Smart Model Elimination Machine Learning for Disease Prediction

Eric Su Zhang[1], **Benjamin Joseph Michael Standefer**[2] **& Stewart Mayer**[3]

Received November 26, 2024
Accepted July 03, 2025
Electronic access July 31, 2025

Automated Machine Learning (AutoML) has emerged as a popular field of research. We present a literature review of existing AutoML papers and conduct a survey on five popular AutoML frameworks. Typically, these frameworks engage in model selection by requiring every model to be run and assessed, a process both time-intensive and computationally expensive. In response, we propose a novel framework, Smart Model Elimination Machine Learning (SMEML), that strategically eliminates models that are unlikely to yield high accuracy. We trained a multi-label XGBoost regression model based on 268 datasets that had their attributes extracted. Based on the attributes of each dataset, our framework can rank the models it believes will be the most performative. Based on this ranking, we can eliminate the lower ranked models. SMEML demonstrates the ability to achieve comparable accuracy to a traditional brute-force approach while significantly reducing the time required. Compared to the brute-force approach, SMEML is 135.2% faster on average. We also believe this innovation is particularly beneficial to machine learning in healthcare, where efficient and accurate disease prediction is crucial.

Keywords: Machine Learning (ML), Automated Machine Learning, Healthcare, Computation Efficiency

Introduction

Machine learning (ML) algorithms require complicated processes like hyperparameter tuning and can be modified in conjunction with one another, such as in ensemble learning. AutoML frameworks automate the complicated processes of producing effective ML models, making it more accessible to non-experts. These frameworks create comfortable environments in which models can excel. They handle missing values, preprocess data, and optimize training to ensure the best results[1]. One sector of the global industry that has the most potential for beneficial integration of AutoML frameworks is healthcare, particularly in developing countries. These countries have severe shortages of healthcare workers and limited tools for diagnosis. For example, Africa has 2.3 healthcare workers per 1000 individuals, while the Americas have 24.8[2]. The World Health Organization (WHO) emphasized that this deficit is growing every year and will likely reach 18 million personnel by 2030[3]. Medical AIs typically automate repetitive tasks, making time consumption a primary concern[4]. This coupled with a lack of adequate resources makes designing faster diagnostic systems for developing countries' medical sectors a pivotal issue. Most AutoML frameworks implement some form of model selection where a pool of models is filtered until a final model is selected. However, these frameworks require that every model be run[1]. In theory, this means that significant energy is spent training models that are unlikely to be selected.

We systematically review and compare five different AutoML frameworks, evaluating every feature relevant to medical implementation in Table 1[5–10].

A Fast Library for Automated Machine Learning & Tuning (FLAML) uses blend search hyper-parameter optimization and focuses on lightweight models, making it quick and applicable to repetitive tasks. It supports user-specified input techniques for missing values and has an intuitive API, making it moderately user-friendly for healthcare workers[9].

H2O AutoML has a wider variety of supported algorithms that may be appropriate for medical diagnosis, making it slower than FLAML on average. In one trial, more than half of FLAML's performances in one minute were better than or equal to H2O's performances in one hour[9]. H2O has automatic and multi-faceted methods for dealing with missing values and flexible interfaces in R and Python, making it intuitive for users[5].

MLJAR uses advanced algorithms such as light gradient boosting and neural networks. It automatically deals with missing values and is known for its automated user interface[7].

Tree-based Pipeline Optimization Tool (TPOT) uses genetic programming. It builds pipelines over multiple generations, which can be time consuming. It has built-in pre-processing for missing values, but the genetic programming requires fine-tuning from users[7].

LightAutoML is an AutoML framework designed for large financial systems. It uses a "Reader" object to perform feature selection based on input data. It may generate additional features such as interaction terms (eg. product of two features), measure the number of times a feature was used in a decision

[1,2] *St. Mark's School of Texas, Dallas, Texas*
[3] *Department of Science, St. Mark's School of Texas, Dallas, Texas*

Table 1 Survey of Five Existing AutoML Frameworks. Abbreviations: EOU: Ease of Use, M: Number of models evaluated by framework, FE: Feature Engineering, TM: Time Management, HPT: Hyper-Parameter Tuning, Inter: Interface, Custom: Customization

