Journal of Data Science 22 (2), 239–258
April 2024

DOI: 10.6339/24-JDS1123
Statistical Data Science

Producing Fast and Convenient Machine Learning Benchmarks in R with the stressor Package

Sam Haycock[1,*], Brennan Bean[1], and Emily Burchfield[2]
[1]*Department of Mathematics and Statistics, Utah State University, Logan, UT, USA*
[2]*Department of Environmental Sciences, Emory University, Atlanta, GA, USA*

Abstract

The programming overhead required to implement machine learning workflows creates a barrier for many discipline-specific researchers with limited programming experience. The `stressor` package provides an `R` interface to `Python`'s `PyCaret` package, which automatically tunes and trains 14-18 machine learning (ML) models for use in accuracy comparisons. In addition to providing an R interface to `PyCaret`, `stressor` also contains functions that facilitate synthetic data generation and variants of cross-validation that allow for easy benchmarking of the ability of machine-learning models to extrapolate or compete with simpler models on simpler data forms. We show the utility of `stressor` on two agricultural datasets, one using classification models to predict crop suitability and another using regression models to predict crop yields. Full ML benchmarking workflows can be completed in only a few lines of code with relatively small computational cost. The results, and more importantly the workflow, provide a template for how applied researchers can quickly generate accuracy comparisons of many machine learning models with very little programming.

Keywords *agriculture; benchmarking; cross-validation; machine learning; Python; R*

1 Introduction

The `R` programming language has long been used for data analysis, especially among academic researchers. Despite `R`'s many advantages, the `Python` programming language has arguably been the leader in machine learning (ML) functionality (Raschka et al., 2020; Bowles, 2015). This innovation includes the development of automated workflows for tuning, training, and evaluating ML models. The `PyCaret` library (Ali, 2020) is an example of one such innovation. It contains access to auto-tuned ML models and a streamlined workflow for accuracy comparisons.

After the training and test sets are automatically created by `PyCaret`, the program tunes hyper-parameters for 14-18 machine learning models and then ranks the models using a user-specified accuracy criteria, such as Root Mean Squared Error (RMSE). Table 1 shows the various ML models available in `PyCaret` for both classification and regression. While `Python` may be the leader in ML model implementation, the `R` programming language has many ancillary software packages that implement ML models. Table 1 also contains references to some `R` packages that replicate each model type available in `PyCaret`, largely derived from Hothorn (2023).

The list of R packages in Table 1 reveals that it would take 18 different R packages to replicate most of the models available in `PyCaret`. While meta-packages such as `tidymodels` (Kuhn

[*]Corresponding author. Email: haycock.sam@outlook.com.

Received July 28, 2023; Accepted February 21, 2024

Table 1: Machine learning (ML) methods available in `PyCaret`, listed alphabetically by token. When a token or name is used for Both, Regression (R) and Classification (C), the C version of the method is listed in parenthesis. Italicized tokens indicate methods that are not available in the default execution of `PyCaret` and are thus not available in the `stressor` interface. The final column contains references to a CRAN package that implements the method. Other acronyms in the table include Gradient Boosting Machine (GBM) and Support Vector Machine (SVM).

Token	**Name**	**Use**	**CRAN Package**
ada	AdaBoost	Both	`ada` (Culp et al., 2016)
ard	*Automatic Relevance Detection*	*R*	n/a
br	Bayesian Ridge	R	n/a
catboost	*CatBoost R/C*	*Both*	n/a
dt	Decision Tree	Both	`rpart` (Therneau and Atkinson, 2022)
dummy	No Explanatory Variables	Both	n/a
en	Elastic Net	R	`glmnet` (Friedman et al., 2010)
et	Extra Tress	C	`ranger` (Wright and Ziegler, 2017)
gbr (gbc)	GBM	Both	`gbm` (Greenwell et al., 2022)
gpc	*Gaussian Process C*	C	`tgp` (Gramacy, 2007)
huber	Huber Regressor	R	`MASS` (Venables and Ripley, 2002)
knn	K Nearest Neighbors	Both	`class` (Venables and Ripley, 2002)
kr	*Kernel Ridge*	*R*	`CVST` (Krueger and Braun, 2022)
lasso	Lasso R	R	`glmnet`
lda	Linear Discriminant Analysis	C	`MASS`
lightgbm	Light GBM	Both	`lightgbm` (Ke et al., 2017)
(l)lar	(Lasso) Least Angle R	R	`lars` (Hastie and Efron, 2022)
lr	Linear (Logistic) R	Both	`stats` (R Core Team, 2022)
mlp	*Multi-layer Perceptron*	Both	`deepnet` (Rong, 2022)
nb	Naive Bayes	C	`naivebayes` (Majka, 2019)
omp	Orthogonal Matching Pursuit	R	`Rfast` (Papadakis et al., 2023)
par	Passive Aggressive Regressor	R	n/a
qda	Quadratic Discriminant Analysis	C	`MASS`
ransac	*Random Sample Consensus*	*R*	n/a
rbfsvm	*SVM - Radial Kernel*	*C*	`e1071` (Meyer et al., 2022)
rf	Random Forest	Both	`ranger`
ridge	Ridge R/C	Both	`glmnet`
svm	SVM (- Linear Kernel)	Both	`e1071`
tr	*TheilSen Regressor*	*R*	`deming` (Therneau, 2018)
xgboost	*Extreme GBM*	*Both*	`xgboost` (Chen et al., 2023)

and Wickham, 2020), `caret` (Kuhn, 2022), and `mlr3` (Lang et al., 2019) bring many of these models together under a common modeling convention, the programming overhead required to replicate the breadth of `PyCaret`'s offerings would be cost-prohibitive to many discipline-specific researchers.

Another challenge faced by many researchers is the need to tune the hyper-parameters of many of these ML models, which can be computationally and manually time intensive. `PyCaret` provides an efficient way of tuning and training ML models using a scoring grid based on ten fold cross-validation, where the best model is selected based on some accuracy metric supplied by the user, with the default being RMSE for regression and percent correctly classified (PCC) for classification. Several other `R` packages have implemented ML methods and automated hyper-parameter tuning including the meta-packages referenced previously, along with `EZtune` (Lundell, 2017). While `R` has a large and robust software library of ML approaches, no single package is as comprehensive in offerings as the `scikit-learn` library in `Python` (Pedregosa et al., 2011), as employed in `PyCaret` and made available in R through the `stressor` package.

