GigaScience, 9, 2020, 1–14

doi: 10.1093/gigascience/giaa026
Technical Note

TECHNICAL NOTE

ShinyLearner: A containerized benchmarking tool for machine-learning classification of tabular data

Stephen R. Piccolo[*], Terry J. Lee, Erica Suh and Kimball Hill

Department of Biology, Brigham Young University, 4102 Life Sciences Building, Provo, UT, 84602, USA

[*]**Correspondence address.** Stephen R. Piccolo, Department of Biology, Brigham Young University, 4102 Life Sciences Building, Provo, UT, 84602, USA. E-mail: stephen_piccolo@byu.edu http://orcid.org/0000-0003-2001-5640

Abstract

Background: Classification algorithms assign observations to groups based on patterns in data. The machine-learning community have developed myriad classification algorithms, which are used in diverse life science research domains. Algorithm choice can affect classification accuracy dramatically, so it is crucial that researchers optimize the choice of which algorithm(s) to apply in a given research domain on the basis of empirical evidence. In benchmark studies, multiple algorithms are applied to multiple datasets, and the researcher examines overall trends. In addition, the researcher may evaluate multiple hyperparameter combinations for each algorithm and use feature selection to reduce data dimensionality. Although software implementations of classification algorithms are widely available, robust benchmark comparisons are difficult to perform when researchers wish to compare algorithms that span multiple software packages. Programming interfaces, data formats, and evaluation procedures differ across software packages; and dependency conflicts may arise during installation. **Findings:** To address these challenges, we created ShinyLearner, an open-source project for integrating machine-learning packages into software containers. ShinyLearner provides a uniform interface for performing classification, irrespective of the library that implements each algorithm, thus facilitating benchmark comparisons. In addition, ShinyLearner enables researchers to optimize hyperparameters and select features via nested cross-validation; it tracks all nested operations and generates output files that make these steps transparent. ShinyLearner includes a Web interface to help users more easily construct the commands necessary to perform benchmark comparisons. ShinyLearner is freely available at https://github.com/srp33/ShinyLearner. **Conclusions:** This software is a resource to researchers who wish to benchmark multiple classification or feature-selection algorithms on a given dataset. We hope it will serve as example of combining the benefits of software containerization with a user-friendly approach.

Keywords: machine learning; supervised learning; classification; software containers; benchmark; feature selection; algorithm optimization; model selection

Background

Classification falls under the category of supervised learning, a branch of machine learning. When performing classification, researchers seek to assign observations to distinct groups. For example, medical researchers use classification algorithms to identify patterns that predict whether patients have a particular disease, will respond positively to a particular treatment, or will survive a relatively long period after diagnosis [1–11]. Applications in molecular biology include annotating DNA sequencing elements, identifying gene structures, and predicting protein secondary structures [12].

Typically, a classification algorithm is "trained" on a dataset that contains samples (observations) from 2 or more groups, and the algorithm identifies patterns that differ among the groups. If these patterns are reliable indicators of group membership, the algorithm will be able to accurately assign new samples to these groups and thus may be suitable for broader application. Different research applications require different levels of accuracy

Received: 18 June 2019; **Revised:** 5 December 2019; **Accepted:** 28 February 2020

© The Author(s) 2020. Published by Oxford University Press. This is an Open Access article distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted reuse, distribution, and reproduction in any medium, provided the original work is properly cited.

before classification algorithms are suitable for broader application. However, even small improvements in accuracy can provide large benefits. For example, if an algorithm predicts drug treatment responses for 1,000 patients and attains accuracy levels that are 2% higher than a baseline method, this algorithm would benefit 20 additional patients. Accordingly, a key focus of classification research in the life sciences is to identify generalizable ways to optimize prediction accuracy.

The machine-learning community have developed hundreds of classification algorithms and have incorporated many of these implementations into open-source software packages [13–18]. Each algorithm has different properties, which affect its suitability for particular applications. In addition, most algorithms require hyperparameters, which alter the algorithms' behavior and can affect the algorithms' accuracy dramatically. In addition, feature selection (or feature-ranking) algorithms can be used in complement to classification algorithms, helping to identify combinations of variables that are most predictive of group membership and aiding in data interpretation [19, 20]. With this abundance of options to consider, researchers face the challenge of identifying which algorithm(s), hyperparameter combinations, and features are optimal for a particular dataset.

To improve the odds of making successful predictions, researchers should choose algorithms, hyperparameters, and features based on empirical evidence rather than hearsay or anecdotal experience. Prior studies can provide insight into algorithm performance, but few studies evaluate algorithms comprehensively, and performance may vary widely for different types of data. One way to select these options empirically is via nested cross-validation [21]. With this approach, a researcher divides a single dataset into training and validation sets. Within each training set, the researcher divides the data further into training and validation subsets and then evaluates various options using these subsets. The top-performing option(s) are then used when making predictions on the outer validation set. Alternatively, a researcher might perform a benchmark study, applying (non-nested) cross-validation to multiple datasets from a given research domain. After testing multiple algorithms, hyperparameters, and/or feature subsets, the researcher can examine overall trends and identify options that tend to perform well [22, 23]. With either approach, it is ideal to evaluate a comprehensive set of options. However, several challenges make it difficult to perform such evaluations effectively:

- Researchers may wish to compare algorithms that have been implemented in different software packages. Although many machine-learning packages allow users to execute algorithms programmatically, APIs are not standardized, and they are implemented in diverse programming languages.
- Different software implementations use different techniques for evaluating algorithm performance, so it is difficult to ensure that comparisons are consistent.
- Input and output formats differ by software implementation, thus requiring custom efforts to prepare data and interpret results.
- When installing the software, researchers typically must install a series of software dependencies. Installation requirements often differ by operating system, and versioning conflicts can arise [24].

To reduce these barriers, we created ShinyLearner. For this open-source project, we have integrated existing machine-learning packages into containers, which provide a consistent interface for performing benchmark comparisons of classification algorithms. ShinyLearner can be installed on Linux, Mac, or Windows operating systems, with no need to install software dependencies other than the Docker containerization software. ShinyLearner currently supports 53 classification algorithms and ≥1,300 hyperparameter combinations across these algorithms; users can perform automatic hyperparameter tuning via nested cross-validation. In addition, ShinyLearner supports 16 feature selection algorithms, enabling researchers to reduce data dimensionality before performing classification (via nested cross-validation). New algorithms can be integrated in an extensible manner.

ShinyLearner is designed to be friendly to non-computational scientists—no programming is required. We provide a Web-based tool [25] to guide users through the process of creating the Docker commands necessary to execute the software. ShinyLearner supports a variety of input formats and produces output files in "tidy data" format [26], thus making it easy to import results into external tools. Even though other machine-learning packages support nested cross-validation, these evaluations may occur in a "black box." ShinyLearner tracks all nested operations and generates output files that make this process transparent.

Below we describe ShinyLearner in more detail and illustrate its use via benchmark evaluations. We evaluate 10 classification algorithms and 10 feature selection algorithms on 10 biomedical datasets. In addition, we assess the effects of hyperparameter optimization on predictive performance, provide insights on model interpretability, and consider practical elements of performing benchmark comparisons.

Methods

ShinyLearner (ShinyLearner, RRID:SCR_017608) encapsulates open-source, machine-learning packages into Docker images [27], which are available on Docker Hub [28]. Currently, ShinyLearner supports algorithms from scikit-learn, Weka, mlr, h2o, and Keras (with a TensorFlow backend) [13–15, 29–31]. To facilitate user interaction, to harmonize execution across the tools, and to evaluate predictive performance, ShinyLearner uses shell scripts, Python scripts, R scripts, and Java code [32–34]; these are included in the Docker images. To perform an analysis, the user executes a shell command, specifying arguments to indicate the location(s) of the input files, which algorithms to use, whether to perform Monte Carlo or *k*-fold cross-validation, etc. The analysis is executed within a container, and output files are saved to a directory that the user specifies. TensorFlow provides support for execution on graphical processing units, which requires a slightly different software configuration, so we provide a separate Docker image that enables this feature [35]. All changes to the ShinyLearner code are tested via continuous integration [36]; build status can be viewed at [37].

Fig. 1 shows an example ShinyLearner command that a user might execute. For convenience, and to help users who have limited experience with Docker or the command line, we created a Web-based user interface where users can specify local data paths, choose algorithms from a list, and select other settings [25]. After the user has made these selections, the Web interface generates a Docker command, which the user can copy and paste; Windows command line, Mac terminal, and Linux terminal commands are generated. We used the R Shiny (R Shiny, RRID:CR_001626) framework to build this web application [38], hence the name "ShinyLearner."

