population of size 8 was used. In each round, the best 5 solutions were kept while 2 new solutions were added through mutation (by mutating one of the 5 selected solutions). Finally, 1 new random solution is also added in each generation, to maintain diversity and explore the search space. The process was run by 100 iterations. Figure 1 shows the evolution of accuracy throughout the 100 generations. The solid line represents the accuracy of the best solution found so far, i.e., the best combination of weights for the models given the test data, while the dotted line represents the average accuracy of all solutions in a generation.

Figure 2 compares the weights of the best solution of the first generation, when they were randomly generated, with the weights of the best solution of the 100th generation. A change in weights is evident, which results from the evolution process. If a new set of test data were to be presented, from a more recent stream of data, the result of the evolution would be different according to how good each model is for that particular set of data.

In order to assess the quality of the evolved ensemble, we compare its accuracy with that of a standard Random Forest implemented in python using the same 25 trees and a voting scheme in which each tree has the same weight (0.04). We also

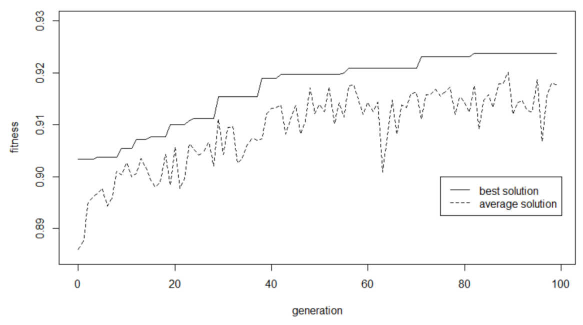

Fig. 1 Evolution of accuracy over time.

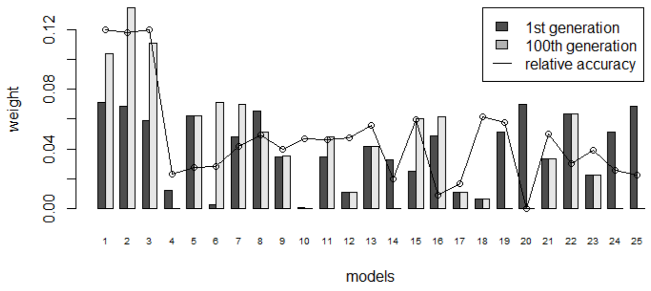

Fig. 2 Weights of each model in the best solutions of the first and last generations, and relative accuracy of each model.

compare it with the H2O's implementation of a Distributed Random Forest algorithm. The standard implementation achieves an accuracy of 90.86% while H2O's DRF achieves an accuracy of 93.45%. In comparison, evoRF shows an accuracy of 92.37%, which is better than the standard RF and close to that of H2O.

4 Conclusions and Future Work

The requirements of Machine Learning algorithms have been changing in the last years, due to two main reasons: datasets are growing increasingly larger and data sometimes arrive in streaming rather than in batch. These calls for algorithms that can deal with the challenges that this changes entail. In this paper we proposed evoRF: a Random Forest with a voting scheme whose weights can be adapted through an evolutionary process to better adjust to new sets of data. Moreover, new trees can be trained with more recent data, to be included in the ensemble, eventually replacing older and/or less suited models. Models can also be trained with different subsets of the data, representing different views on the problem (e.g. long-term vs. short term views on the problem).

This approach was devised to be used in domains in which data and its patterns change frequently, such as in social networks, in which changes may occur within a day and models become obsolete really fast. This approach may thus allow ensembles to remain up to date, whether by continuously adjusting the weights of existing models or by changing the pool of models. Its main characteristics are: 1) the ensemble does not need to be retrained when data changes significantly; 2) adaptation takes place by changing the weights of each model and does not require changing the trees; 3) new data/patterns can be incorporated by training new weak learners that are included in the ensemble. Moreover, as with traditional Random Forest algorithms, this approach is easy to parallelize and/or distribute (as are other evolutionary algorithms). All these factors make this a computationally simple approach, especially when compared to other methods that change the structure of existing trees according to some heuristic.

We are now working on the implementation of an online service that continuously receives data and carries out the training of models and an ongoing evolution of the weights, in order for the ensemble to adapt to these new data. In the future we will study how this approach behaves with large streams of data from social media, especially when there are new classes. We will, specifically, assess the time it takes for the ensemble to adapt to significant changes in the data, such as when a social event takes place that significantly alters the way the network behaves, namely when predicting interactions.

Acknowledgments

This work has been supported by FCT – Fundação para a Ciência e Tecnologia within the Project Scope: UID/CEC/00319/2019.

References

1. Safavian, S.R., Landgrebe, D.: A survey of decision tree classifier methodology. IEEE transactions on systems, man, and cybernetics **21**(3) (1991) 660–674
2. Janikow, C.Z.: Fuzzy decision trees: issues and methods. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) **28**(1) (1998) 1–14
3. Singh, A., Thakur, N., Sharma, A.: A review of supervised machine learning algorithms. In: 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom), IEEE (2016) 1310–1315
4. Gehrke, J., Ramakrishnan, R., Ganti, V.: Rainforest-a framework for fast decision tree construction of large datasets. In: VLDB. Volume 98. (1998) 416–427
5. Ranka, S., Singh, V.: Clouds: A decision tree classifier for large datasets. In: Proceedings of the 4th Knowledge Discovery and Data Mining Conference. Volume 2. (1998) 8
6. Vilalta, R., Drissi, Y.: A perspective view and survey of meta-learning. Artificial intelligence review **18**(2) (2002) 77–95
7. Biau, G., Scornet, E.: A random forest guided tour. Test **25**(2) (2016) 197–227