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

Randal S. Olson OLSONRAN@UPENN.EDU and **Jason H. Moore** JHMOORE@UPENN.EDU
Institute for Biomedical Informatics, University of Pennsylvania, Philadelphia, PA, USA

Abstract

As data science becomes more 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 paper we present TPOT, 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 22 of them, while experiencing minimal degradation in accuracy on 5 of the benchmarks—all without any domain knowledge nor human input. As such, GP-based AutoML systems show considerable promise in the AutoML domain.

Keywords: automated machine learning, hyperparameter optimization, pipeline optimization, genetic programming, Pareto optimization, data science, Python

1. Introduction

Machine learning is commonly described as a "field of study that gives computers the ability to learn without being explicitly programmed" (Simon, 2013). Despite this common claim, 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 brute force search to accomplish (Olson et al., 2016a). 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 (Hutter et al., 2015). Over the past year, we have been developing a Tree-based Pipeline Optimization Tool (TPOT) that automatically designs and optimizes machine learning pipelines for a given problem domain (Olson et al., 2016b), 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 (Banzhaf et al., 1998). 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 (Olson et al., 2016a). In this paper, 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.

2. Methods

In the following sections, we provide an overview of the Tree-based Pipeline Optimization Tool (TPOT), 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 data sets used to evaluate the latest version of TPOT in this paper. TPOT is an open source project on GitHub, and the underlying Python code can be found at `https://github.com/rhiever/tpot`.

2.1. Machine Learning Pipeline Operators

At its core, TPOT is a wrapper for the Python machine learning package, scikit-learn (Pedregosa et al., 2011). 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 (Pedregosa et al., 2011) and Hastie et al. (2009) for detailed explanations of the machine learning algorithms used in TPOT.

Supervised Classification Operators. DecisionTree, RandomForest, eXtreme Gradient Boosting Classifier (from XGBoost, Chen and Guestrin (2016)), LogisticRegression, and KNearestNeighborClassifier. Classification operators store the classifier's predictions as a new feature as well as the classification for the pipeline.

Feature Preprocessing Operators. StandardScaler, RobustScaler, MinMaxScaler, MaxAbsScaler, RandomizedPCA (Martinsson et al., 2011), Binarizer, and PolynomialFeatures. Preprocessing operators modify the data set in some way and return the modified data set.

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

We also include an operator that combines disparate data sets, as demonstrated in Figure 1, which allows multiple modified copies of the data set to be combined into a single data set. 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.

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 1 shows an example tree-based pipeline, where two copies of the data set are provided to the pipeline, modified in a successive manner by each operator, combined into a single data set, and finally used to make classifications. Because all operators receive a data set as input and return the modified data set as output, it is possible to construct arbitrarily shaped machine learning pipelines that can act on multiple copies of the data set. Thus, GP trees provide an inherently flexible representation of machine learning pipelines.