ShinyLearner interfaces with each third-party machine-learning package via shell scripts that wrap around the soft-

```
mkdir -p "/home/user/OutputData"

docker run --rm -i \
  -v "/home/user/InputData":"/InputData" \
  -v "/home/user/OutputData":"/OutputData" \
  --user $(id -u):$(id -g) \
  srp33/shinylearner:version511 \
  /UserScripts/nestedclassification_montecarlo \
    --data "data.tsv" \
    --description "My_Analysis_Description" \
    --outer-iterations 10 \
    --inner-iterations 5 \
    --classif-algo "/AlgorithmScripts/Classification/tsv/keras/dnn/*" \
    --classif-algo "/AlgorithmScripts/Classification/tsv/mlr/h2o.randomForest/*" \
    --classif-algo "/AlgorithmScripts/Classification/tsv/mlr/xgboost/*" \
    --classif-algo "/AlgorithmScripts/Classification/tsv/sklearn/svm/*" \
    --seed 1 \
    --ohe true \
    --scale true \
    --impute false
```

Figure 1: Example ShinyLearner command for performing a benchmark comparison. In this example, the user wishes to place output files in a directory located at/home/user/OutputData. To avoid problems with file permissions, this directory should be created before Docker is executed. The Docker run command builds a container and maps input and output directories from the host operating system to locations within the container (separated by colons). The –user directive indicates that the container should execute using the executing user's permissions. The name of the Docker image and tag name are specified (srp33/shinylearner: version511) as well as the name of a ShinyLearner script that performs nested, Monte Carlo cross-validation (/UserScripts/nestedclassification_montecarlo). The remaining arguments indicate the name of the input data file, a description of the analysis, the number of Monte Carlo iterations, the classification algorithms, etc. ShinyLearner provides documentation on each of these arguments as well as a Web application for building such commands dynamically.

ware's API. For each algorithm, 1 shell script specifies the algorithm's default hyperparameters. In most cases, additional shell scripts specify alternative hyperparameters. The classification algorithms in ShinyLearner span methodological categories, including linear models, kernel-based techniques, tree-based approaches, Bayesian models, distance-based methods, ensemble approaches, and neural networks. In selecting algorithms to include, we focused primarily on implementations that can handle discrete and continuous data values, support multiple classes, and produce probabilistic predictions. For each algorithm, we reviewed documentation for the third-party software and identified a representative variety of hyperparameter options. Admittedly, these selections are somewhat arbitrary and inexhaustive. However, they can be extended with additional options. We excluded some algorithm implementations and hyperparameter combinations because errors occurred when we attempted to execute them or because they failed to achieve reasonable levels of classification accuracy on simulated data.

Additional algorithms (and hyperparameter combinations) can be incorporated into ShinyLearner. The sole requirements are that they have been implemented as free and open-source software and provide an API (that can be executed via Linux command line scripts). Users who wish to extend ShinyLearner must:

1. Identify any software dependencies that the new algorithm requires. If those dependencies are not currently included in the ShinyLearner image, the user must modify the ShinyLearner Dockerfiles accordingly.
2. Create bash script(s) that accept specific arguments and invoke the new algorithm.
3. Request that these changes be included in ShinyLearner via a GitHub pull request.

ShinyLearner supports the following input data formats: tab-separated value (.tsv), comma-separated value (.csv), and attribute-relation file format (.arff). When tab-separated or comma-separated files are used, column names and row names must be specified; by default, rows must represent samples (observations) and columns must represent features (variables). However, transposed versions of these formats can be used (features as rows and samples as columns); in these cases, the user should use ".ttsv" or ".tcsv" as the file extension. ShinyLearner accepts files that have been compressed with the gzip algorithm (using ".gz" as the file extension). Users may specify >1 data file as input, after which ShinyLearner will identify sample identifiers that overlap among the files and merge on those identifiers. If the user specifies, ShinyLearner will scale numeric values, one-hot encode categorical variables [39], and impute missing values.

ShinyLearner supports 2 schemes for evaluating predictive performance: Monte Carlo cross-validation and k-fold cross-validation [40, 41]. In Monte Carlo cross-validation, the data are split randomly into a training and validation set; the algorithm is allowed to access the class labels for the training data only. Later the algorithm makes predictions for the validation samples, and the accuracy of those predictions is evaluated using various metrics. Typically, this process is repeated many times to derive confidence intervals for the accuracy metrics. In k-fold cross-validation, the process is similar, except that the data are partitioned into evenly sized groups and each group is used as a validation set through rounds of training and testing. When multiple algorithms or hyperparameter combinations are used, ShinyLearner evaluates nested training and validation sets, with the goal of identifying the optimal combination for each algorithm. Then it uses these selections when making predictions on the outer validation set. Nested cross-validation is also used for feature selection; a feature selection algorithm ranks the features within each nested training set, and different quantities of top-ranked features are used to train the classification algorithm. The feature subsets that perform best are used in making the outer validation set predictions. Hyperparameter optimization and feature selection may be combined; however, such analyses are highly computationally intensive for large benchmarks.

All outputs are stored in tab-delimited files, thus enabling users to import results directly into external analysis tools. ShinyLearner produces output files that contain the following information for each combination of algorithm, hyperparameters, and cross-validation iteration: (i) predictions for each sample, (ii) classification metrics, (iii) execution times, and (iv) standard output, including a log that indicates the arguments that were used, thus supporting reproducibility. When nested cross-validation is performed, ShinyLearner produces output for every hyperparameter combination that was tested in the nested folds and indicates which combination performed best for each algorithm.

ShinyLearner supports the following classification metrics:

- AUROC (area under the receiver operating characteristic curve) [42]
- Accuracy (proportion of samples whose discrete prediction was correct)
- Balanced accuracy (to account for class imbalance)
- Brier score [43]
- F1 score [44]
- False discovery rate
- False-negative rate
- False-positive rate
- Matthews correlation coefficient [45]
- Mean misclassification error
- Negative predictive value
- Positive predictive value
- Recall (sensitivity)
- True-negative rate (specificity)
- True-positive rate (sensitivity)

To calculate these metrics and to perform other data-processing tasks, ShinyLearner uses the AUC [46], mlr [15], dplyr [47], data.table [48], and readr [49] packages. For multi-class problems, ShinyLearner allows the underlying machine-learning packages to use whatever strategy they have implemented for classifying with multiple classes. ShinyLearner then calculates performance metrics in a one-versus-rest manner and averages results across the class options.

When feature selection is performed, each algorithm produces a ranked list of features for each nested training set. To aid the user in understanding which features are most informative, ShinyLearner aggregates these ranked lists using the ”Borda count” method [50]. These aggregate rankings are stored in tab-delimited output files.

The steps of preparing the data and executing ShinyLearner for the results described in this article are in a Jupyter notebook (see [51]). We used the ggplot2 and cowplot packages [52, 53] to create figures.

Analyses

ShinyLearner enables researchers to perform classification benchmark studies. To illustrate this functionality, we performed 3 types of benchmark: (i) basic classification with default hyperparameters, (ii) classification with hyperparameter optimization, and (iii) classification with feature selection. For each analysis, we used 10 classification algorithms:

- `keras/dnn`—Deep neural networks (implemented in Keras/TensorFlow) [29, 31, 54]
- `mlr/h2o.randomForest`—Random forests (implemented in mlr, h2o) [15,30]
- `mlr/mlp`—Multilayer perceptron (mlr) [55]
- `mlr/xgboost`—xgboost (mlr) [56]
- `sklearn/decision_tree`—Decision tree (implemented in scikit-learn) [13, 57]
- `sklearn/logistic_regression`—Logistic regression with the LIBLINEAR solver (scikit-learn) [58]
- `sklearn/svm`—Support vector machines (scikit-learn) [59]
- `weka/HoeffdingTree`—Hoeffding tree (implemented in Weka) [14, 60]
- `weka/MultilayerPerceptron`—Multilayer perceptron (Weka)
- `weka/SimpleLogistic`—Simple logistic regression (Weka) [61]

In the third analysis, we used 10 feature selection algorithms:

- `mlr/kruskal.test`—Kruskal-Wallis rank sum test (mlr) [62]
- `mlr/randomForestSRC.rfsrc`—Permuted random forests (mlr) [63]
- `sklearn/mutual_info`—Mutual information (scikit-learn) [64]
- `sklearn/random_forest_rfe`—Random forests—recursive feature elimination (scikit-learn) [65, 66]
- `sklearn/svm_rfe`—Support vector machines—recursive feature elimination (scikit-learn) [66]
- `weka/Correlation`—Pearson correlation (Weka) [67]
- `weka/GainRatio`—Information gain ratio (Weka) [57]
- `weka/OneR`—OneR (Weka) [68]
- `weka/ReliefF` (Weka) [69]
- `weka/SymmetricalUncertainty`—Symmetrical uncertainty (Weka) [70]

In each analysis, we used 5 rounds of Monte Carlo cross-validation. For the second and third analyses, we used 3 rounds of *nested* Monte Carlo cross-validation for each *outer* round of cross-validation. In the third analysis, we evaluated the top-ranked 1, 3, 5, 10, 15, 20, 50, and 200 features and identified the best of these options via nested cross-validation. In evaluating the results, we focused on AUROC because this metric can be applied to probabilistic predictions and accounts for class imbalance.