The best way to access the power of the `scikit-learn` library is to learn how to use it directly within `Python`. However, it is difficult for many discipline-specific researchers to master multiple languages while simultaneously making advances within their discipline. The `stressor` package is designed to give `R` users access to the ML innovations in `Python` without having to know `Python` programming. The `stressor` package allows the users to use models tuned and trained using `Python`'s `PyCaret` library as if they were created in `R`. This provides `R` users quick and convenient ML benchmarks they can seamlessly incorporate into their `R` workflows. The `stressor` package is not intended to replace existing machine learning workflows in `R`. Rather, the `stressor` package is designed to supplement existing workflows with fast and easy comparisons to other ML methods with little programming overhead. The `mlr3`, `tidymodels`, and `caret` packages in `R` provide powerful machine learning workflows (tune, train, and predict) with more flexibility than is provided by the `stressor` package. However, these frameworks require more programming expertise than `stressor` requires, and the time it takes to implement them may be a deterrent to their use for many applied researchers.

In addition to providing `R` users access to `PyCaret`, the `stressor` package also allows users to validate their models with various forms of cross-validation (CV), including spatial CV (i.e. SCV) (see section 3.1 for details). The `stressor` package also includes `R` functionality for data generation, which is useful for testing the loss in accuracy that occurs when a ML model is employed on a dataset with a relatively simple underlying structure.

Figure 1 summarizes the main functions of `stressor` that will be described in the remainder of this paper. We first explain the general programming philosophy of `stressor` along with how to use `stressor` to obtain ML benchmarks from `PyCaret`, this is followed by a summary of the cross-validation variants available in stressor along with the package's data generation functionality. The paper concludes with two real data application examples of the `stressor` package on agricultural datasets.

2 Package Overview

Consider a typical data analysis workflow as partitioned into three general steps, namely (1) data preparation, (2) model creation/validation and (3) results visualization/presentation. The `stressor` package is designed leverage `R`'s advantages in data manipulation and visualization (i.e. Steps 1 and 3) in `R` while capitalizing on `Python`'s offerings in ML model development via

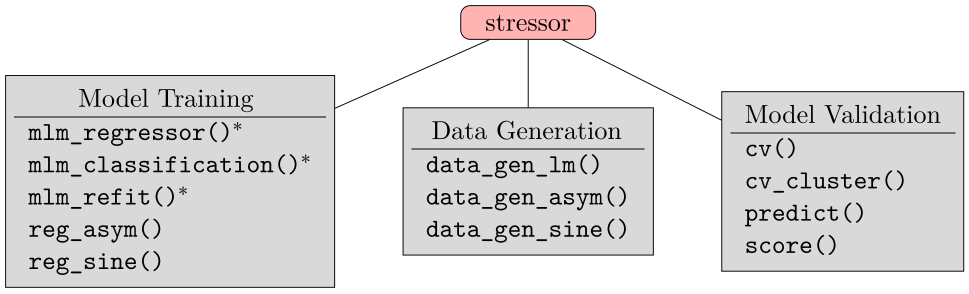

Figure 1: Diagram indicating main operations and utility functions of `stressor`. Asterisks indicate functions that are wrappers to `Python` functions.

the `PyCaret` library (Ali, 2020) (Step 2). The `stressor` package provides functionality for Step 2 in ways that works seamlessly with the typical R user's approach to Steps 1 and 3, as outlined in the following list:

1. data preparation (`R`)
2. model creation/validation
   (a) tune (`Python`)
   (b) train (`Python`)
   (c) validate (`Python`)
   (d) stress (`R`)
3. results visualization/presentation (`R`).

The "stress" sub-step is an optional step that involves additional model validation techniques intended to assess the performance of ML models in less than ideal conditions, such as the use of spatial cross-validation (SCV) explained further in Section 3.1.

2.1 Illustrative Example

One pre-requisite to the use of `stressor`'s `PyCaret` interface is an installed version of `Python` on the local machine or server. Currently, the creators of `PyCaret` recommend the using `Python` version 3.8.10. Once the user has installed `Python` and (possibly) added `Python` to their searchable path, the user can execute the `R` command

```
R> create_virtualenv()
```

which initializes a virtual `Python` environment, named based on the timestamp of creation, and installs `PyCaret` to this environment. This function prevents the user from needing to understand the details about how to properly initiate and format virtual environment with the necessary `Python` packages. To avoid cluttering a user's computer with multiple virtual environments, the function always tries to reuse a previous virtual environment created by `stressor` when it is available. The `create_virtualenv()` operates by calling functions from `R`'s `reticulate` package (Ushey et al., 2022). Note that, in some instances, the user may be required to restart their `R` session and point their project to the correct user environment using the command:

```
R> reticulate::use_virtualenv("<environment_name>")
```

Once the virtual environment has been created and initialized, the user can go from prepared dataset to ML benchmark in a few easy function calls. This is illustrated using the using the `boston` housing dataset (Harrison and Rubinfeld, 1978), made available in the `stressor` package, as adapted from the corrected version available in the `mlbench` package (Leisch and Dimitriadou, 2021), which omits all spatial and race-based variables. This dataset contains median house prices in 506 census tracts with 12 candidate explanatory variables. Information about the variables can be obtained running:

```
R> ?boston
```

We first withhold 20% of the observations for post-benchmark validation using `stressor`'s `split_data_prob()` function.

```
R> set.seed(90211)
R> ind <- split_data_prob(boston, 0.2)
R> train <- boston[!ind, ]
R> test <- boston[ind, ]
```

Then, using the training data, we call `PyCaret`'s tuning and training workflow with

```
R> mlm_boston <- mlm_regressor(formula = cmedv ~ ., data = train,
R>                             seed = 123)
```

which executes the following steps, all in `PyCaret`:

- Partition the training data into a tuning and validation dataset.
- Perform hyper-parameter tuning on the tuning data via a `RandomGridSearch` from `scikit-learn` (Pedregosa et al., 2011). A random grid search explores the same space of parameters as an exhaustive search (i.e. all possible combinations of hyper-parameters), but only randomly samples over the distribution of possible parameter values. Random grid search likely performs worse than an exhaustive search (Pedregosa et al., 2013). However, multiple runs over the same search space will generally result in the better model being returned than simply using model defaults.
- Score the models on several accuracy metrics using the validation data based on a user–defined criteria, which is RMSE by default for regression problems. The command
  ```
  R> mlm_boston$pred_accuracy
  ```
  will generate a table of all models and their performance across all accuracy metrics. Table 2 shows the accuracy metrics provided by `PyCaret` for both classification and regression problems.