Framework	EOU	M	FE	TM	HPT	Inter	Custom
H2O AutoML[5]	Easy	12	Basic	Yes	Auto	SHAP	High
TPOT[6]	Hard	10	Basic	No	Genetic	POJO	Moderate
MLJAR[7,8]	Easy	12	Advanced	Yes	Auto	Basic	High
FLAML[9]	Easy	9	Limited	Yes	Auto	SHAP	Low-Moderate
LightAutoML[10]	Easy	11	Advanced	Yes	Auto	Basic	Moderate

tree split to determine importance, and randomly shuffle the values of a single feature to observe its effect on the model's performance. It then uses ensemble learning and finds the best performing combination. The model pool includes Gradient Boosting Decision Trees (LightGBM, CatBoost), Linear Models (LinearLBFGS, LinearL1CD), Neural Networks (MLP). For the ensemble learning, LightAutoML employs the Stacking, Bagging, and Boosting methods[10].

Overall, we see several limitations with the current field of AutoML frameworks. These tools have a "black-box" tendency which makes it difficult to understand why certain predictions are made. While the complex and diverse number of algorithms may produce excellent results, the reasons certain models are trained in ensemble are often just results of randomized choosing or optimization algorithms without a hard criterion. Additionally, the usage of randomization in the initialization of many of these models and the selection process of these models makes results difficult to reproduce between runs and between researchers.

Through this survey, we uncover a key continuity in the above frameworks' design: these frameworks must run all of their models to maximize accuracy at the expense of efficiency. Given the specific nature of certain tabular data, there are some models that are less likely to perform well, thus running and training these models is wasteful. Cutting these models completely, however, is not optimal, as they are the best option in a minority of cases. When considering a specific dataset, the distribution and relationship between data points varies from dataset to dataset. We believe these differences can be crucial clues for giving insight as to which models are best for which datasets. Furthermore, we believe that specific models are more tailored for specific dataset characteristics. For example, linear models tend to work better on linear data[11]. We hypothesize that using a combination of multi-layer machine learning implementation and meta-feature extraction would yield an equally versatile system that minimizes time spent on training without sacrificing accuracy within a medical context. We've termed this approach Smart Model Elimination Machine Learning (SMEML).

Experiment Implementation

To evaluate our framework, we choose six different datasets of diverse size:

The dataset "Lung Cancer" uploaded by user "Ms. Nancy Al Aswad" is a lung cancer prediction dataset with 15 features The uploader of this dataset states that it is to "practice data analysis" with her students. The data distribution is 87% positive and 13% negative and has 309 rows, making it a relatively small dataset. The dataset has no missing values[12].

The "Chronic Kidney Disease Dataset" from user "Mansoor Iqbal" is originally sourced from the UCI Machine Learning Repository. The data is taken from a 2 month period in India. It has 25 features and 400 rows, making it a relatively small dataset. Its distribution is 62% positive and 38% negative[13].

The "Pima Indians Diabetes Database" uploaded by "UCI Machine Learning" is from the National Institute of Diabetes and Digestive and Kidney Disease. All the patients were women at least 21 years old of Pima Indian Heritage. The dataset has 8 features and 768 rows with a distribution of 35% positive and 65% negative, making this a relatively small dataset[14].

The dataset "Brain stroke prediction dataset" uploaded by user "Izzet Turkalp Akbasli" contains 10 features and 4892 rows, making it a medium sized dataset. The distribution is 5% positive and 95% negative, making it extremely unbalanced[15].

The "stroke prediction dataset" uploaded by user "Fedesoriano" has 10 features and 5110 rows, making it a medium sized dataset. The distribution is 5% positive and 95% negative, making it extremely unbalanced[16].

The dataset "Eye State Classification EEG Dataset" uploaded by user "Rob Mulla" is sourced from the UCI Machine Learning Repository. It contains 14 features and has 14980 rows, making it a relatively large dataset[17].

To conduct this experiment, we used accuracy and time to train as our metrics. We split the dataset into a 80% train set and 20% test set split. We feed the test set into SMEML and evaluate the final model on the test set. We compare this against a "dumb" mode which simply runs every model on the dataset, using the same train and test set. All experiments are run on an Ubuntu server with Intel Xeon E5520 @ 2.27GHz and 24GB of RAM. We run SMEML and the brute-force on each dataset for

20 iterations of hyperparameter tuning [12–17].