As an initial test, we generated a “null” dataset using numpy [71]. We used this dataset to verify that ShinyLearner produces classification results in line with random-chance expectations when no signal is present. This dataset consisted of 20 numeric variables (mean = 0, standard deviation = 1) and 10 categorical variables across 500 simulated samples. AUROC values for all classification algorithms were near 0.5, as expected by random chance, irrespective of whether hyperparameter optimization or feature selection was performed (Fig. S1).

Next, we collected 10 biomedical datasets from the Penn Machine Learning Benchmarks repository [72]:

- Acquired Immune Deficiency Syndrome (AIDS) categorical data [73]
- Thyroid disease [57]
- Breast cancer [74]

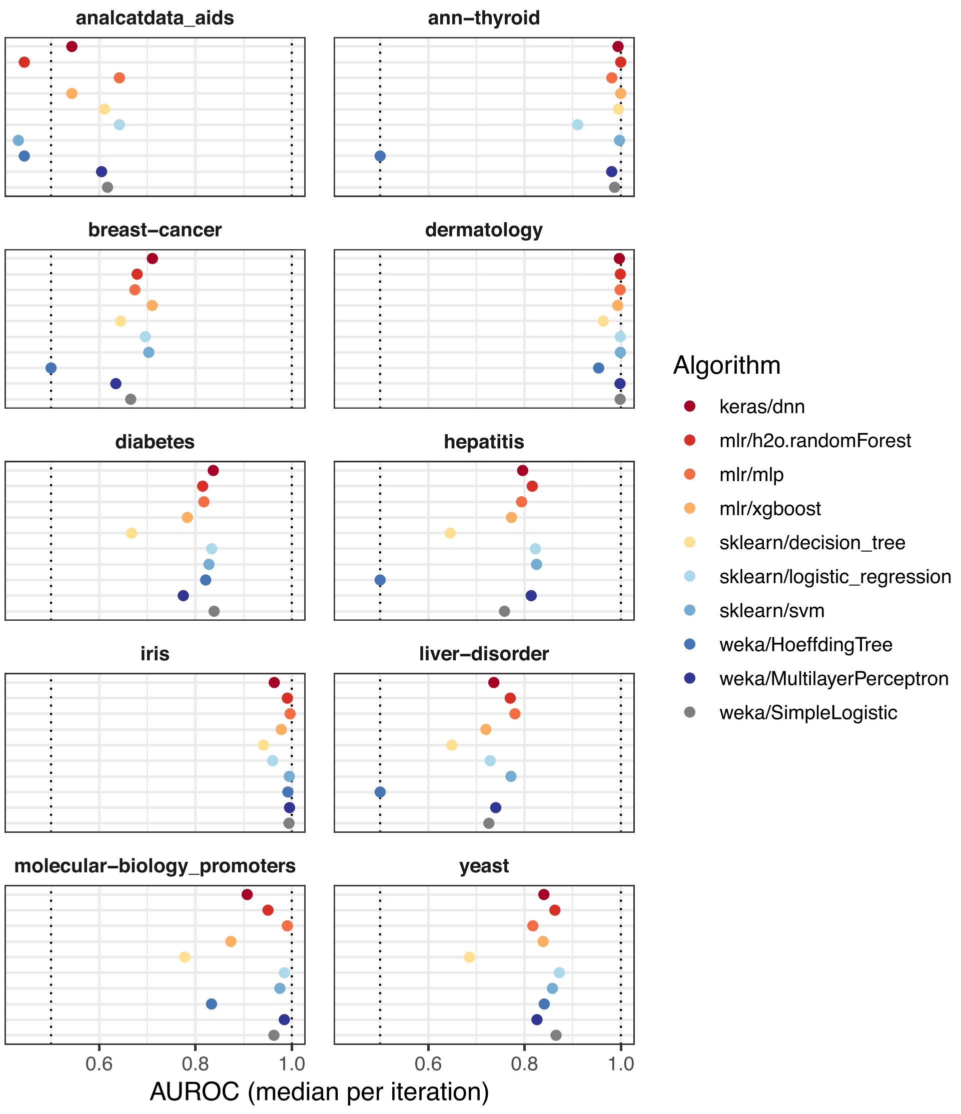

Figure 2: Classification performance per dataset (default hyperparameters). We evaluated the predictive performance of 10 classification algorithms on 10 biomedical datasets. These results were generated using default hyperparameters for each algorithm. We measured predictive performance using area under the receiver operating characteristic curve (AUROC) and calculated the median across 5 Monte Carlo iterations. Predictive performance differed considerably across and within the datasets.

- Dermatology [75]
- Diabetes
- Hepatitis [76]
- Iris [77]
- Liver disorder [78]
- Molecular biology (promoter gene sequences) [79]
- Yeast [80]

These datasets vary by number of samples (minimum = 51; maximum = 7,201) and number of features (minimum = 5; maximum = 172). For all datasets, we converted categorical variables to multiple binary variables using one-hot encoding. When executing ShinyLearner, we scaled numeric values using scikit-learn's RobustScaler, which subtracts the median and scales the data based on the interquartile range [81]; accordingly, this method is robust to outliers. In addition, we used ShinyLearner to impute missing values; this method uses the median for numeric variables and the mode for categorical variables.

Classification analysis with default hyperparameters

Initially, we applied 10 classification algorithms to 10 biomedical datasets using default hyperparameters. Most algorithms made near-perfect predictions for the Thyroid, Dermatology, and Iris datasets, whereas predictions were less accurate overall for the remaining datasets (Fig. 2). The `weka/HoeffdingTree` and `sklearn/decision_tree` algorithms often underperformed relative to the other algorithms (Fig. S2). Indeed, for half of the datasets, `weka/HoeffdingTree` performed as poorly or worse than would be expected by random chance. The remaining 8 classification algorithms performed relatively well, but predictive performance varied considerably across the datasets (Fig. S3). For example, the AUROC for `mlr/mlp` and `sklearn/logistic_regression` was 0.07 higher than the median on the AIDS dataset; the AUROC for `sklearn/svm` was 0.14 lower than the median.

Across the Monte Carlo iterations for each dataset, the predictive performance of `sklearn/decision_tree` and weka/MultilayerPerceptron varied most, whereas weka/HoeffdingTree varied least (in part because AUROC was frequently 0.5) (Fig. S4). The keras/dnn and mlr/h2o.randomForest algorithms took longest to execute, whereas sklearn/svm and sklearn/logistic_regression were among the fastest (and most accurate) algorithms (Fig. S5). Two pairs of classification algorithms use similar theoretical approaches but were implemented in different machine-learning libraries; multilayer perceptron was implemented in Weka and mlr; logistic regression was implemented in Weka and scikit-learn. The AUROC values were strongly—but not perfectly—correlated between these pairs of implementations (Figs S6 and S7). For both of these algorithms, the available hyperparameters as well as options that users can select are considerably different between the underlying machine-learning libraries.

With the exception of sklearn/decision_tree, all classification algorithms produced sample-wise, probabilistic predictions. We examined these predictions for the Diabetes dataset and found that the range and shape of these predictions differed widely across the algorithms (Fig. 3). Although many classification metrics, including AUROC, can cope with distributional differences, these differences must be considered in multiple classifier systems [82].

Classification analysis with hyperparameter optimization

In the second analysis, we applied the same classification algorithms to the same datasets but allowed ShinyLearner to perform hyperparameter optimization via nested cross-validation. As few as 2 (mlr/xgboost) and as many as 95 (sklearn/decision_tree and weka/MultilayerPerceptron) hyperparameter combinations were available for each algorithm. In nearly every example, classification performance improved after hyperparameter optimization (Fig. 4), sometimes dramatically. The performance improvements were most drastic for the weka/HoeffdingTree and sklearn/decision_tree algorithms, which often performed poorly with default parameters.

ShinyLearner supports 53 hyperparameter combinations for the keras/dnn algorithm. Each of these combinations altered the algorithm's performance at least to a small degree on every dataset (Fig. S8). The Thyroid dataset varied least across the hyperparameter combinations, perhaps because the number of instances ($n = 7{,}200$) was nearly 10 times larger than any other dataset. Generally, this algorithm performed better with a wider architecture containing only 2 layers. Having a wider structure greatly increases the parameter space of the network and allows it to learn more complex relationships among features, while limiting the network to only 2 layers prevents overfitting, a common problem when applying neural networks to datasets with a limited number of instances. In addition, adding dropout and L2 regularization also helps to prevent the network from overfitting. In tuning these hyperparameters, we found that a smaller dropout rate, more training epochs, and a smaller regularization rate resulted in higher AUROC values (Fig. S9). Fig. S10 illustrates for the Diabetes dataset that diagnosis predictions can differ considerably, depending on which hyperparameter combination is used.