The seed argument sets a random number seed in Python, which is essential for reproducibility since `PyCaret` does not recognize `R`'s random number stream. The user also has the option to set two additional arguments, though the authors generally recommend sticking with the defaults in each case:

- `fit_models`: A character vector of model tokens (see Table 1) to be included in the model fitting.
- `n_models`: The number of models returned in the final scoring table. If `n_models` is less than the number of models fit by `PyCaret`, then only the top `n_models` will be returned based on the primary accuracy metric selected.

For users only interested in a quick ML benchmark to compare against their manually created model, this step may complete the `stressor` portion of their project workflow. Some users may elect to perform the benchmark using the full dataset, rather than partitioning the data beforehand as is done in this example. That in mind, for users interested in further validation,

Table 2: Summary of default accuracy metrics available in PyCaret.

Regression		**Classification**	
Token	**Name**	**Token**	**Name**
mae	Mean Absolute Error	acc	Accuracy
mse	Mean Squared Error	auc	Area Under ROC Curve
rmse	Root Mean Squared Error	recall	Recall
r2	R^2	precision	Precision
rmsle	Root Mean Squared Log Error	f1	F1
mape	Mean Absolute Percentage Error	kappa	Kappa
		mcc	Mathews Correlation Coefficient

stressor facilitates a re-training of the PyCaret models using all available data (retaining the hyper-parameters selected in the previous step) and predicting the withheld test data with the command:

```
R> test_pred <- mlm_refit(mlm_object = mlm_boston, train_data = train,
R>                        test_data = test)
```

which returns a matrix of predictions, one column per method, on the test dataset using the models re-trained using the full training data (i.e. without the internal partition for hyper-parameter tuning and scoring). Users can then re-score their models in R using stressor's score function, i.e.,

```
R> results <- score(test_pred, test$cmedv, metrics = "RMSE")
R> head(results)
#     models     RMSE
# 1       et 3.092555
# 2      gbr 3.466087
# 3       rf 3.859793
# 4 lightgbm 3.686077
# 5      ada 4.518261
# 6       dt 4.552001
```

which shows the top performing models on the test set in descending order.

2.2 General Workflow

The stressor workflow is designed to keep data manipulation confined to R as much as is feasible. The only data manipulation step outsourced to PyCaret is the partitioning prior to hyper-parameter tuning. Figure 2 demonstrates the details of the stressor workflow. Red rounded-boxes in the figure are the functions that are called by the user, while blue boxes represent the Python functions called to execute each task. Tan boxes represent the R objects, often derived from Python objects as facilitated by the reticulate package.