Table 2 Specifications and Parameters of the Study

Parameter	Specification
Train Split	80%
Test Split	20%
OS	Ubuntu Server
CPU	Intel Xeon E5520
Memory	24GB
Iterations of Hyperparameter Tuning	20 iterations

Experiment Results

A table of the results can be found in Table 3, and a graphical visualization can be found in Figure 1. The first observation made is that accuracy does not change for 4 of the 6 benchmarks. The other two benchmarks on eye states and diabetes prediction represent a loss of accuracy for SMEML by 0.13% and 1.95%, respectively. Training time, however, favors SMEML in every benchmark. SMEML outperforms the dummy framework by 173.96 seconds in its worst performance on the lung cancer benchmark and by 383.94 seconds in its best performance on the eye state benchmark. Relative to the dummy's performance, a percent decrease in time to run for SMEML of at least 46.78% could be seen, with a decrease of up to 69.22% observed as well. SMEML's worst performance by percent decrease in runtime is also the performance in which it returned perfect accuracy.

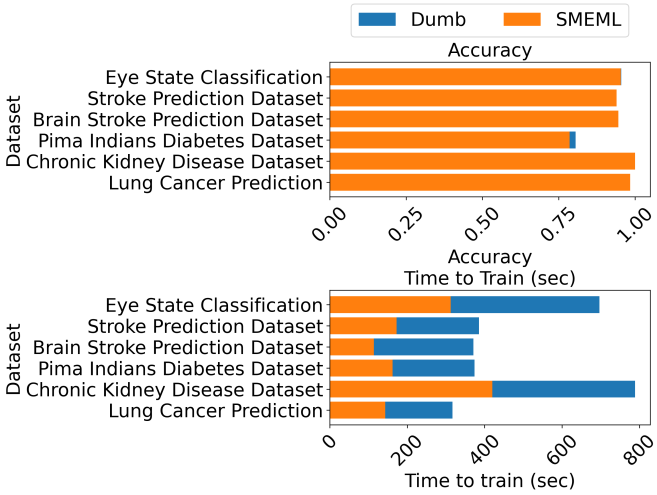

Fig. 1 Graphical Visualization of the Experiment Results for SMEML and Dumb Mode.

Discussion

SMEML consistently achieves comparable accuracy to its dumb counterpart while significantly reducing the time required to train the model. Notably, SMEML's performance is not limited by the size of the dataset, achieving its best runtimes–first, third, and fourth fastest overall–on 'large' datasets (those exceeding 1000 rows). SMEML also chooses less common algorithms, such as selecting SGDClassifier for the Pima Indians Diabetes Dataset rather than defaulting to industry standard boosting algorithms because SGDClassifier produces better accuracy [18]. This showcases the advantages of SMEML compared to a simple selection of the most popular models that may work well for a majority of datasets, but not for a minority of cases. This efficiency and versatility are important in third world medicine where there is a constant influx of new patient data that must be processed quickly. It is also worth noting that SMEML prioritizes extraction of mean Q3 and STD of Percent of Z-score outliers during the training process, with those two values being the first and second highest weighted attributes respectively (Table 5). Q3 is a measure of upper-quartile data distribution, and STD of Percent of Z-score outliers quantifies the distribution of outliers. The prioritization of these metrics potentially speaks to the medical applicability of this approach, given that medical data is often subject to skewed distributions favoring rare outliers [19]. Favoring Q3 means the model is potentially more aware of these outliers, and favoring STD of Percent of Z-score outliers means the model can potentially better compensate for how these outliers skew the distribution. A practical demonstration of this software's real-world application can be found on our SmartDx website [20]. Source code can be found at [21].

Conclusion

In this paper, we proposed a novel AutoML framework called SMEML that automatically eliminates models from a pool that are unlikely to be performant based on the data features of input data. We trained a multi-label regression that predicts the ranking of a model's performance based on various attributes from the data such as data size, distribution, and correlation between columns. Our experimentation results demonstrate that SMEML can effectively eliminate the models that are unlikely to perform well and thus reduce the time necessary to train models.

Limitations

Our implementation of SMEML has limitations. These include a small training set due to lack of available datasets, finite computing power, time constraints, and a lack of access to particular scientific literature. Our framework only improves on the model selection process and does not address the various other aspects of AutoML such as hyperparameter tuning and feature selection

Table 3 Comparing Experiment Results for SMEML and Dumb Mode. The performance results, including metrics of accuracy and time, of SMEML compared to its brute-force dumb counterpart.