Classification analysis with feature selection

In any dataset, some features are likely to be more informative than other features. We used ShinyLearner to perform feature selection (via nested cross-validation) before classification. In total, we evaluated 100 unique combinations of feature selection algorithm and classification algorithm (with default hyperparameters). In 44% of cases, feature selection increased the median AUROC, whereas it decreased AUROC in 39% of cases (Fig. 5). Feature selection sometimes improved the performance of weka/HoeffdingTree and sklearn/decision_tree, which were the lowest performers without feature selection.

Fig. 6 illustrates the relative predictive ability of each combination of feature selection and classification algorithms. The mlr/randomForestSRC.rfsrc and sklearn/random_forest_rfe algorithms performed best on average; both approaches use the random forests algorithm to evaluate feature relevance. The weka/OneR algorithm, which evaluates a single feature at a time in isolation, performed worst. Across the datasets, the combination of mlr/randomForestSRC.rfsrc (feature selection) and mlr/xgboost (classification) performed best. Perhaps surprisingly, the combination of sklearn/svm_rfe (feature selection) and sklearn/svm (classification), which are both based on Support Vector Machines, was ranked in the bottom quartile.

In seeking to identify the most informative features, ShinyLearner evaluated various quantities of top-ranked features via nested cross-validation. Fig. 7 illustrates the relative performance of each of these quantities on each dataset. In all cases but 1, using 1 feature performed worst. Generally, a larger number of features resulted in higher AUROC values. However, more features sometimes decreased performance. For example, on the Breast Cancer dataset, the highest AUROC values were attained using 3 of 14 features.

ShinyLearner can inform users about which features are most informative for classification. In the Dermatology dataset, these feature ranks were highly consistent across the feature selection algorithms (Fig. S11). The goal of this classification problem was to predict a patient's type of erythematosquamous disease. Elongation and clubbing of the rete ridges, as well as thinning of the suprapapillary epidermis, were most highly informative of disease type, whereas features such as the patient's age were less informative.

Discussion

The machine-learning community has developed an abundance of algorithms and software implementations of those algorithms. Life scientists use these resources for many research applications. But they face the challenge of identifying which algorithms and hyperparameters will be most accurate and which features are most informative for a given dataset. Many researchers limit classification analyses to a single algorithm, perhaps one that is familiar to them or that has been reported in the literature for a similar study. Others may try a large number of algorithms; however, performing benchmark comparisons in an ad hoc manner requires a considerable coding effort and can introduce biases if done improperly. Alternatively, some researchers may develop new algorithms without providing evidence that these algorithms outperform existing ones. We developed ShinyLearner as a way to simplify the process of performing classification benchmark studies.

ShinyLearner does not implement any classification or feature selection algorithm; rather, it serves as a wrapper around existing software implementations. Currently, algorithms from Weka, scikit-learn, mlr, h2o, and Keras are supported in ShinyLearner. In aggregate, these algorithms represent a diverse range of methodological approaches and thus can support

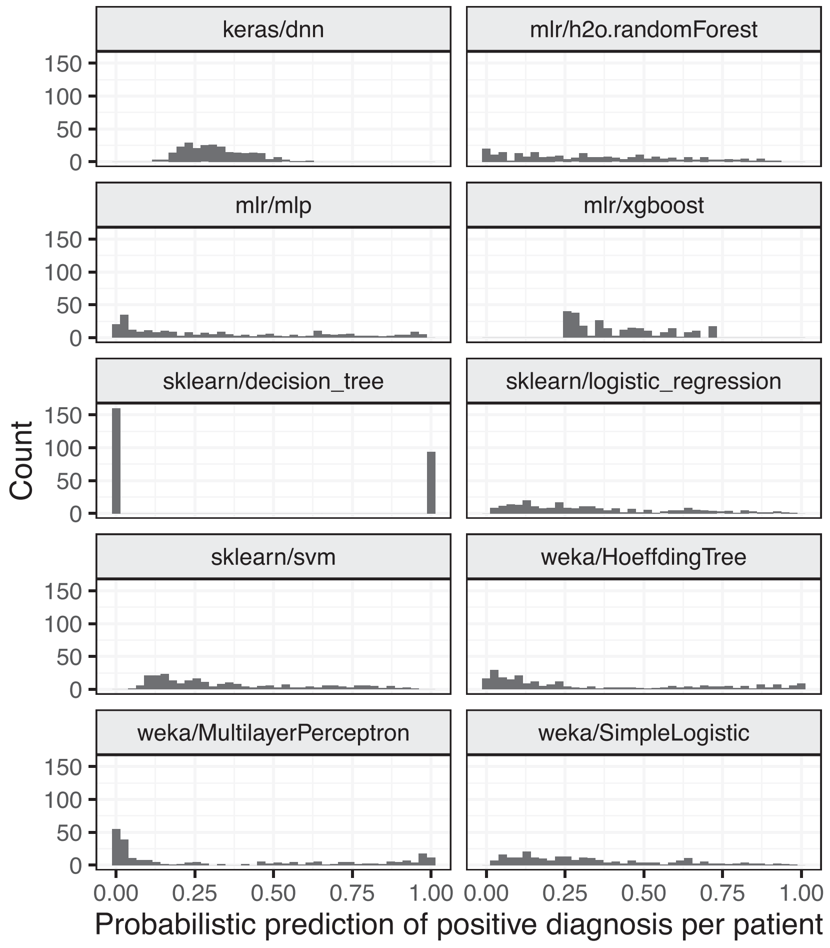

Figure 3: Sample-level predictions for each algorithm on the Diabetes dataset (default hyperparameters). The Diabetes dataset includes a class variable indicating whether patients received a positive diagnosis. Each panel of this figure shows positive-diagnosis predictions for each classification algorithm. All algorithms except sklearn/decision_tree produced probabilistic predictions. The range and distribution of these predictions differed greatly across the algorithms.

comprehensive benchmark evaluations. On their own, each of the third-party tools encapsulated within ShinyLearner provides a way to optimize hyperparameters programmatically and perform feature selection. In addition, tools such as caret [17], KNIME [18], and Orange [83] provide these options. Thus, in situations where a researcher has programming expertise and is satisfied with the algorithms and tuning functionality available in one of those tools, the researcher might prefer to use these tools directly rather than use ShinyLearner. ShinyLearner is most useful when a researcher:

1. wishes to compare algorithms that have been implemented in multiple machine-learning packages,
2. does not have programming expertise,
3. desires to perform complex operations via nested cross-validation, such as evaluating different sizes of feature subsets,
4. wishes to analyze algorithm performance using a tool or programming language that is different than was used to perform classification,
5. wishes to gain deeper insight into decisions made during nested cross-validation, and/or
6. seeks to evaluate the tradeoff between predictive accuracy and time of execution.

Although many of these tasks could be performed by a researcher who has programming expertise, care must be taken to ensure that the steps are performed in a robust manner (e.g., not mixing training and test sets in nested validation). In addition, we hope ShinyLearner will increase the efficiency of such benchmark studies by reducing duplicate efforts.

ShinyLearner is limited to datasets that fit into computer memory. For larger datasets, frameworks such as Apache SystemML support distributed algorithm execution [84]; however, the number of algorithms implemented in these frameworks is still relatively small.

The current release of ShinyLearner supports diverse classification algorithms and hyperparameter combinations; however, this collection is far from exhaustive. Using ShinyLearner's extensible architecture, the research community can integrate additional algorithms and hyperparameter combinations. In addition, algorithm designers can use our framework to compare their algorithms against competing methods and disseminate their algorithms to the research community.

