Chapter 8

TPOT: A Tree-based Pipeline Optimization Tool for Automating Machine Learning

Randal S. Olson and Jason H. Moore

Abstract

As data science becomes increasingly mainstream, there will be an ever-growing demand for data science tools that are more accessible, flexible, and scalable. In response to this demand, automated machine learning (AutoML) researchers have begun building systems that automate the process of designing and optimizing machine learning pipelines. In this chapter we present TPOT v0.3, an open source genetic programming-based AutoML system that optimizes a series of feature preprocessors and machine learning models with the goal of maximizing classification accuracy on a supervised classification task. We benchmark TPOT on a series of 150 supervised classification tasks and find that it significantly outperforms a basic machine learning analysis in 21 of them, while experiencing minimal degradation in accuracy on 4 of the benchmarks—all without any domain knowledge nor human input. As such, genetic programming-based AutoML systems show considerable promise in the AutoML domain

8.1 Introduction

Machine learning is commonly described as a "field of study that gives computers the ability to learn without being explicitly programmed" [18]. Despite this

common claim, experienced machine learning practitioners know that designing effective machine learning pipelines is often a tedious endeavor, and typically requires considerable experience with machine learning algorithms, expert knowledge of the problem domain, and time-intensive brute force search to accomplish [13]. Thus, contrary to what machine learning enthusiasts would have us believe, machine learning still requires considerable explicit programming.

In response to this challenge, several automated machine learning methods have been developed over the years [10]. Over the past several years, we have been developing a Tree-based Pipeline Optimization Tool (TPOT) that automatically designs and optimizes machine learning pipelines for a given problem domain [15], without any need for human intervention. In short, TPOT optimizes machine learning pipelines using a version of genetic programming (GP), a well-known evolutionary computation technique for automatically constructing computer programs [1]. Previously, we demonstrated that combining GP with Pareto optimization enables TPOT to automatically construct high-accuracy *and* compact pipelines that consistently outperform basic machine learning analyses [13]. In this chapter, we extend that benchmark to include 150 supervised classification tasks and evaluate TPOT in a wide variety of application domains ranging from genetic analyses to image classification and more.

8.2 Methods

In the following sections, we provide an overview of the Tree-based Pipeline Optimization Tool (TPOT) v0.3, including the machine learning operators used as genetic programming (GP) primitives, the tree-based pipelines used to combine the primitives into working machine learning pipelines, and the GP algorithm used to evolve said tree-based pipelines. We follow with a description of the datasets used to evaluate the latest version of TPOT in this chapter. TPOT is an open source project on GitHub, and the underlying Python code can be found at `https://github.com/rhiever/tpot`.

8.2.1 Machine Learning Pipeline Operators

At its core, TPOT is a wrapper for the Python machine learning package, scikit-learn [16]. Thus, each machine learning pipeline operator (i.e., GP primitive) in TPOT corresponds to a machine learning algorithm, such as a supervised classification model or standard feature scaler. All implementations of the machine learning algorithms listed below are from scikit-learn (except XGBoost), and we refer to the scikit-learn documentation [16] and [9] for detailed explanations of the machine learning algorithms used in TPOT.

Supervised Classification Operators. DecisionTree, RandomForest, eXtreme Gradient Boosting Classifier (from XGBoost, [3]), LogisticRegression, and KNearestNeighborClassifier. Classification operators store the classifier's predictions as a new feature as well as the classification for the pipeline.

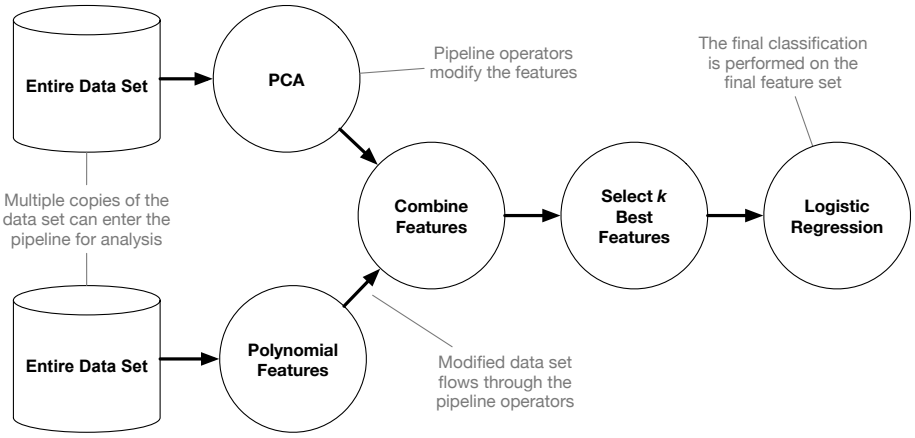

Figure 8.1: An example tree-based pipeline from TPOT. Each circle corresponds to a machine learning operator, and the arrows indicate the direction of the data flow.