Dataset	Accuracy (SMEML)	Accuracy (Dumb)	Time to Train (sec) (SMEML)	Time to Train (sec) (Dumb)	Size (R×C)
Lung Cancer Prediction [12]	0.9839	0.9839	143.07	317.04	309×16
Chronic Kidney Disease Dataset [13]	1.0000	1.0000	419.64	788.46	400×26
Pima Indians Diabetes Dataset [14]	0.7857	0.8052	161.99	374.01	768×8
Brain Stroke Prediction Dataset [15]	0.9458	0.9458	114.16	370.95	4892×11
Stroke Prediction Dataset [16]	0.9393	0.9393	172.67	385.23	5110×12
Eye State Classification [17]	0.9536	0.9549	312.57	696.51	14980×15

that other frameworks in the literature review improve upon. Therefore, SMEML is not comparable to other full-featured AutoML frameworks, and a direct comparison would not be informative. Additionally, we cannot completely guarantee the independence of datasets. While we ensure that the same dataset is not downloaded twice from Kaggle, we cannot ensure that users do not reupload or upload the same or similar data from multiple sources.

Future Work

Future work will include increasing the size of the training set, implementing a fully-fledged AutoML framework with improvement in the other stages of Auto-Machine Learning, and conducting a detailed experiment comparing SMEML to other popular AutoML frameworks.

Methodology

Design Overview

A general summary of the SMEML Framework process is shown in Figure 2. SMEML specifically targets binary classification of tabular data and is designed to be a component in a larger fully-fledged framework. SMEML consists of four major stages:

- *Attribute Extraction:* extracts all attributes of a dataset specified in Table 5
- *SME Model Ranking:* predicts a ranking based on model performance
- *Hyperparameter Tuning:* optimization of model hyperparameters
- *Final Model Selection:* selection and creation of a final model using both performance metrics and ensemble learning

We also create a variant version of SMEML that is a "dumb" mode. It eliminates the Attribute Extraction and SME Model Ranking stages of the framework and is an example of a brute force run of all the models. The "dumb" mode process is outlined in Figure 3. We use this for our evaluation against SMEML.

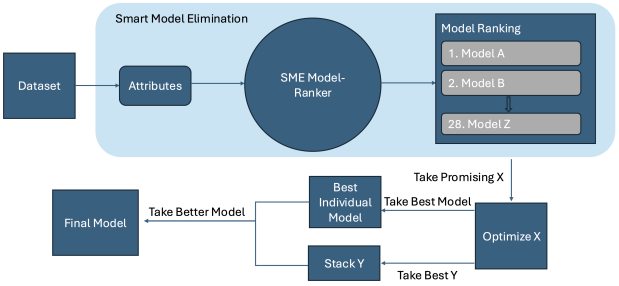

Fig. 2 Overview of the SMEML Process

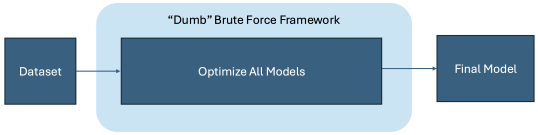

Fig. 3 Overview of the "Dumb" Brute Force Process

Model Pool

SMEML generates its models from a pool of 28 different models. The goal of this large pool size and model diversity is to have a model that is designed for every type of dataset. The model pool includes all supervised and semi-supervised Sklearn classification models and common boosting models such as XGBoost, LightGBM, and Catboost [22–25]. The majority of these models are implemented in the frameworks mentioned in Table 1. A complete list of all of these models can be found in Table 4 with their respective hyperparameters.

Table 4 List of Implemented Models. List of all models in the pool of 28 used in SMEML including specification of library and hyperparameters.