Containers provide many advantages for software deployment. Tool installation and computational reproducibility are easier because all software components are encapsulated within the container, and container images can be archived and versioned [85]. One other benefit may be less apparent: container-

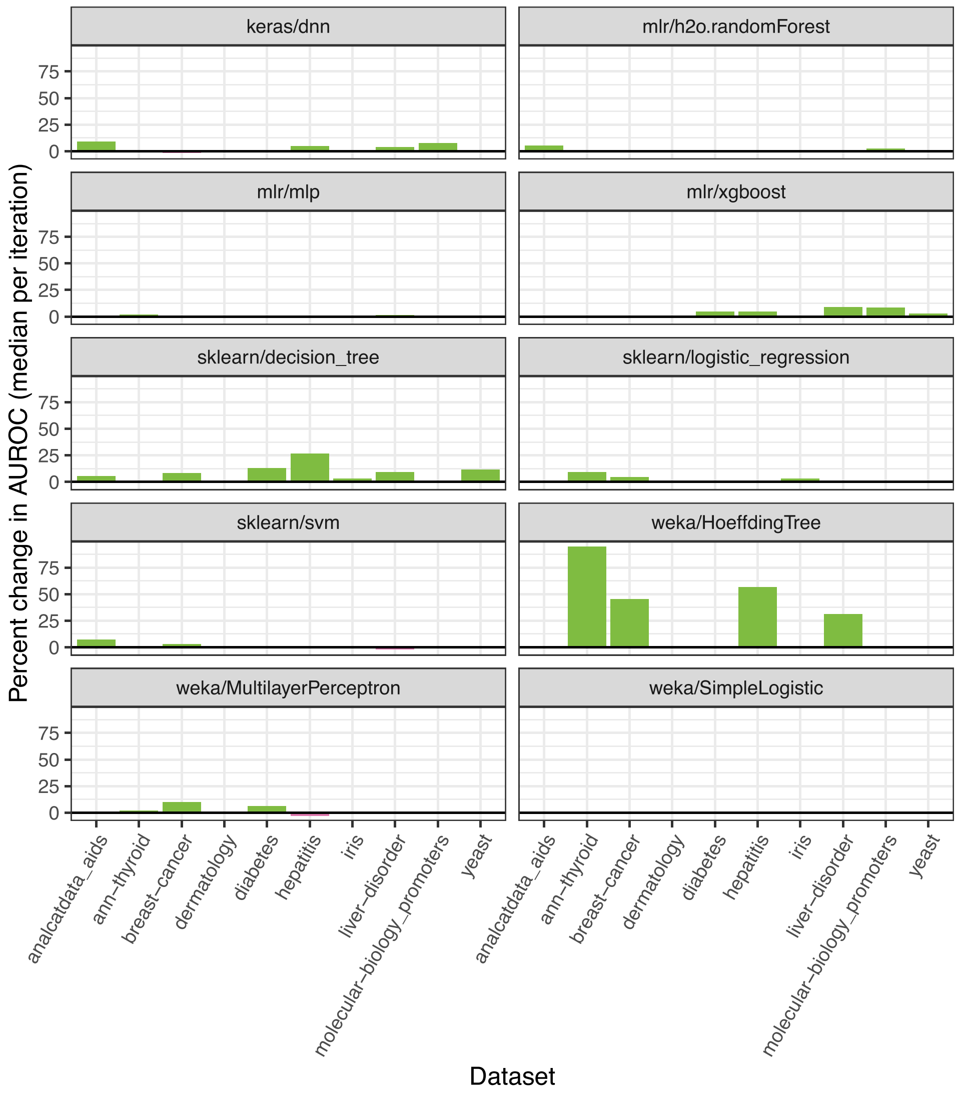

Figure 4: Classification performance when optimizing vs not optimizing hyperparameters. We tested 10 classification algorithms on 10 biomedical datasets and used nested cross-validation to select hyperparameters. To evaluate for change in predictive performance, we calculated the percent change in the median AUROC values when using optimized vs default hyperparameters. Most algorithms demonstrated improved classification performance with optimized hyperparameters.

ization facilitates the use of diverse programming languages. Distinct components of ShinyLearner are implemented in 4 different programming languages. We chose this approach because we determined that each language was suited to specific types of tasks. We posit that the future of bioinformatics development will increasingly follow this pattern. Furthermore, we advocate for the approach of providing a graphical user interface, such as the Web-based tool we provided for ShinyLearner. Such tools make it easier for users—especially those who have limited command line experience—to formulate Docker commands.

Our analysis of 10 biomedical datasets, 10 classification algorithms, and 10 feature selection algorithms confirmed that the choice of algorithm and hyperparameters has a considerable effect on classification performance and selected features. Although some algorithms typically performed better than others, no single algorithm consistently outperformed any other. This finding supports the “No Free Lunch” theorem [86] and confirms that multiple classifier systems hold promise for aggregating evidence across algorithms [87]. Also importantly, algorithm performance is likely to differ according to data characteristics. Algorithms that perform well on “wide” datasets (many features, few samples) may not perform as well on “tall” datasets. Algorithms that perform well with numeric data may not perform as well on categorical or mixed data. These differences highlight the importance of domain-specific benchmark comparisons.

Finally, we offer recommendations regarding benchmark comparisons. When performing benchmarks across multiple algorithms and/or hyperparameters, it is important to exercise caution in interpreting those results. Below are recommendations on performing benchmarks and interpreting such results:

- If you apply multiple algorithms or hyperparameter combinations, you should always report those (e.g., in the Methods section of a journal article). It is poor form to report only the best results.
- After you have identified the best-performing algorithm and/or hyperparameters, it is usually best to test those find-

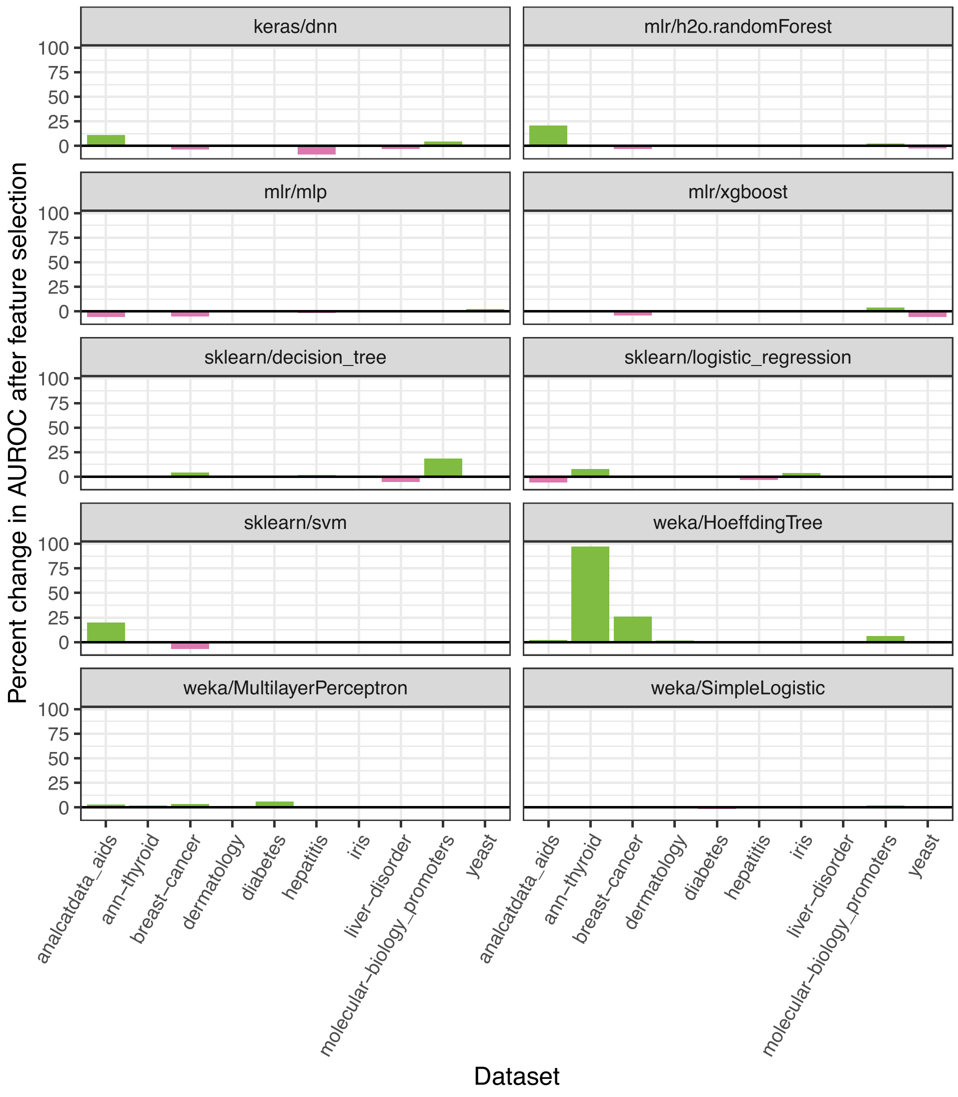

Figure 5: Classification performance when performing feature selection vs not performing feature selection. In combination with classification, we performed feature selection via nested cross-validation on 10 biomedical datasets. For each algorithm, we used default hyperparameters. These plots show the percent change in the median AUROC when using vs not using feature selection. Although the effects of feature selection varied across the algorithms, median AUROCs increased in many cases.

ings on a completely independent dataset that was not used in the benchmark comparison.

- Merely because an algorithm (or parameter) appears to work well in one setting doesn't necessarily mean that the same will be true in alternate settings.