Feature Preprocessing Operators. StandardScaler, RobustScaler, MinMaxScaler, MaxAbsScaler, RandomizedPCA [12], Binarizer, and PolynomialFeatures. Preprocessing operators modify the dataset in some way and return the modified dataset.

Feature Selection Operators. VarianceThreshold, SelectKBest, SelectPercentile, SelectFwe, and Recursive Feature Elimination (RFE). Feature selection operators reduce the number of features in the dataset using some criteria and return the modified dataset.

We also include an operator that combines disparate datasets, as demonstrated in Figure 8.1, which allows multiple modified variants of the dataset to be combined into a single dataset. Additionally, TPOT v0.3 does not include missing value imputation operators, and therefore does not support datasets with missing data. Lastly, we provide integer and float terminals to parameterize the various operators, such as the number of neighbors k in the k-Nearest Neighbors Classifier.

8.2.2 Constructing Tree-Based Pipelines

To combine these operators into a machine learning pipeline, we treat them as GP primitives and construct GP trees from them. Figure 8.1 shows an example tree-based pipeline, where two copies of the dataset are provided to the pipeline, modified in a successive manner by each operator, combined into a single dataset, and finally used to make classifications. Other than the restriction that every pipeline must have a classifier as its final operator, it is possible to construct arbitrarily shaped machine learning pipelines that can act on multiple copies of the dataset. Thus, GP trees provide an inherently flexible representation of machine learning pipelines.

In order for these tree-based pipelines to operate, we store three additional variables for each record in the dataset. The "class" variable indicates the true label for each record, and is used when evaluating the accuracy of each pipeline. The "guess" variable indicates the pipeline's latest guess for each record, where the predictions from the final classification operator in the pipeline are stored as the "guess". Finally, the "group" variable indicates whether the record is to be used as a part of the internal training or testing set, such that the tree-based pipelines are only trained on the training data and evaluated on the testing data. We note that the dataset provided to TPOT as training data is further split into an internal stratified 75%/25% training/testing set.

8.2.3 Optimizing Tree-Based Pipelines

To automatically generate and optimize these tree-based pipelines, we use a genetic programming (GP) algorithm [1] as implemented in the Python package DEAP [7]. The TPOT GP algorithm follows a standard GP process: To begin, the GP algorithm generates 100 random tree-based pipelines and evaluates their balanced cross-validation accuracy on the dataset. For every generation of the GP algorithm, the algorithm selects the top 20 pipelines in the population according to the NSGA-II selection scheme [4], where pipelines are selected to simultaneously maximize classification accuracy on the dataset while minimizing the number of operators in the pipeline. Each of the top 20 selected pipelines produce five copies (i.e., offspring) into the next generation's population, 5% of those offspring cross over with another offspring using one-point crossover, then 90% of the remaining unaffected offspring are randomly changed by a point, insert, or shrink mutation (1/3 chance of each). Every generation, the algorithm updates a Pareto front of the non-dominated solutions [4] discovered at any point in the GP run. The algorithm repeats this evaluate-select-crossover-mutate process for 100 generations—adding and tuning pipeline operators that improve classification accuracy and pruning operators that degrade classification accuracy—at which point the algorithm selects the highest-accuracy pipeline from the Pareto front as the representative "best" pipeline from the run.

8.2.4 Benchmark Data

We compiled 150 supervised classification benchmarks[1] from a wide variety of sources, including the UCI machine learning repository [11], a large preexisting benchmark repository from [17], and simulated genetic analysis datasets from [19]. These benchmark datasets range from 60 to 60,000 records, few to hundreds of features, and include binary as well as multi-class supervised classification problems. We selected datasets from a wide range of application domains, including genetic analysis, image classification, time series analysis, and many more. Thus, this benchmark—called the Penn Machine Learning Benchmark (PMLB) [14]—represents a comprehensive suite of tests with which to evaluate automated machine learning systems.