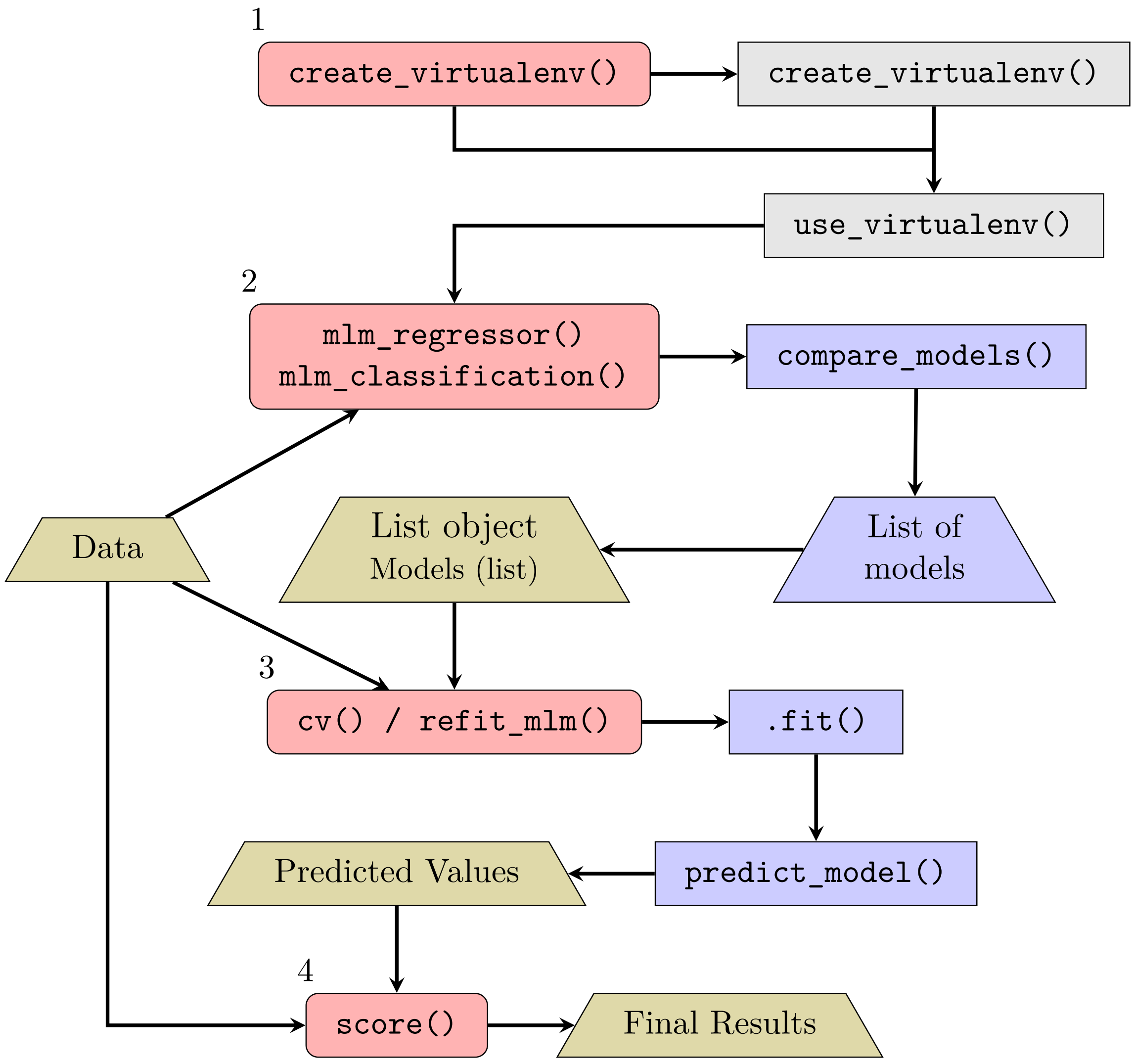

Figure 2: A flowchart demonstrating the model comparison workflow of the `stressor` package. Red squares indicate functions called by the user in `R`, blue squares indicate `Python` functions/objects, gray squares indicate functions from the package `reticulate`, and tan squares indicate `R` objects. A summary of steps include (1) create a virtual environment, (2) call fit and score the models, (3 - optional) perform cross-validation (CV), and (4 - optional) re-score all the models based on the CV results.

3 Additional Package Features

The primary purpose of the `stressor` package is to provide `R` users easy access to `PyCaret` in order to provide quick ML benchmarks to supplement a practitioners existing data analysis workflows. However, researchers often want to use their models – not always appropriately – to make predictions beyond the bounds of the observed data (i.e. extrapolation). Measuring predictive capability in such scenarios requires adaptations of traditional accuracy metrics to determine how the model performs “under stress”. The `stressor` package also provides `R`-specific cross-validation functions that allow `R` users to "stress-test" the trained machine learning models beyond what is provided in `PyCaret`. Additionally, the `stressor` package also provides some data generation functions that generate simple datasets with known variable relationships. This allows users to explore the penalty associated with deploying a complicated ML model on relatively simple data. These package features are described in the following sub-sections.

3.1 Cross-Validation

The `stressor` package also allows users to easily test their models predictive capabilities using several variants of CV and/or constrained sample sizes. For example, users could re-score all ML models using CV using two lines of code:

```
R> cv_models <- cv(mlm_boston, data = boston, n_folds = 10)
R> head(score(boston$cmedv, cv_models, metrics = "RMSE"))
#     models     RMSE
# 1       et 3.049584
# 2      gbr 2.878879
# 3       rf 3.225423
# 4 lightgbm 3.327633
# 5      ada 3.590602
# 6       dt 4.735953
```

At this point, we caution the reader on the potentially problematic use of cross-validation on the same data used to tune the models. This argument is explained succinctly in Neunhoeffer and Sternberg (2019) with additional discussion in Hastie et al. (2009). However, the practical experience of the authors suggest this problem usually only occurs on heavily tuned ML models and its consequences can be avoided by using multiple model validation paradigms to confirm model performance. This includes using variants of traditional CV that better quantify a ML model's ability to make predictions for new (and yet unseen) explanatory variable combinations via spatial cross-validation (SCV).

SCV is similar to traditional tenfold CV, with the primary difference being that the data are partitioned into spatially disjoint sets versus a random partition of the data to reduce the influence of spatial-autocorrelation (Brenning et al., 2012; Brenning, 2012; Lovelace et al., 2019). If ignored, the autocorrelation may result in underestimates of the true RMSE (Wadoux et al., 2021; Hengl et al., 2021; Ploton et al., 2020). There are several R and Python packages that implement variants of SCV (Schratz and Becker, 2023). However, these methods are focused on partitioning data based on space and time variables and use methodology that does not guarantee balance in the resulting cross-validation groups. The `stressor` package implements an adaptation of SCV that can be generalized to any set of variables (not just space and time) while also preserving approximate balance in the resulting k-folds, or groups. The approach is implemented using the following algorithm:

- Scale the X variables.
- Perform k-means clustering on the scaled variables, creating $k_{mult} * k$, sub-clusters that will be combined into k roughly equal sized groups for cross-validation. The variable k_{mult} is a multiplier with a recommended value of 5-10. This range for k_{mult} ensures the creation of enough sub-groups to ensure balanced observations in the k-fold groups during the consolidation step, while still ensuring sufficient sample sizes within each sub-group.

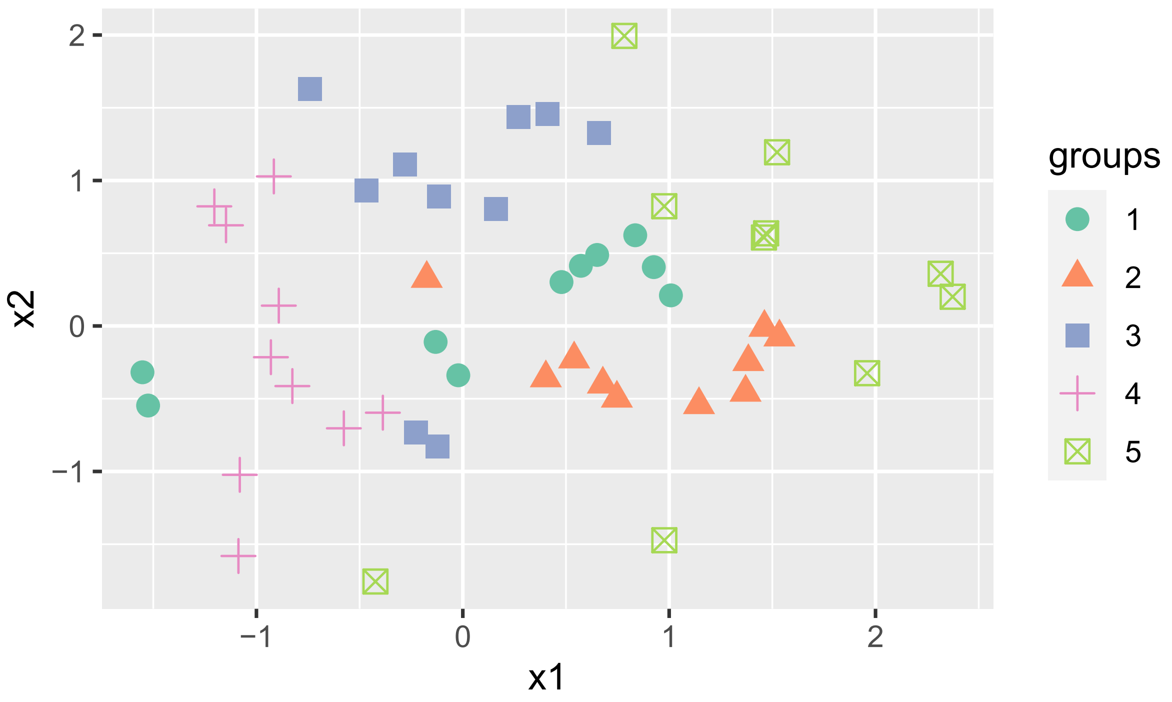

Figure 3: Illustration of two dimensional grouping with respect to spatial grouping. Note that multiple groups are spread across each dimension.