Availability of Source Code and Requirements

- Project name: ShinyLearner
- Project home page: https://github.com/srp33/ShinyLearner
- Operating system(s): Any operating system on which Docker can be installed
- Programming languages: Java, Python, R, bash
- Other requirements: Docker (https://docker.com)
- License: MIT
- DOIs of Zenodo archives of GitHub repositories: 10.5281/zenodo.3543724, 10.5281/zenodo.3543726, 10.5281/zenodo.3543728, 10.5281/zenodo.3543730

The code for creating the figures in this article can be found and re-executed in a Code Ocean capsule [88].

This tool has been registered at bio.tools (https://bio.tools/ShinyLearner) and at SciCrunch.org (ShinyLearner, RRID:SCR_017608).

Availability of Supporting Data and Materials

Archives of the GitHub repositories are available in Zenodo [89–92]. Snapshots of all of the archives are also available in the *GigaScience* GigaDB repository [93].

Editor's Note

A CODECHECK certificate for this paper is available confirming that the figures in the paper could be independently reproduced [94].

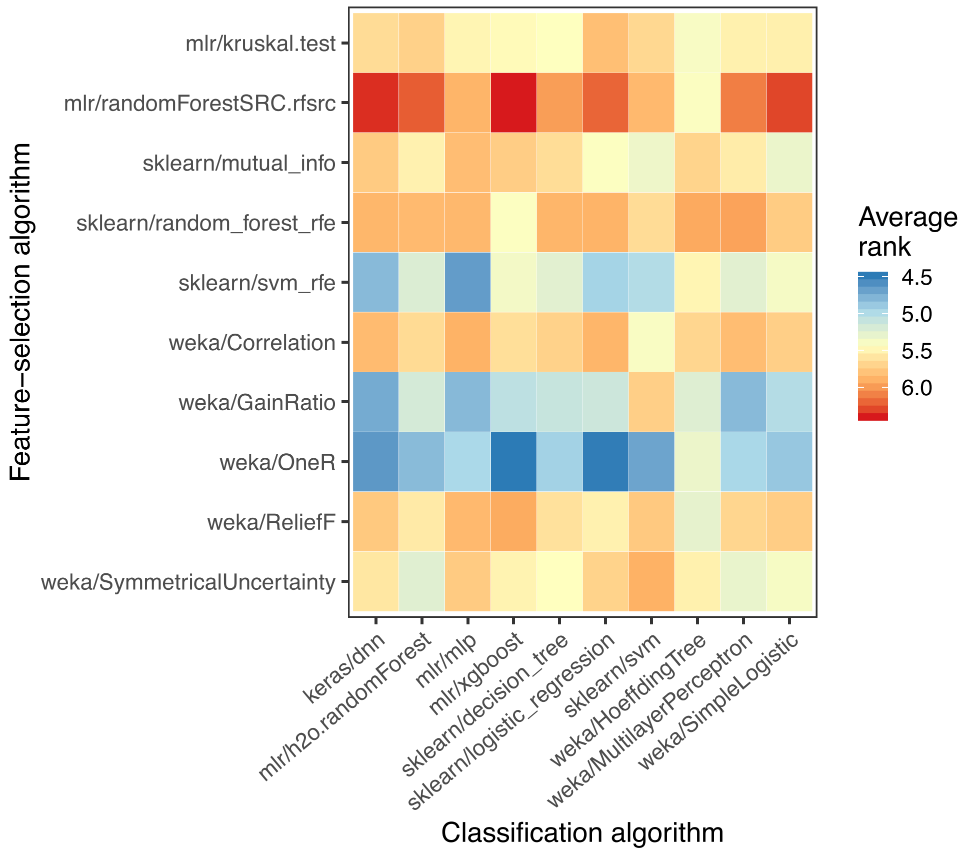

Figure 6: Performance for each combination of classification and feature selection algorithm. This figure shows classification results for the nested cross-validation folds across each combination of feature selection algorithm and classification algorithm. Averaged across all datasets and classification algorithms, we ranked the feature selection algorithms based on AUROC values attained for nested validation sets. For simplicity and consistency across the datasets, this figure shows only the results when the top 5 features were used. Higher average ranks indicate better classification performance.

Additional Files

Figure S1: Classification performance on the "null" dataset. To verify ShinyLearner's functionality, we randomly generated a "null" dataset and applied 3 types of analysis to the data. In the Basic analysis, default hyperparameters were used for each algorithm. In the second analysis, we used the same algorithms but used nested cross-validation to select hyperparameters. In the third analysis, we performed feature selection via nested cross-validation. For each analysis, area under the receiver operating characteristic curve (AUROC) was consistently close to 0.5, as expected by random chance. The vertical, dotted lines on the left represents an AUROC of 0.5; the dotted lines on the right represent an AUROC of 1.0 (perfect predictions).
Figure S2: Classification performance per algorithm relative to other classification algorithms (default hyperparameters). We evaluated the predictive performance of 10 classification algorithms on 10 biomedical datasets. These results were generated using default hyperparameters for each algorithm. For each dataset, we calculated the AUROC for each algorithm relative to the median across all algorithms. The weka/HoeffdingTree and sklearn/decision_tree algorithms underperformed in comparison to the other algorithms.
Figure S3: Consistency of results across datasets for each algorithm (default hyperparameters). We evaluated the consistency of area under the receiver operating characteristic curve (AUROC) values for each algorithm across the datasets. After calculating the median AUROC across Monte Carlo iterations, we calculated the coefficient of variation (expressed as percentages) across the datasets. The weka/HoeffdingTree algorithm varied most across the datasets, while sklearn/logistic_regression varied least.
Figure S4: Consistency of results across Monte Carlo iterations for each algorithm (default hyperparameters). We evaluated the consistency of area under the receiver operating characteristic curve (AUROC) values across Monte Carlo iterations within each dataset and then calculated the median across the datasets for each algorithm. These values are coefficients of variation (expressed as percentages). The weka/HoeffdingTree algorithm varied least, while sklearn/decision_tree varied most.
Figure S5: Relationship between execution time and predictive performance per classification algorithm. Across 10 biomedical datasets, execution time and area under the receiver operating characteristic curve (AUROC) differed considerably. Each point represents the median value across all datasets for a single Monte Carlo iteration. We observed little to no association between execution time and predictive performance. Some of the best-performing algorithms were also quite fast.
Figure S6: Comparison of classification performance between 2 implementations of the multilayer perceptron algorithm (default hyperparameters). We evaluated the predictive performance (area under the receiver operating characteristic curve) for 2 implementations of the multilayer perceptron classification algorithm. We compared implementations from the weka and mlr software packages. Predictive performance was highly

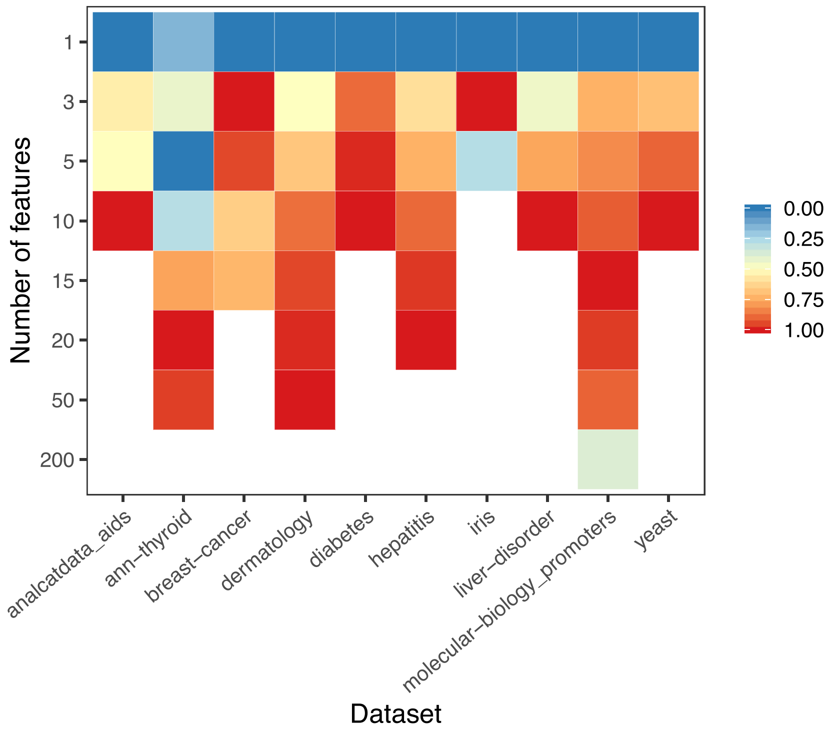

Figure 7: Median classification performance of feature selection algorithms by number of features. We applied feature selection to each dataset, in combination with each of the 10 classification algorithms. For each algorithm, we selected the top *x* number of features and averaged across each combination of feature selection and classification algorithm. This figure shows which values of *x* resulted in the highest AUROC values for each dataset. Different datasets had different quantities of features; this graph only shows results for *x* values relevant to each dataset. Accordingly, we scaled the AUROC values in each column between 0 and 1 to ensure that the comparisons were consistent across all datasets. Higher values indicate better classification performance. Generally, a larger number of features resulted in better classification performance, but this varied across the datasets.

consistent but not identical. We used the Pearson method to calculate the correlation coefficient.