[1] Benchmark data at `https://github.com/EpistasisLab/penn-ml-benchmarks`

8.3 Results

To evaluate TPOT, we ran 30 replicates of it on each of the 150 benchmarks, where each replicate had 8 hours to complete 100 generations of optimization (i.e., $100 \times 100 = 10,000$ pipeline evaluations). In each replicate, we divided the dataset into a stratified 75%/25% training/testing split and used a distinct random number generator seed for each split and subsequent TPOT run.

In order to provide a reasonable control as a baseline comparison, we similarly evaluated 30 replicates of a Random Forest with 500 trees on the 150 benchmarks, which is meant to represent a basic machine learning analysis that a novice practitioner would perform. We also ran 30 replicates of a version of TPOT that randomly generates and evaluates the same number of pipelines (10,000), which is meant to represent a random search in the TPOT pipeline space. In all cases, we measured accuracy of the resulting pipelines or models as balanced accuracy [20], which corrects for class frequency imbalances in datasets by computing the accuracy on a per-class basis then averaging the per-class accuracies. In the remainder of this chapter, we refer to "balanced accuracy" as simply "accuracy."

Shown in Figure 8.2, the average performance of TPOT and a Random Forest with 500 trees is similar on most of the datasets. Overall, TPOT discovered pipelines that perform statistically significantly better than a Random Forest on 21 benchmarks, significantly worse on 4 benchmarks, and had no statistically significant difference on 125 benchmarks. (We determined statistical significance using a Wilcoxon rank-sum test, where we used a conservative Bonferroni-corrected p-value threshold of $< 0.000\overline{333}$ ($\frac{0.05}{150}$) for significance.) In Figure 8.3, we show the distributions of accuracies on the 25 benchmarks that had significant differences, where the benchmarks are sorted by the difference in median accuracy between the two experiments.

Notably, the majority of TPOT's improvements on the benchmarks are quite large, with several ranging from 10%–60% median accuracy improvement over a Random Forest analysis. In contrast, the 4 benchmarks where TPOT experienced a degradation in median accuracy ranged from only 2–5% accuracy degradation. In some cases, TPOT's improvements were made by discovering useful feature preprocessors that allow the models to better classify the data[2], e.g., TPOT discovered that applying a RandomizedPCA feature preprocessor prior to modeling the "Hill_valley" benchmarks allows Random Forests to classify the dataset with near-perfect accuracy. In other cases, TPOT's improvements were made by applying a different model to the benchmark, e.g., TPOT discovered that a k-nearest-neighbor classifier with $k = 10$ neighbors can classify the "parity5" benchmark, whereas a Random Forest consistently achieved 0% accuracy on the same benchmark.

When we compared TPOT to a version of TPOT that uses random search ("TPOT Random" in Figure 8.3), we found that random search typically discovered pipelines that achieve comparable accuracy to pipelines discovered by