- Consolidate the k_{mult} groups into k-folds for cross-validation. This is done by ordering the size of each cluster and merging clusters in order of their size based on existing cluster centroids.
- Once a group gets too large, prevent smaller groups from joining it.
- Continue consolidation until the k groups are obtained.

Figure 3 shows an example of what two dimensional grouping looks like using our SCV approach. The grouping is accomplished via a `grouping_formula` argument in the `cv()` function, with constraints that ensure that the final k groups contain roughly equal numbers of observations. If the `grouping_formula` argument is NULL, then SCV will use all explanatory variables in the original trained formula of the object. The focus on group size balancing in this algorithm inevitably leaves one or two of the k groups that seem scattered because they have consolidated the leftover k_{mult} groups. This is intentional, as other methods (Le et al., 2019; Wang et al., 2019) for k-means clustering that impose approximate group balance within the k-means clustering step are more computationally intensive and lack widespread implementation.

We note that the motivations for SCV have only been rigorously explored for spatio-temporal variables in the literature. However, we argue that the use of a generalized SCV often provides a better measure of how a model will be used in practice: which is making a prediction on a new and yet unseen combination of x-variables.

To illustrate the use of SCV, we adapt the previous CV example with the `boston` housing dataset as follows:

```
R> cv_models_2 <- cv(mlm_boston, data = boston, n_folds = 10, k_mult = 5)
R> head(score(boston$cmedv, cv_models_2, metrics = "RMSE"))
#     models     RMSE
# 1       et 3.632840
# 2      gbr 3.564639
# 3       rf 3.793448
# 4 lightgbm 3.772861
# 5      ada 4.200270
# 6       dt 4.751349
```

The SCV routine allows users a fast way to determine the sensitivity of their accuracy results to changes in the method for validating accuracy. For example, if the accuracy metric for a particular ML varies widely when using train/test data, CV, and SCV, the user will know to be skeptical of using that ML model in deployment due to its inconsistent performance.

Other exploratory forms of model stress-testing, such as methods to iteratively reduce the training data size and explore the degradation in accuracy across iterations, are also included in the `stressor` package. However, these methods are only exploratory and not as well vetted as the SCV cross-validation variant described previously. Details regarding these exploratory methods are outlined in Haycock (2023), but their demonstration of use remains a topic of future work.

3.2 Data Generation

Another feature of the `stressor` package is the ability to generate datasets with known relationships between the explanatory variables and the response variable. These synthetic data sets can then be used for testing the ability of new functions or algorithms to recover known solutions in simple data cases. The `stressor` package contains methods for generating data from linear, asymptotic, and sinusoidal data generating models relating a response variable $\boldsymbol{y}$ to a matrix $\boldsymbol{X}$ of explanatory variables represented mathematically as

$$\boldsymbol{y} = \beta_0 + \beta_1 \boldsymbol{x_1} + \cdots + \beta_p \boldsymbol{x_p} + \boldsymbol{\epsilon}, \tag{1}$$

$$\boldsymbol{y} = \beta_0 - \alpha_1 e^{-\beta_1 \boldsymbol{x_1}} - \cdots - \alpha_p e^{-\beta_p \boldsymbol{x_p}} + \boldsymbol{\epsilon}, \tag{2}$$

$$\boldsymbol{y} = \beta_0 + \alpha_1 \sin(\beta_1(\boldsymbol{x_1} - \gamma_1)) + \cdots + \alpha_p \sin(\beta_p(\boldsymbol{x_p} - \gamma_p)) + \boldsymbol{\epsilon}, \tag{3}$$

where $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\boldsymbol{\gamma}$ represent model coefficients with residuals $\boldsymbol{\epsilon} \sim N(0, \sigma_\epsilon^2)$ where σ_ϵ is user-defined. Note that this linear dataset generation function does not allow for the specification of interaction terms. Note also that the individual X_i's are sampled from a standard normal distribution with the exception of Equation 2, which samples the X_i's from a uniform distribution. This can be executed by the `data_gen` function within `stressor` with arguments:

- `n` – an integer with the number of observations for the data.frame,
- `weight_vec/weight_mat` – a vector or matrix specifying the coefficients of the parameters,
- `y_int` – a numeric specifying the intercept along the y-axis, and
- `resp_sd` – a numeric specifying ϵ_σ.

The functions `reg_asym()` and `reg_sine()` are functions that use the `optim` function in order to quickly calculate the performance of the "ideal" model fit for the created datasets. Haycock (2023) shows the ability of `reg_asym()` to approximate the true asymptotic model coefficients with good-enough initial parameter guesses, but a consistent inability to approximate true model coefficients using `reg_sine()`. The intent of these functions, along with `R`'s `lm()` function for linear data, is to determine the penalty associated with using an ML model to when the true model form is known. To illustrate this, generate a dataset with 5000 observations from a data generating model derived from Equation 2 with $\alpha_1 = \cdots = \alpha_5 = 5$, $\beta_1 = \cdots = \beta_5 = 1$, $\beta_0 = 0$, and $\sigma_\epsilon = 1$.

```
R> set.seed(70157)
R> tmat <- matrix(c(rep(5, 5), rep(1, 5)), nrow = 2,
R>                byrow = TRUE)
R> data_1 <- data_gen_asym(5000, weight_mat = tmat,
R>                        resp_sd = 1)
```