Model	Library	Hyperparameter Options
SVC	sklearn	C, gamma
SGDClassifier	sklearn	loss, penalty, max_iter
RidgeClassifier	sklearn	alpha, solver
Perceptron	sklearn	max_iter
PassiveAggressiveClassifier	sklearn	C, max_iter, tol
LogisticRegression	sklearn	C, penalty, solver, max_iter
LinearSVC	sklearn	C, penalty, loss, max_iter
RandomForestClassifier	sklearn	n_estimators, max_features, max_depth, min_samples_split, min_samples_leaf, bootstrap
HistGradientBoostingClassifier	sklearn	learning_rate, max_iter, max_leaf_nodes, max_depth, min_samples_leaf
GradientBoostingClassifier	sklearn	n_estimators, learning_rate, max_depth, min_samples_split, min_samples_leaf, subsample
ExtraTreesClassifier	sklearn	n_estimators, max_features, max_depth, min_samples_split, min_samples_leaf, subsample
AdaBoostClassifier	sklearn	n_estimators, learning_rate
XGBClassifier	XGBoost	n_estimators, learning_rate, max_depth, subsample, colsample_bytree
LGBMClassifier	LightGBM	n_estimators, learning_rate, num_leaves
CatBoostClassifier	Catboost	iterations, learning_rate, depth
RadiusNeighborsClassifier	sklearn	radius, weights, algorithm
KNeighborsClassifier	sklearn	n_neighbors, weights, algorithm
NearestCentroid	sklearn	metric
QuadraticDiscriminantAnalysis	sklearn	reg_param
LinearDiscriminantAnalysis	sklearn	solver, shrinkage
GaussianNB	sklearn	var_smoothing
BernoulliNB	sklearn	alpha, binarize
MLPClassifier	sklearn	hidden_layer_sizes, activation, solver, alpha, learning_rate, max_iter
ExtraTreeClassifier	sklearn	criterion, max_depth, min_samples_split, min_samples_leaf
DecisionTreeClassifier	sklearn	criterion, splitter, max_depth, min_samples_split, min_samples_leaf
LabelSpreading	sklearn	gamma, n_neighbors
LabelPropagation	sklearn	gamma, n_neighbors
DummyClassifier	sklearn	strategy

Attribute Extraction

To quantify the internal data distribution and relationships, we selected various metrics that describe a specific dataset attribute. In total, we selected 16 different attributes. These extracted attributes are shown in Table 5 with their formula, purpose, and weight determined by the SME Model Ranker which will be discussed in the next section. The following purposes specified in Table 5 are:

- *Data Size:* The size of the dataset quantified through row and column size
- *Data Type:* The type of data that is in the dataset, whether it is numerical or categorical
- *Data Distribution:* The distribution of data among numerical columns
- *Outlier Data Points:* How much outlier data there is?
- *Data Linearity:* The relationship of data between columns and their correlations

All of these attributes are extracted from the given dataset and used in the SME Model Elimination Ranker which is discussed in the next section.

SME Model Elimination Ranker

To implement the model elimination process, we train a multilabel-regression boosting model called the SME Model-Ranker on 268 Kaggle datasets. Specifically, we used an XGBoost Multi-Label Regression as the model. We choose this

Table 5 Attributes Extracted From a Dataset. Abbreviations: DS: Data Size, DT: Data Type, DD: Data Distribution, ODP: Outlier Data Points, DL: Data Linearity, Weight: Variable Importance.

Attribute	Formula	Purpose	Weight
# of Rows	$\|A\|_r$	DS	0.0422
# of Columns	$\|A\|_c$	DS	0.0415
Target Distribution	$\max(\text{count}_1, \text{count}_0)/\|A\|_r$	DT	0.0489
Numerical Columns	$\|A_{\text{num}}\|/\|A\|_c$	DT	0.0117
Binary Categorical Columns	$\|A_{\text{bin}}\|/\|A\|_c$	DT	0.0117
Mean # of Distinct Values	$\mu\{\#\text{ distinct values in column}\}$	DT	0.0076
Mean IQR	$\mu\{\text{IQR of column}\}$	DD	0.0594
STD of IQR	$\sigma\{\text{IQR of column}\}$	DD	0.0700
Mean Q1	$\mu\{\text{Q1 of column}\}$	DD	0.0789
STD of Q1	$\sigma\{\text{Q1 of column}\}$	DD	0.0613
Mean Q3	$\mu\{\text{Q3 of column}\}$	DD	0.0812
STD of Q3	$\sigma\{\text{Q3 of column}\}$	DD	0.1103
Mean Percent of Z-Score Outliers	$\mu\{\#\text{ of Z-Score Outliers}\}$	ODP	0.0558
STD of Percent of Z-Score Outliers	$\sigma\{\#\text{ of Z-Score Outliers}\}$	ODP	0.0798
Mean Correlation	$\mu\{\text{Correlation Matrix}\}$	DL	0.0559
STD of Correlation	$\sigma\{\text{Correlation Matrix}\}$	DL	0.1526