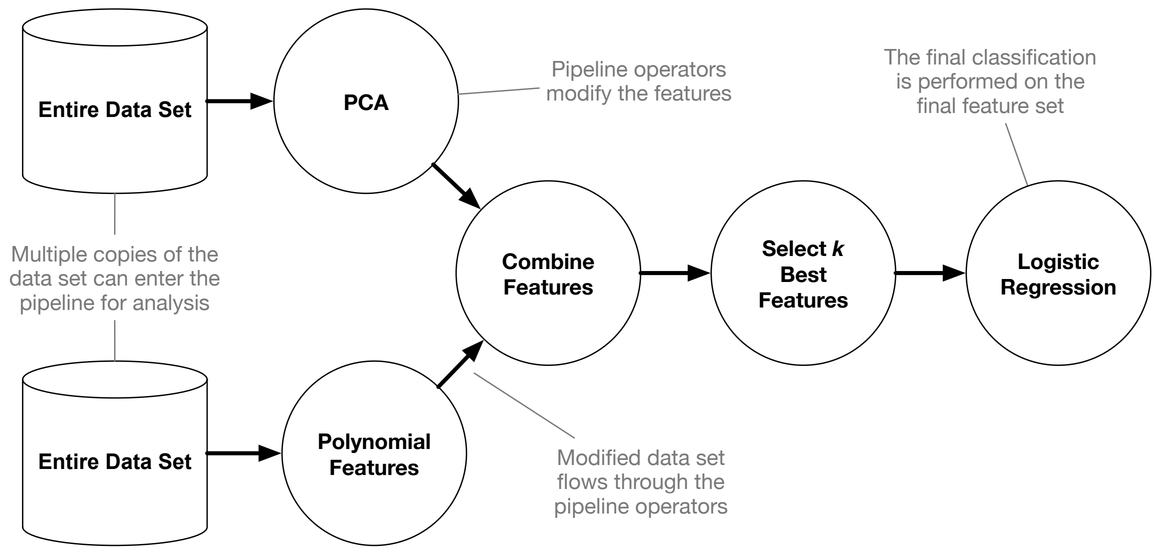
Entire Data Set
PCA
Pipeline operators modify the features
The final classification is performed on the final feature set
Multiple copies of the data set can enter the pipeline for analysis
Combine Features
Select *k* Best Features
Logistic Regression
Entire Data Set
Polynomial Features
Modified data set flows through the pipeline operators

Figure 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.

In order for these tree-based pipelines to operate, we store three additional variables for each record in the data set. 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 classifications from the last 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 data set provided to TPOT as training data is further split into an internal stratified 75%/25% training/testing set.

2.3. Optimizing Tree-based Pipelines

To automatically generate and optimize these tree-based pipelines, we use a GP algorithm (Banzhaf et al., 1998) as implemented in the Python package DEAP (Fortin et al., 2012). 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 data set. For every generation of the GP algorithm, the algorithm selects the top 20 pipelines in the population according to the NSGA-II selection scheme (Deb et al., 2002), where pipelines are selected to simultaneously maximize classification accuracy on the data set 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 (Deb et al., 2002) 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.

2.4. Benchmark Data

We compiled 150 supervised classification benchmarks[1] from a wide variety of sources, including the UCI machine learning repository (Lichman, 2013), a large preexisting benchmark repository from Reif (2012), and simulated genetic analysis data sets from Urbanowicz et al. (2012). These benchmark data sets 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 data sets from a wide range of application domains, including genetic analysis, image classification, time series analysis, and many more. Thus, this benchmark represents a comprehensive suite of tests with which to evaluate automated machine learning systems.

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 data set into a stratified 75%/25% training/testing split and used a distinct random number generator seed for each split and subsequent TPOT run.

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. In all cases, we measured accuracy of the resulting pipelines or models as balanced accuracy (Velez et al., 2007), which corrects for class frequency imbalances in data sets by computing the accuracy on a per-class basis then averaging the per-class accuracies. In the remainder of this paper, we refer to "balanced accuracy" as simply "accuracy."

Overall, TPOT discovered pipelines that perform statistically significantly better than a Random Forest with 500 trees on 22 benchmarks, significantly worse on 5 benchmarks, and had no significant difference on 123 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 2, we show the distributions of accuracies on the 27 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 5 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 fea-

1. Benchmark data available at `http://www.randalolson.com/data/benchmarks/`
2. Full list: `https://gist.github.com/rhiever/578cc9c686ffd873f46bca29406dde1d`