We then split the data into a 80/20 training/test set and fit the true model using the true coefficients as our initial guess (including an initial guess of zero for β_0).

```
R> tind <- split_data_prob(data_1, 0.2)
R> train_1 <- data_1[!tind, ]
R> test_1 <- data_1[tind, ]
R> model_1 <- reg_asym(Y ~ ., train_1,
R>                    init_guess = c(as.vector(tmat), 0))
R> score(test_1$Y, predict(model_1, test_1), metrics = "RMSE")
# 0.9902157
```

This validates the ability of the model, when given a proper initial guess, to recover the true model error, which in this case was one. This validation serves as a best case baseline for any other models fit to these data. We then compare this with the model results obtained from `PyCaret` for this same dataset.

```
R> mlm_asy <- mlm_regressor(Y ~ ., train_data = train_1, seed = 457)
R> asy_pred <- mlm_refit(mlm_asy, train_1, test_1)
R> score(test_1$Y, asy_pred, metrics = "RMSE")
#  models     RMSE
# 1       gbr 1.033126
# 2  lightgbm 1.053717
# 3        rf 1.087795
# 4        et 1.085792
#        ...
# 15     llar 2.070206
# 16      omp 2.268977
# 17    dummy 2.378251
# 18      par 4.492784
```

These few lines of code revealed the ability of GBM and several tree-based methods to well approximate the true model error, while several of the linear-based estimators struggled in comparison. This process could be quickly repeated for different coefficients and different dataset sizes to get a sense of the sample sizes and data types required for each ML model to be successful. It is also a useful way to identify ML models that are over-fitting the data in situations where the ML model reports an accuracy value substantially less than the true model error.

4 Real Data Examples

Over half of the land in the Conterminous United States (CONUS) is devoted to the production of food, fuel, and fiber (USDA, 2019). Though biophysical factors (i.e. sun, soil, and water) strongly influence these cultivation geographies (Burchfield and Nelson, 2021; Liang et al., 2017; Ray et al., 2015), centuries of human activity have pushed cultivation geographies far beyond what landscapes can naturally support. For example, over the last century, technological innovation and political-economic incentives have driven a tripling of soy and wheat yields and a five-fold increase in corn yields (USDA, 2019). Recent efforts have focused on using ML modeling to quantify the predictive power of these non-biophysical factors, including landscape diversity metrics in predicting crop yields (Spangler et al., 2022; Schumacher et al., 2023).

One particularly important crop for farming in CONUS is corn. Urban et al. (2015) states that the demand for corn is increasing, with the US as the world's largest producer. Roberts et al. (2013) mentions that the changing climate will have drastic effects on crop yield. Understanding what factors are most predictive of current corn yields is critical for determining how yields may be altered, and/or how farmers will need to adapt in a changing climate.

Predicting crop yield for the past decades has relied heavily on classic econometric models (Roberts et al., 2013), biophysical crop models (Rosenzweig et al., 2013), process-based models (Soltani, 2012), and statistical models (Srivastava et al., 2014). Recently, there has been a drive to incorporate ML models into predicting crop yield (van Klompenburg et al., 2020). Studies have found that when ML models are combined with previous statistical models, predictive capability is increased (Crane-Droesch, 2018; Roberts et al., 2017a,b).

Three examples of using ML models for agricultural data include one use of RF to predict crop yields (Schumacher et al., 2023) and two examples of agricultural diversity (Spangler et al., 2022; Goslee, 2020). RF has been a popular method for agricultural problems due to its high accuracy results without much hyper-parameter tuning. For many researchers, the need to tune hyper-parameters for ML models becomes a barrier to their use. This section demonstrates how the ML workflow can be greatly simplified for `R` users through a demonstration of one regression and one classification example of two agricultural datasets.

4.1 Classification Example: Crop Suitability

Understanding what factors influence the suitability of a parcel of land for growing a crop is critical for determining how suitability might shift in a changing climate. An example of this kind of analysis is found in Burchfield (2022). The data set detailed in that reference consists of variables of the following general categories:

- climate - includes variables such as various measures of temperature (i.e. different times of the year and conditions) and precipitation, at the 1 km grid for North America.
- soil - includes variables such as amount of clay, sand, gravel in the topsoil and the pH of the soil, at the 1km scale.
- irrigation - includes the percentage of time the land has been irrigated from the United States Department of Agriculture (USDA's) Census of Agriculture (USDA, 2019) years of 2002, 2007, 2012, and 2017, also at the 1km pixel scale.
- topography - includes variables such as slope and elevation, data from the 1km and 30m resolution.
- Presence/Absence - the presence of the following crops: corn, winter wheat, soybeans, cotton, spring wheat, alfalfa, and hay. This is reported as binary response if the 30m pixel contains two instances of that crop being planted in the years from 2008 to 2019.

For this classification problem, we are trying to predict the suitability of a crop for a certain plot of land, with presence being defined as the crop was planted in a given plot of land (represented by 30m pixels) more than twice from 2008 to 2019 as a binary response (one present,

Table 3: Table of the PCC results for CV and SCV for the `stressor` ML models.

Abbr.	Models	CV	SCV
rf	Random Forest Classifier	83.64	81.23
et	Extra Trees Classifier	83.17	80.49
lightgbm	Light Gradient Boosting Machine	82.60	80.55
gbc	Gradient Boosting Classifier	79.99	78.40
ada	AdaBoost Classifier	77.32	74.42
lda	Linear Discriminant Analysis	76.29	74.52
ridge	Ridge Classifier	76.28	74.51
dt	Decision Tree Classifier	75.90	70.72
knn	K Neighbors Classifier	75.73	71.98
lr	Logistic Regression	74.41	72.77
qda	Quadratic Discriminant Analysis	74.20	71.62
nb	Naive Bayes	71.99	70.59
svm	SVM - Linear Kernel	61.59	63.74
dummy	Dummy Classifier	48.20	38.65

zero absent). In the data, 2.5 million pixels were randomly sampled across the Midwest to East Coast, and then paired with soil and climate information available at the 1km scale. There are 56 candidate explanatory variables for model training. Climate and soil data was available for CONUS at the 1km scale. The code necessary to fit the models to the data and perform both CV and CV are as follows:

```
R> ml_ap <- mlm_classification(AP ~ ., data = apmc)
R> ml_cv <- cv(ml_ap, data = apmc, n_folds = 10)
R> ml_cv_accuracy <- score(apmc$AP, ml_cv, metrics = "Accuracy")
R> ml_scv <- cv(ml_ap, data = apmc, n_folds = 10, k_mult = 5)
ml_scv_accuracy <- score(apmc$AP, ml_scv, metrics = "Accuracy")
```

Table 3 shows the PCC for CV and SCV. We observe for all models except for `svm`, the PCC is reduced when we group for SCV on the predictor variables. We see the `rf`, `et`, and `lightgbm` models have a PCC greater than 80% for both CV and SCV. We also see that the null, or `dummy`, model had a PCC of 48% for CV and 38% for SCV. The table shows that tree-based methods and variations of GBM are more effective than other model forms for classifying crop suitability, with similar results for both traditional and spatial cross-validation.