[2]Full list: `https://gist.github.com/rhiever/578cc9c686ffd873f46bca29406dde1d`

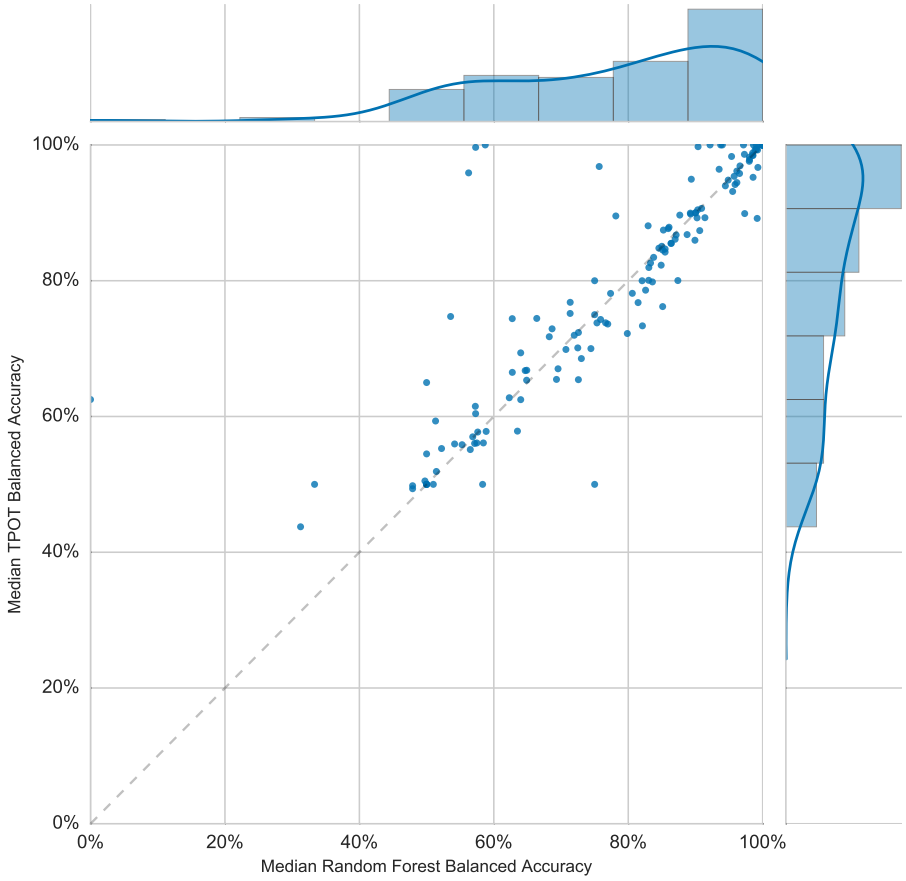

Figure 8.2: Scatter plot showing the median balanced accuracies of TPOT and a Random Forest with 500 trees on the 150 benchmark datasets. Each dot represents the accuracies on one benchmark dataset, and the diagonal line represents the line of parity (i.e., when both algorithms achieve the same accuracy score). Dots above the line represent datasets where TPOT performed better than the Random Forest, and dots below the line represent datasets where Random Forests performed better.