Figure S7: Comparison of classification performance between 2 implementations of the logistic regression algorithm (default hyperparameters). We evaluated the predictive performance (area under the receiver operating characteristic curve) for 2 implementations of the logistic regression classification algorithm. We compared implementations from the weka and scikit-learn (sklearn) software packages. Predictive performance was highly consistent but not identical. We used the Pearson method to calculate the correlation coefficient.

Figure S8: Performance of different hyperparameter combinations for the keras/dnn classification algorithm. We evaluated predictive performance for the keras/dnn algorithm using 53 different hyperparameter combinations. Each dataset was affected by the combinations to some degree. The Thyroid dataset demonstrated the least variability, possibly due to its large number of instances. Relatively cool colors indicate hyperparameter combinations that result in relatively worse performance relative to default hyperparameters, whereas warmer colors indicate the opposite.

Figure S9: Effect of changing different hyperparameters on performance of the keras/dnn algorithm. We evaluated predictive performance for the keras/dnn algorithm using 53 different hyperparameter combinations. The level of performance varied depending on the hyperparameter combination used. Generally, AUROC values increased when using a smaller dropout rate, more training epochs, a wider layer structure, and a smaller regularization rate.

Figure S10: Probabilistic predictions of positive diagnosis for patients in the Diabetes dataset for different hyperparameter combinations. The Diabetes dataset includes a class variable indicating whether patients received a positive diagnosis. This figure shows probabilistic predictions of a positive diagnosis for 3 patients with diabetes; the predictions were made using the keras/dnn algorithm. Each line represents prediction probabilities across different hyperparameter combinations for each of the 3 patients with diabetes.

Figure S11: Rankings of each feature in the Dermatology dataset for each feature selection algorithm. In this example, 10 feature selection algorithms were applied to the Dermatology dataset. Each cell represents the average rank of each feature across nested cross-validation folds. Lower average ranks indicate greater relevance of the feature to the class variable (the patient's type of erythematosquamous disease). The average ranks were largely consistent across the feature selection algorithms.

Abbreviations

AIDS: Acquired Immune Deficiency Syndrome; API: application programming interface; AUROC: area under receiver operating characteristic curve.

Competing Interests

The authors declare that they have no competing interests.

Funding

S.R.P. was supported by internal funds from Brigham Young University. T.J.L. and K.H. were supported by fellowships from the Simmons Center for Cancer Research at Brigham Young University.

Authors' Contributions

S.R.P., T.J.L., and K.H. helped to develop the software. S.R.P. conceived of the software design with critical input from T.J.L. and K.H. E.S. and S.R.P. performed the analyses. All authors helped to write the manuscript.

References

1. Shipp MA, Ross KN, Tamayo P, et al. Diffuse large B-cell lymphoma outcome prediction by gene-expression profiling and supervised machine learning. Nat Med 2002;**8**:68–74.
2. Nutt CL, Mani DR, Betensky RA, et al. Gene expression-based classification of malignant gliomas correlates better with survival than histological classification. Cancer Res 2003;**63**:1602–7.
3. Yuan Y, Van Allen EM, Omberg L, et al. Assessing the clinical utility of cancer genomic and proteomic data across tumor types. Nat Biotechnol 2014;**32**:644–52.
4. Bilal E, Dutkowski J, Guinney J, et al. Improving breast cancer survival analysis through competition-based multidimensional modeling. PLoS Comput Biol 2013;**9**:e1003047.
5. Piccolo SR, Andrulis IL, Cohen AL, et al. Gene-expression patterns in peripheral blood classify familial breast cancer susceptibility. BMC Med Genomics 2015;**8**:72.
6. Piccolo SR, Frey LJ. Clinical and molecular models of glioblastoma multiforme survival. Int J Data Min Bioinform 2013;**7**:245–65.
7. Desautels T, Calvert J, Hoffman J, et al. Prediction of sepsis in the intensive care unit with minimal electronic health record data: a machine learning approach. JMIR Med Inform 2016;**4**:e28.
8. Szlosek DA, Ferrett J. Using machine learning and natural language processing algorithms to automate the evaluation of clinical decision support in electronic medical record systems. EGEMS (Wash DC) 2016;**4**:1222.
9. Statnikov A, Aliferis CF, Tsamardinos I, et al. A comprehensive evaluation of multicategory classification methods for microarray gene expression cancer diagnosis. Bioinformatics 2005;**21**:631–43.
10. Kim J-W, Sharma V, Ryan ND. Predicting methylphenidate response in ADHD using machine learning approaches. Int J Neuropsychopharmacol 2015;**18**:pyv052.
11. Braman NM, Etesami M, Prasanna P, et al. Intratumoral and peritumoral radiomics for the pretreatment prediction of pathological complete response to neoadjuvant chemotherapy based on breast DCE-MRI. Breast Cancer Res 2017;**19**:57.
12. Libbrecht MW, Noble WS. Machine learning applications in genetics and genomics. Nat Rev Genet 2015;**16**:321–32.
13. Pedregosa F, Varoquaux G, Gramfort A, et al.. Scikit-learn: Machine learning in Python. J Mach Learn Res 2011;**12**:2825–30.
14. Frank E, Hall M, Holmes G, et al. Weka-a machine learning workbench for data mining. In: Maimon O, Rokach L , eds. Data Mining and Knowledge Discovery Handbook. 2nd ed. New York: Springer; 2010:1269–77.
15. Bischl B, Lang M, Kotthoff L, et al. mlr: Machine learning in R. J Mach Learn Res 2016;**17**:1–5.
16. Piccolo SR, Frey LJ. ML-Flex: a flexible toolbox for performing classification analyses in parallel. J Mach Learn Res 2012;**13**:555–9.
17. Kuhn M.Building predictive models in R using the caret package. J Stat Soft 2008;**28**:1–26.
18. Berthold MR, Cebron N, Dill F, et al. KNIME: The Konstanz Information Miner. Studies in Classification, Data Analysis, and Knowledge Organization (GfKL 2007). Springer; 2007.
19. Guyon I, Elisseeff A. An introduction to variable and feature selection. J Mach Learn Res 2003;**3**:1157–82.
20. Dougherty ER, Hua J, Sima C. Performance of feature selection methods. CurrGenomics 2009;**10**:365–74.
21. Varma S, Simon R. Bias in error estimation when using cross-validation for model selection. BMC Bioinformatics 2006;**7**:91.
22. Statnikov A, Wang L, Aliferis CF. A comprehensive comparison of random forests and support vector machines for microarray-based cancer classification. BMC Bioinformatics 2008;**9**:319.
23. Fernández-Delgado M, Cernadas E, Barro S, et al. Do we need hundreds of classifiers to solve real world classification problems? J Mach Learn Res 2014;**15**:3133–81.
24. Piccolo SR, Frampton MB. Tools and techniques for computational reproducibility. Gigsci 2016;**5**, doi:10.1186/s13742-016-0135-4.
25. ShinyLearner helper application, https://bioapps.byu.edu/shinylearner. Accessed December 12, 2019.
26. Wickham H. Tidy data. J Stat Soft 2014;**59**, doi:10.18637/jss.v059.i10.
27. Docker. https://www.docker.com/. Accessed June 2019.
28. ShinyLearner Image on DockerHub. https://hub.docker.com/r/srp33/shinylearner/. Accessed December 12, 2019.
29. Chollet F . Keras. https://keras.io/. Accessed 18 June 2019.
30. Cook D. Practical Machine Learning with H2O: Powerful, Scalable Techniques for Deep Learning and AI. O'Reilly; 2016.
31. Abadi M, Barham P, Chen J, et al. TensorFlow: A system for large-scale machine learning. In: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), Savannah, GA. 2016:265–83.
32. Python Software Foundation. Python Language Reference, version 3.6. 2013. https://docs.python.org/release/3.6.0/. Accessed 18 June 2019.
33. R Core Team. R: A Language and Environment for Statistical Computing. Vienna, Austria: R Foundation for Statistical Computing; 2019. https://www.r-project.org/. Accessed 18 June 2019.
34. Java Software | Oracle. https://www.oracle.com/java/. Accessed 18 June 2019.
35. ShinyLearner (GPU version) Image on DockerHub. https://hub.docker.com/r/srp33/shinylearner_gpu/. Accessed December 12, 2019.
36. Fowler M. Continuous Integration. https://martinfowler.com/articles/continuousIntegration.html. Accessed 18 June 2019.
37. ShinyLearner Continuous Integration Page. https://travis-ci.org/srp33/ShinyLearner. Accessed December 12, 2019.
38. Chang W, Cheng J, Allaire JJ, et al. Shiny: Web Application Framework for R. 2019. https://cran.r-project.org/web/packages/shiny/index.html. Accessed 18 June 2019.
39. Harris D, Harris S. Digital Design and Computer Architecture. Morgan Kaufmann; 2010.
40. Geisser S. The predictive sample reuse method with applications. J Am Statist Assoc 1975;**70**:320–8.