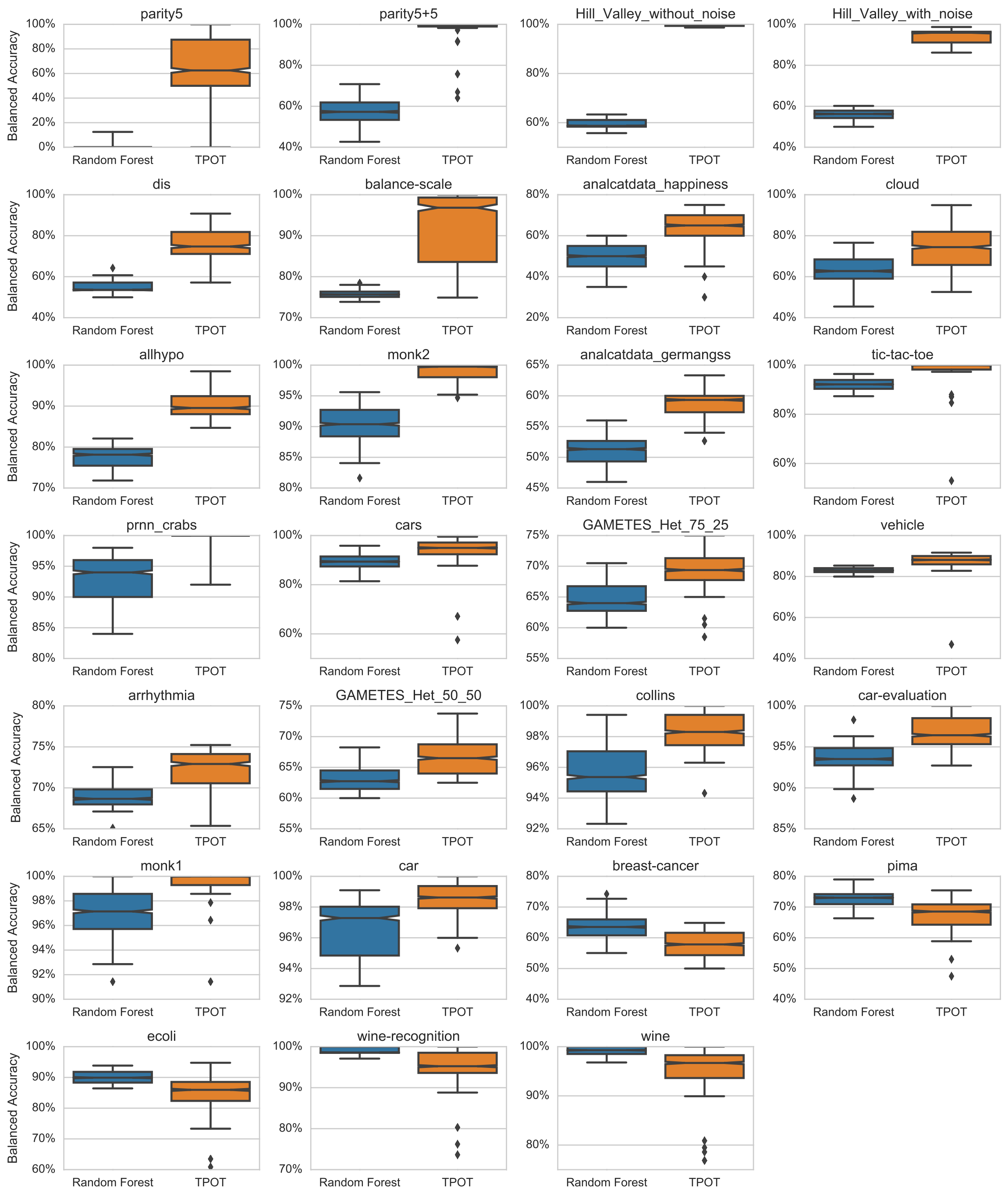

Figure 2: Box plots showing the distribution of balanced accuracies for the 27 benchmarks with a significant difference in median accuracy between TPOT and a Random Forest with 500 trees. Each box plot represents 30 replicates, the inner line shows the median, the notches represent the bootstrapped 95% confidence interval of the median, the ends of the box represent the first and third quartiles, and the dots represent outliers.

ture preprocessor prior to modeling the "Hill_valley" benchmarks allows Random Forests to classify the data set 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.

4. Conclusions and Future Work

We benchmarked the Tree-based Pipeline Optimization Tool (TPOT) on 150 supervised classification data sets and found that it discovers effective 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 (Greene et al., 2007) 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 (Feurer et al., 2015b). In brief, meta-learning harnesses information from previous machine learning runs to predict how well each pipeline configuration will work on a particular data set. To place data sets on a standard scale, meta-learning algorithms compute meta-features from the data sets, such as data set size, the number of features, and various aspects about the features, which are then used to map data set meta-features to corresponding pipeline configurations that may work well on data sets 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, Feurer et al. (2015a) imposed a short and fixed pipeline structure of a data preprocessor, a feature preprocessor, and a model. In another GP-based AutoML system, Zutty et al. (2015) 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. Bhowan et al. (2013) 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.