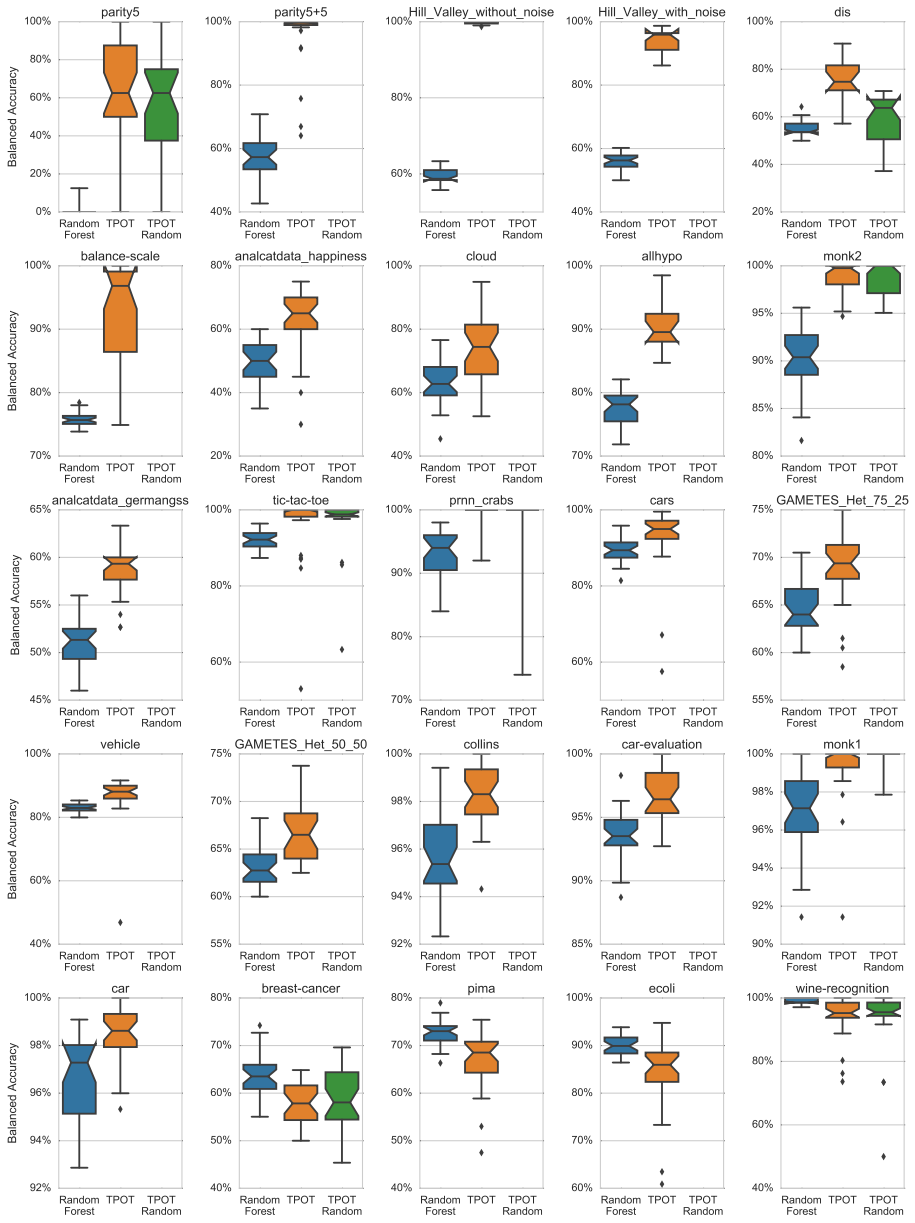

Figure 8.3: Box plots showing the distribution of balanced accuracies for the 25 benchmarks with a significant difference in median accuracy between TPOT and a Random Forest with 500 trees. Each box plot represents 30 replicates, the inner line shows the median, the notches represent the bootstrapped 95% confidence interval of the median, the ends of the box represent the first and third quartiles, and the dots represent outliers.

TPOT, except in the "dis" benchmark where TPOT consistently discovered better-performing pipelines. For 17 of the presented benchmarks, none of the random search runs finished within 24 hours, which we indicated by leaving the box plot blank in Figure 8.3. We found that random search often generated needlessly complex pipelines for the benchmark problems, even when a simple pipeline with a tuned model was sufficient to classify the benchmark problem. Thus, even if random search can sometimes perform as well as TPOT in terms of accuracy, performing a guided search for pipelines that achieve high accuracy with as few pipeline operations as possible still offers considerable advantages in terms of search run-time, model complexity, and model interpretability.

8.4 Conclusions and Future Work

We benchmarked the Tree-based Pipeline Optimization Tool (TPOT) v0.3 on 150 supervised classification datasets and found that it discovers machine learning pipelines that can outperform a basic machine learning analysis on several benchmarks. In particular, we note that TPOT discovered these pipelines without any domain knowledge nor human input. As such, TPOT shows considerable promise in the automated machine learning (AutoML) domain and we will continue to refine TPOT until it consistently discovers human-competitive machine learning pipelines. We discuss some of these future refinements below.

First, we will explore methods to provide sensible initialization [8] for genetic programming (GP)-based AutoML systems such as TPOT. For example, we can use meta-learning techniques to intelligently match pipeline configurations that may work well on the particular problem being solved [6]. In brief, meta-learning harnesses information from previous machine learning runs to predict how well each pipeline configuration will work on a particular dataset. To place datasets on a standard scale, meta-learning algorithms compute meta-features from the datasets, such as dataset size, the number of features, and various aspects about the features, which are then used to map dataset meta-features to corresponding pipeline configurations that may work well on datasets with those meta-features. Such an intelligent meta-learning algorithm is likely to improve the TPOT sensible initialization process.

Furthermore, we will attempt to characterize the ideal "shape" of a machine learning pipeline. In auto-sklearn, [5] imposed a short and fixed pipeline structure of a data preprocessor, a feature preprocessor, and a model. In another GP-based AutoML system, [21] allowed the GP algorithm to design arbitrarily-shaped pipelines and found that complex pipelines with several preprocessors and models were useful for signal processing problems. Thus, it may be vital to allow AutoML systems to design arbitrarily-shaped pipelines if they are to achieve human-level competitiveness.

Finally, genetic programming (GP) optimization methods are typically criticized for optimizing a large population of solutions, which can sometimes be slow and wasteful for certain optimization problems. Instead, it is possible to turn GP's purported weakness into a strength by creating an ensemble out of

the GP populations. [2] explored one such population ensemble method previously with a standard GP algorithm and showed that it significantly improved performance, and it is a natural extension to create ensembles out of TPOT's population of machine learning pipelines.

In conclusion, these experiments demonstrate that there is much to be gained from taking a model-agnostic approach to machine learning and allowing the machine to automatically discover what series of preprocessors and models work best for a given problem domain. As such, AutoML stands to revolutionize data science by automating some of the most tedious—yet most important—aspects of machine learning.