41. Kohavi R. A study of cross-validation and bootstrap for accuracy estimation and model selection. In: International Joint Conference on Artificial Intelligence. San Francisco: Morgan Kaufmann; 1995:1137–45.
42. Green DM, Swets JA. Signal Detection Theory and Psychophysics. New York: Wiley; 1966.
43. Brier GW. Verification of forecasts expressed in terms of probability. Mon Weather Rev 1950;**78**:1–3.
44. Vickery BC. Techniques of Information Retrieval. London: Butterworths; 1970.
45. Matthews BW. Comparison of the predicted and observed secondary structure of T4 phage lysozyme. Biochim Biophys Acta Protein Struct 1975;**405**:442–51.
46. Ballings M, Van den Poel D. AUC: Threshold independent performance measures for probabilistic classifiers. 2013. https://cran.r-project.org/web/packages/AUC/index.html. Accessed 15 June 2018.
47. Wickham H, François R. A Grammar of Data Manipulation [Internet]. 2018. Available from: https://acran.r-project.org/package=dplyr.
48. Dowle M, Srinivasan A. data.table: Extension of 'data.frame` [Internet]. 2018. Available from: https://CRAN.R-project.org/package=data.table.
49. Wickham H, Hester J, Francois R. readr: Read Rectangular Text Data [Internet]. 2018. Available from: https://CRAN.R-project.org/package=readr.
50. Black D. Partial justification of the Borda count. Public Choice 1976;**28**:1–15.
51. ShinyLearner Demo Jupyter Notebook. https://github.com/srp33/ShinyLearner/blob/master/Demo/Execute_Algorithms.ipynb. Accessed December 12, 2019.
52. Wickham H. Ggplot2: Elegant Graphics for Data Analysis. New York: Springer; 2016.
53. Wilke CO. cowplot: Streamlined Plot Theme and Plot Annotations for "ggplot2" [Internet]. 2017. Available from: https://CRAN.R-project.org/package=cowplot.
54. Lecun Y, Bengio Y, Hinton G. Deep learning. Nature 2015;**521**:436–44.
55. Rosenblatt F. Principles of neurodynamics. In: Perceptrons and the Theory of Brain Mechanisms. Buffalo, NY: Cornell Aeronautical Lab Inc; 1961.
56. 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. ACM; 2016:785–94.
57. Quinlan JR. Induction of decision trees. Mach Learn 1986;**1**:81–106.
58. Fan R-E, Chang K-W, Hsieh C-J, et al. LIBLINEAR: A library for large linear classification. J Mach Learn Res 2008;**9**:1871–4.
59. Vapnik VN. Statistical Learning Theory. New York: Wiley; 1998.
60. Hulten G, Spencer L, Domingos P. Mining time-changing data streams. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM; 2001:97–106.
61. Landwehr N, Hall M, Frank E. Logistic model trees. Mach Learn 2005;**59**:161–205.
62. Kruskal WH, Wallis WA. Use of ranks in one-criterion variance analysis. J Am Stat Assoc 1952;**47**:583–621.
63. Ishwaran H, Kogalur UB, Kogalur MUB. Package 'randomForestSRC' [Internet] 2019. Available from: https://CRAN.R-project.org/package=randomForestSRC.
64. Cover TM, Thomas JA. Elements of Information Theory. Wiley; 2012.
65. Breiman L. Random forests. Mach Learn 2001;**45**:5–32.
66. Guyon I, Weston J, Barnhill S, et al. Gene selection for cancer classification using support vector machines. Mach Learn 2002;**46**:389–422.
67. Pearson K, VII. Note on regression and inheritance in the case of two parents. Proc R Soc Lond 1895;**58**:240–2.
68. Holte RC. Very simple classification rules perform well on most commonly used datasets. Mach Learn 1993;**11**:63–90.
69. Kononenko I. Estimating attributes: Analysis and extensions of RELIEF. In: Bergadano F, De Raedt L , eds. Machine Learning ECML94. Berlin/Heidelberg: Springer; 1994:171–82.
70. Witten IH, Frank E, Hall MA, et al. Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann; 2016.
71. Walt S van der, Colbert SC, Varoquaux G. The NumPy Array: a structure for efficient numerical computation. Comput Sci Eng 2011;**13**:22–30.
72. Olson RS, La Cava W, Orzechowski P, et al. PMLB: A large benchmark suite for machine learning evaluation and comparison. BioData Mining 2017;**10**:36.
73. Simonoff JS. Analyzing Categorical Data. New York: Springer; 2003.
74. Michalski RS, Mozetic I, Hong J, et al. The multi-purpose incremental learning system AQ15 and its testing application to three medical domains. In: Proceedings of the Fifth AAAI National Conference on Artificial Intelligence. AAAI Press; 1986:1041–5.
75. Güvenir HA, Demiröz G, Ilter N. Learning differential diagnosis of erythemato-squamous diseases using voting feature intervals. Artif Intelli Med 1998;**13**:147–65.
76. Diaconis P, Efron B. Computer-intensive methods in statistics. Sci Am 1983;**248**:116–31.
77. Fisher RA. The use of multiple measurements in taxonomic problems. Ann Eugen 1936;**7**:179–88.
78. Robinson D, Allaway SL, Ritchie CD, et al. The use of artificial intelligence in the prediction of alcohol-induced fatty liver. In: MEDINFO 89: Proceedings of the Sixth Conference on Medical Informatics, Beijing and Singapore. 1989:170.
79. Harley CB, Reynolds RP. Analysis of *E. coli* promoter sequences. Nucleic Acids Res 1987;**15**:2343–61.
80. Horton P, Nakai K. A probabilistic classification system for predicting the cellular localization sites of proteins. Proc Int Conf Intell Syst Mol Biol 1996;**4**:109–15.
81. Sklearn. Preprocessing.RobustScaler scikit-learn 0.21.2 documentation. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html. Accessed 18 June 2018.
82. Ho TKT, Hull J, Srihari SNS, et al. Decision combination in multiple classifier systems. IEEE Trans Pattern Anal Mach Intell 1994;**16**:66–75.
83. Demšar J, Zupan B, Leban G, et al. Orange: From experimental machine learning to interactive data mining. In: Boulicaut JF, Esposito F, Giannotti F , et al.. Knowledge Discovery in Databases: PKDD 2004. Berlin: Springer; 2004:537–9.
84. Elgohary A, Boehm M, Haas PJ, et al. Compressed linear algebra for large-scale machine learning. VLDB J 2018;**27**:719–44.
85. Boettiger C. An introduction to Docker for reproducible research. SIGOPS Oper Syst Rev 2015;**49**:71–9.
86. Ho YY, Pepyne D. Simple explanation of the no-free-lunch theorem and its implications. J Optim Theory Appl 2002;**115**:549–70.
87. Xu L, Krzyzak A, Suen C. Methods of combining multiple classifiers and their applications to handwriting recognition. IEEE Trans Syst Man Cybern B Cybern 1992;**22**: 418–35.

88. Piccolo S. ShinyLearner Demo [Source Code]. CodeOcean 2019, doi:10.24433/CO.5449763.v1.
89. Piccolo S, Hill K, Suh E, et al. srp33/ShinyLearner: Gigascience (Version 1). Zenodo 2019, doi:10.5281/zenodo.3543724.
90. Piccolo S. srp33/ShinyLearner_gpu: Gigascience (Version 1). Zenodo 2019, doi:10.5281/zenodo.3543726.
91. Piccolo S. srp33/ShinyLearner_Environment_gpu: Giga science (Version 1), Zenodo 2019, doi:10.5281/zenodo.3543728.
92. Piccolo S. srp33/ShinyLearner_Environment: Gigascience (Version 1), Zenodo 2019, doi:10.5281/zenodo.3543730.
93. Piccolo SR, Lee TJ, Suh E, et al. Supporting data for "ShinyLearner: A containerized benchmarking tool for machine-learning classification of tabular data." GigaScience Database 2020. http://dx.doi.org/10.5524/100701.
94. Eglen SJ. CODECHECK Certificate 2020-001. Zenodo 2020, doi:10.5281/zenodo.3674056.