References

W Banzhaf, P Nordin, R E Keller, and F D Francone. *Genetic Programming: An Introduction*. Morgan Kaufmann, San Meateo, CA, USA, 1998.

Urvesh Bhowan, Mark Johnston, Mengjie Zhang, and Xin Yao. Evolving diverse ensembles using genetic programming for classification with unbalanced data. *Trans. Evol. Comp*, 17(3):368–386, June 2013. ISSN 1089-778X.

Tianqi Chen and Carlos Guestrin. XGBoost: A Scalable Tree Boosting System. *CoRR*, abs/1603.02754, 2016. URL http://arxiv.org/abs/1603.02754.

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

Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In C. Cortes, N.D. Lawrence, D.D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2944–2952. Curran Associates, Inc., 2015a.

Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. Initializing bayesian hyperparameter optimization via meta-learning. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pages 1128–1135, 2015b.

Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. DEAP: Evolutionary Algorithms Made Easy. *Journal of Machine Learning Research*, 13:2171–2175, 2012.

Casey S Greene, Bill C White, and Jason H Moore. An expert knowledge-guided mutation operator for genome-wide genetic analysis using genetic programming. In *Pattern Recognition in Bioinformatics*, pages 30–40. Springer Berlin Heidelberg, 2007.

Trevor J. Hastie, Robert John Tibshirani, and Jerome H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, New York, NY, USA, 2009. ISBN 978-0-387-84857-0.

Frank Hutter, Jörg Lücke, and Lars Schmidt-Thieme. Beyond Manual Tuning of Hyperparameters. *KI - Künstliche Intelligenz*, 29:329–337, 2015.

M Lichman. UCI machine learning repository, 2013. URL http://archive.ics.uci.edu/ml.

Per Gunnar Martinsson, Vladimir Rokhlin, and Mark Tygert. A randomized algorithm for the decomposition of matrices. *Applied and Computational Harmonic Analysis*, 30:47–68, 2011.

Randal S. Olson, Nathan Bartley, Ryan J. Urbanowicz, and Jason H. Moore. Evaluation of a tree-based pipeline optimization tool for automating data science, 2016a. URL http://arxiv.org/abs/1603.06212.

Randal S. Olson, Ryan J. Urbanowicz, Peter C. Andrews, Nicole A. Lavender, La Creis Kidd, and Jason H. Moore. *Applications of Evolutionary Computation: 19th European Conference, EvoApplications 2016, Porto, Portugal, March 30 April 1, 2016, Proceedings, Part I*, chapter Automating Biomedical Data Science Through Tree-Based Pipeline Optimization, pages 123–137. Springer International Publishing, 2016b.

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*, 12:2825–2830, 2011.

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

Phil Simon. *Too big to ignore: the business case for big data.* Wiley & SAS Business Series. Wiley, New Delhi, 2013.

Ryan J Urbanowicz, Jeff Kiralis, Nicholas A Sinnott-Armstrong, Tamra Heberling, Jonathan M Fisher, and Jason H Moore. GAMETES: a fast, direct algorithm for generating pure, strict, epistatic models with random architectures. *BioData Mining*, 5, 2012.

Digna R Velez, Bill C White, Alison A Motsinger, William S Bush, Marylyn D Ritchie, Scott M Williams, and Jason H Moore. A balanced accuracy function for epistasis modeling in imbalanced datasets using multifactor dimensionality reduction. *Genetic Epidemiology*, 31(4):306–315, 2007.

Jason Zutty, Daniel Long, Heyward Adams, Gisele Bennett, and Christina Baxter. Multiple objective vector-based genetic programming using human-derived primitives. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO '15, pages 1127–1134, New York, NY, USA, 2015. ACM.