Bibliography

[1] Banzhaf, W., Nordin, P., Keller, R.E., Francone, F.D.: Genetic Programming: An Introduction. Morgan Kaufmann, San Meateo, CA, USA (1998)

[2] Bhowan, U., Johnston, M., Zhang, M., Yao, X.: Evolving diverse ensembles using genetic programming for classification with unbalanced data. Trans. Evol. Comp 17(3), 368–386 (Jun 2013)

[3] Chen, T., Guestrin, C.: Xgboost: A scalable tree boosting system. In: Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 785–794. KDD '16, ACM, New York, NY, USA (2016)

[4] Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation 6, 182–197 (2002)

[5] Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., Hutter, F.: Efficient and robust automated machine learning. In: Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., Garnett, R. (eds.) Advances in Neural Information Processing Systems 28, pp. 2944–2952. Curran Associates, Inc. (2015)

[6] Feurer, M., Springenberg, J.T., Hutter, F.: Initializing bayesian hyperparameter optimization via meta-learning. In: Proceedings of the 29th AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA. pp. 1128–1135 (2015)

[7] Fortin, F.A., De Rainville, F.M., Gardner, M.A., Parizeau, M., Gagné, C.: DEAP: Evolutionary Algorithms Made Easy. Journal of Machine Learning Research 13, 2171–2175 (2012)

[8] Greene, C.S., White, B.C., Moore, J.H.: An expert knowledge-guided mutation operator for genome-wide genetic analysis using genetic programming. In: Pattern Recognition in Bioinformatics, pp. 30–40. Springer Berlin Heidelberg (2007)

[9] Hastie, T.J., Tibshirani, R.J., Friedman, J.H.: The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer, New York, NY, USA (2009)

[10] Hutter, F., Lücke, J., Schmidt-Thieme, L.: Beyond Manual Tuning of Hyperparameters. KI - Künstliche Intelligenz 29, 329–337 (2015)

[11] Lichman, M.: UCI machine learning repository (2013), `http://archive.ics.uci.edu/ml`

[12] Martinsson, P.G., Rokhlin, V., Tygert, M.: A randomized algorithm for the decomposition of matrices. Applied and Computational Harmonic Analysis 30, 47–68 (2011)

[13] Olson, R.S., Bartley, N., Urbanowicz, R.J., Moore, J.H.: Evaluation of a tree-based pipeline optimization tool for automating data science. In: Proceedings of the Genetic and Evolutionary Computation Conference 2016. pp. 485–492. GECCO '16, ACM, New York, NY, USA (2016)

[14] Olson, R.S., La Cava, W., Orzechowski, P., Urbanowicz, R.J., Moore, J.H.: PMLB: A Large Benchmark Suite for Machine Learning Evaluation and Comparison. arXiv e-print. https://arxiv.org/abs/1703.00512 (2017)

[15] Olson, R.S., Urbanowicz, R.J., Andrews, P.C., Lavender, N.A., Kidd, L.C., Moore, J.H.: Applications of Evolutionary Computation: 19th European Conference, EvoApplications 2016, Porto, Portugal, March 30 — April 1, 2016, Proceedings, Part I, chap. Automating Biomedical Data Science Through Tree-Based Pipeline Optimization, pp. 123–137. Springer International Publishing (2016)

[16] 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)

[17] Reif, M.: A comprehensive dataset for evaluating approaches of various meta-learning tasks. In: First International Conference on Pattern Recognition and Methods (ICPRAM) (2012)

[18] Simon, P.: Too big to ignore: the business case for big data. Wiley & SAS Business Series, Wiley, New Delhi (2013)

[19] Urbanowicz, R.J., Kiralis, J., Sinnott-Armstrong, N.A., Heberling, T., Fisher, J.M., Moore, J.H.: GAMETES: a fast, direct algorithm for generating pure, strict, epistatic models with random architectures. BioData Mining 5 (2012)

[20] Velez, D.R., White, B.C., Motsinger, A.A., Bush, W.S., Ritchie, M.D., Williams, S.M., Moore, J.H.: A balanced accuracy function for epistasis modeling in imbalanced datasets using multifactor dimensionality reduction. Genetic Epidemiology 31(4), 306–315 (2007)

[21] Zutty, J., Long, D., Adams, H., Bennett, G., Baxter, C.: Multiple objective vector-based genetic programming using human-derived primitives. In: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation. pp. 1127–1134. GECCO '15, ACM, New York, NY, USA (2015)