4.2 Regression Example: Corn Yield

Beside crop suitability, another important factor to consider is the yield (i.e. amount of food produced per acre of land) of the crop. An example of the use of this dataset can be found in Schumacher et al. (2023). Most of the variables are the same as the classification data set, though all variables are aggregated to represent the average for the county. Additional variables include time, geographic location (latitude and longitude) and the Shannon Diversity Index

Table 4: The RMSE values for CV, SCV, SCV with group on latitude and longitude of the 18 ML models.

Abbr.	Models	CV	SCV	SCV lat/lon
et	Extra Trees Regressor	16.33	22.37	22.69
rf	Random Forest Regressor	17.78	23.82	24.56
lightgbm	Light Gradient Boosting Machine	18.41	23.41	24.18
gbr	Gradient Boosting Regressor	22.38	25.73	26.05
knn	K Neighbors Regressor	23.58	33.76	34.53
dt	Decision Tree Regressor	26.37	33.40	33.75
ada	AdaBoost Regressor	28.19	30.52	29.96
lr	Linear Regression	29.17	30.56	30.84
ridge	Ridge Regression	29.17	30.57	30.84
br	Bayesian Ridge	29.17	30.58	30.85
lar	Least Angle Regression	30.65	31.85	31.69
lasso	Lasso Regression	29.84	31.78	31.74
llar	Lasso Least Angle Regression	29.84	31.78	31.74
en	Elastic Net	30.91	32.63	32.58
huber	Huber Regressor	33.50	35.19	35.16
omp	Orthogonal Matching Pursuit	38.16	39.37	38.87
dummy	Dummy Regressor	39.38	40.15	39.69
par	Passive Aggressive Regressor	48.63	43.81	52.62

(SDI) (Aguilar et al., 2015). The corn yield variable is provided at the county level and all other variables are aggregated to the county scale. This data spans from 2008 to 2018 and we assume that each year is independent of the previous one.

The code for producing the results in Table 4 is nearly identical to the previous examples provided in this paper. The only difference is that SCV is run twice: one using all the variables, and again using only the geographical coordinates of the county centroids, which is more akin to other implementations of SCV.

```
R> mlm_yield <- mlm_regressor(Yield ~ ., corn_yield)
R> yield_cv <- cv(mlm_yield, corn_yield, n_folds = 10)
R> yield_scv <- cv(mlm_yield, corn_yield, n_folds = 10, k_mult = 5)
R> yield_scv_latlon <- cv(mlm_yield, corn_yield, n_folds = 10, k_mult = 5,
R>                         grouping_formula = ~ lat + lon)
```

Table 4 shows that the RMSE values for SCV are notably larger than for CV, illustrating the potential loss of accuracy that comes with predicting corn yields for a new county not already represented in the training data in a different year. The only variables relevant for the SCV seem to be the spatial ones (i.e. latitude and longitude). Figure 4 provides a visual representation of the results in Table 4 for twenty separate runs of cross-validation. We see in Figure 4 SCV and SCV_latlon have more variation in their RMSE, suggesting that estimates from SCV have a