model after also testing Catboost and LightGBM's regression models. Their spearman coefficient results are in Table 6. We choose medical datasets and focus our approach on binary classification. We extract the attributes in Table 5 from each dataset, which are the inputs for the boosting model. Then a "dumb", brute-force framework that trains every model on every dataset is run. The accuracy of every model is ranked and transformed into its ranking index. For example, given three models a, b, c with accuracies 90%, 60%, and 70% respectively, their accuracies would be transformed into ranking indexes of 1, 3, 2. Every model is treated as a label with its value being its rank index. We train the SME to predict the rank of each model, given the extracted attributes. SME produces a median Spearman correlation of 0.7301 between the predicted rank and the actual rank and a median p-value of 1.032e-05. Furthermore, we guarantee that we can predict a top two model within the predicted top eight 90.12% of the time. Therefore, after the rank index of all models is predicted, we sort the models based off their rank index. We take the top X models, in this case 8, based off the ranking and optimize their hyperparameters which will be explained in the next section.

Table 6 Spearman Coefficient of Regression Models

Model	Spearman Coefficient
XGBoost	0.7301
CatBoost	0.3313
LightGBM	0.2934

Hyper-parameter Optimization

For hyperparameter optimization, we run a Bayesian optimization which is one of the most common optimization algorithms in the field. We run Bayesian optimization using the scikit-optimize framework on the top eight models to get the best hyperparameters for each model[26]. All hyperparameters are listed in Table 4.

Final Model Selection

To select the final model, we compared the best performing model against a stacked model with the top Y models. This Y value can be adjusted by the user. In our experiments, we choose Y to be 3. The final model is the better performing model out of these two based on accuracy.

Evaluation Metrics

To evaluate the models, we picked two metrics: accuracy and time. We use accuracy to evaluate the framework because it is the metric we optimized for during hyper-parameter optimization. Its formula can be found below. To measure the speed and efficiency of the framework, we measure the time it takes to run the models in seconds, from when the framework is first initialized to when the final model is generated, including all stages between.

Formula for accuracy

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN} \quad (1)$$

Acknowledgments

We would like to thank St. Mark's School of Texas for supporting the associated NASA Hunch project where this research was initially motivated.

References

1 X. He, *AutoML: A Survey of the State-of-the-Art*.

2 G. Nchasi, *Challenges Faced by African Healthcare Workers During the Third Wave of the Pandemic*.

3 W. H. Organization, *Global Strategy on Human Resources for Health: Workforce 2030*.

4 J. Bajwa, *Artificial Intelligence in Healthcare: Transforming the Practice of Medicine*.

5 E. LeDell and S. Poirier, *H2O AutoML: Scalable Automatic Machine Learning*.

6 S. Olson, *Automating Biomedical Data Science Through Tree-Based Pipeline Optimization*.

7 A. Płonska, *MLJAR: State-of-the-Art Automated Machine Learning Framework for Tabular Data, Version 0.10.3*.

8 F. Conrad, *Benchmarking AutoML for Regression Tasks on Small Tabular Data in Materials Design*.

9 Z. Shang, *FLAML: A Fast and Lightweight AutoML Library*.

10 A. Ryzhkov, *LightAutoML: AutoML Solution for a Large Financial Services Ecosystem*, arXiv:2109.01528,.

11 A. Akalin, *Computational Genomics with R*.

12 N. Al Aswad, *Lung Cancer*.

13 M. Iqbal, *Chronic Kidney Disease Dataset*.

14 U. Learning, *Pima Indians Diabetes Dataset*.

15 I. Akbasli, *Brain Stroke Prediction Dataset*.

16 Fedesoriano, *Stroke Prediction Dataset*.

17 R. Mulla, *Eye State Classification EEG Dataset*.

18 A. Mayr, *The Evolution of Boosting Algorithms From Machine Learning to Statistical Modelling*.

19 W. P. Bensken, *Basic Introduction to Statistics in Medicine, Part 1: Describing Data*.

20 *SmartDx Website*, https://smartdx.vercel.app/.

21 *Github Repository*, https://github.com/ericspring08/SMEML-Paper.

22 F. Pedregosa, *Scikit-learn: Machine Learning in Python*.

23 T. Chen, *XGBoost: A Scalable Tree Boosting System*.

24 G. Ke, *LightGBM: A Highly Efficient Gradient Boosting Decision Tree*.

25 L. Prokhorenkova, *CatBoost: Unbiased Boosting with Categorical Features*.

26 T. Head, *Scikit-Optimize: Sequential Model-Based Optimization*.