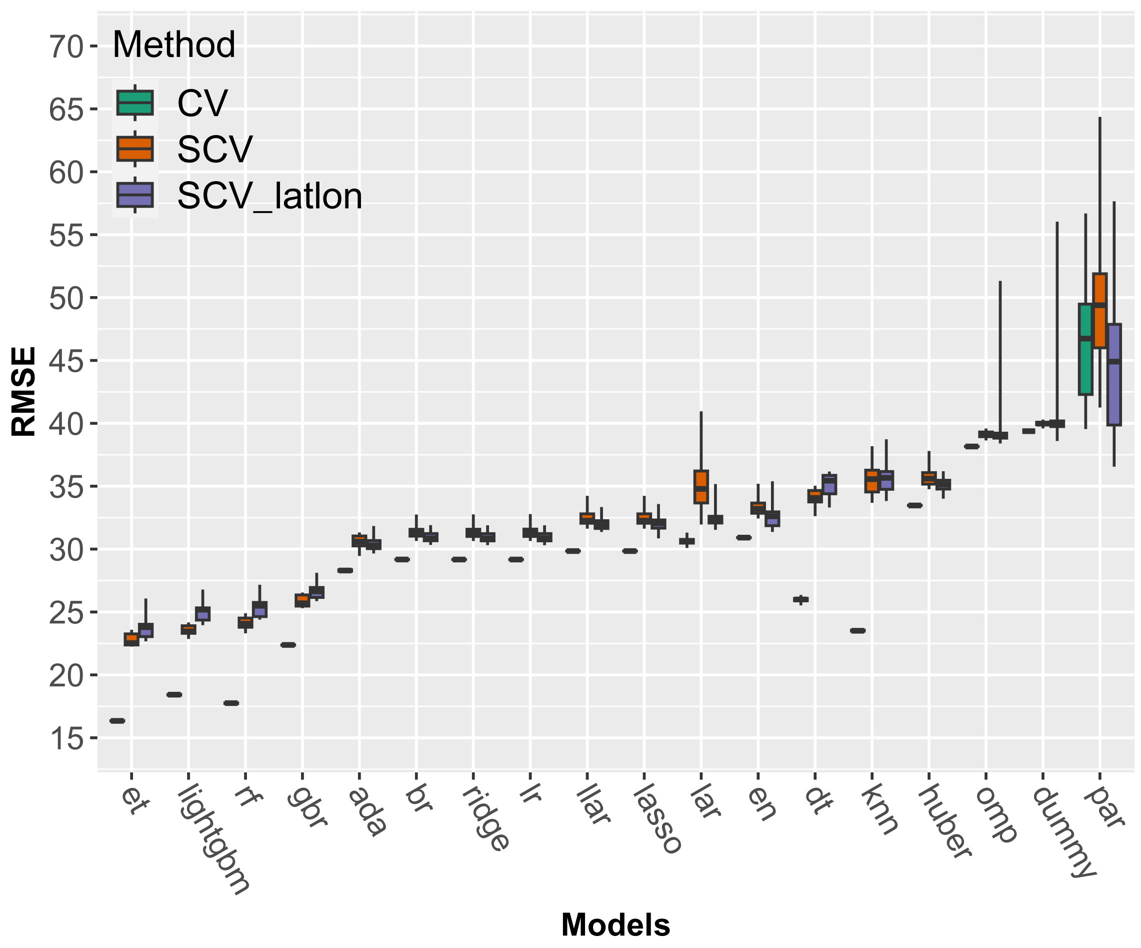

Figure 4: Side by side comparison of the three methods of CV to show that overall CV has better RMSE values than SCV and SCV with grouping. SCV groups are clustered first on all explanatory variables (SCV) and then again using only latitude and longitude to create the clusters (SCV lat/lon).

higher variance than estimates from CV. However, in this case, the best models using CV (which are, as before, tree-based methods and variations of GBM) are also the best models using SCV, which encourages their use.

This section has demonstrated the `stressor` package workflow on a real regression example. Note that the workflow is very similar to the classification example previously presented. With only a few lines of code, we can go from prepared data to accuracy results for 14-18 ML models, which is beneficial for discipline-specific researchers looking for a quick comparison against their preferred ML or analytical modeling approach.

5 Conclusion and Further Work

This paper has demonstrated the utility of the `stressor` by giving `R` users access to `Python` ML libraries without having to learn a new programming language. We have demonstrated the utility of the package on two different data examples, in each case showing the simplicity of the programming required to obtain results. We have also shown some of the additional features of the `stressor` package, including a novel implementation of SCV as well as data generation routines to explore the hypothetical performance of ML models on synthetic dataasets. Access to the development version of the `stressor` package can be found at:

https://github.com/beanb2/stressor

The continued use of the package on applied research problems will inevitably identify opportunities for package improvement. One example of a pending improvement involves improving the data generation methods by allowing for the ability to generate noise variables (i.e., explanatory variables with no relationship to the response variable) in the explanatory variable set to better test the sensitivity of the various ML models to extraneous variables. Finally, an opportunity exists to make the non default ML models available in `PyCaret` also available to the user of the `stressor` package.

There will always be more flexibility available to those who use `PyCaret` directly. However, for those who are not positioned to learn `Python` or who do not want to disrupt their `R` workflow when performing a simple ML benchmark, the `stressor` package provides a simple and convenient interface to `PyCaret` that requires very little programming or software overhead to implement. The `stressor` package has already proven to be a useful tool for benchmarking ML results (Schumacher et al., 2023), and its ongoing use reinforces the potential of the `stressor` package to improve access to `Python` ML models for current and future `R` users.

Supplementary Material

The supplementary materials associated with this paper contain all the data and code necessary to reproduce the figures and tables shown in this paper. Dataset descriptions have been provided in the text, but additional information about the files can be found in the README file contained in the supplementary materials folder.

References

Aguilar J, Gramig GG, Hendrickson JR, Archer DW, Forcella F, Liebig MA (2015). Crop species diversity changes in the United States: 1978–2012. *PLoS ONE*. 10(8): 1–4. https://doi.org/10.1371/journal.pone.0136580.

Ali M (2020). *PyCaret: An open source, low-code machine learning library in Python*. PyCaret version 1.0.0. https://www.pycaret.org (Accessed May 17, 2023).

Bowles M (2015). *Machine Learning in Python: Essential Techniques for Predictive Analysis*. John Wiley & Sons, Hoboken, NJ, USA.

Brenning A (2012). Spatial cross-validation and bootstrap for the assessment of prediction rules in remote sensing: The R package sperrorest. In: *2012 IEEE International Geoscience and Remote Sensing Symposium*, 5372–5375. IEEE. https://doi.org/10.1109/IGARSS.2012.6352393 (Accessed Dec 29, 2023).

Brenning A, Long S, Fieguth P (2012). Detecting rock glacier flow structures using Gabor filters and ikonos imagery. *Remote Sensing of Environment*, 125: 227–237. https://doi.org/10.1016/j.rse.2012.07.005.

Burchfield EK (2022). Shifting cultivation geographies in the central and eastern US. *Environmental Research Letters*, 17(5): 1–11. https://doi.org/10.1088/1748-9326/ac6c3d.

Burchfield EK, Nelson KS (2021). Agricultural yield geographies in the United States. *Environmental Research Letters*, 16(5): 1–12. https://doi.org/10.1088/1748-9326/abe88d.

Chen T, He T, Benesty M, Khotilovich V, Tang Y, Cho H, et al. (2023). *xgboost: Extreme Gradient Boosting*. R package version 1.7.6.1. https://CRAN.R-project.org/package=xgboost.

Crane-Droesch A (2018). Machine learning methods for crop yield prediction and climate change impact assessment in agriculture. *Environmental Research Letters*, 13(11): 114003. https://doi.org/10.1088/1748-9326/aae159.

Culp M, Johnson K, Michailidis G (2016). *ada: The R Package Ada for Stochastic Boosting*. R package version 2.0-5. https://CRAN.R-project.org/package=ada (Accessed May 17, 2023).

Friedman J, Hastie T, Tibshirani R (2010). Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1): 1–22. https://doi.org/10.18637/jss.v033.i01.

Goslee SC (2020). Drivers of agricultural diversity in the contiguous United States. *Frontiers in Sustainable Food Systems*, 4: 75. https://doi.org/10.3389/fsufs.2020.00075.

Gramacy RB (2007). tgp: An R package for Bayesian nonstationary, semiparametric nonlinear regression and design by treed Gaussian process models. *Journal of Statistical Software*, 19: 1–46. https://doi.org/10.18637/jss.v019.i09.

Greenwell B, Boehmke B, Cunningham J, GBM Developers (2022). *gbm: Generalized Boosted Regression Models*. R package version 2.1.8.1. https://CRAN.R-project.org/package=gbm.

Harrison D, Rubinfeld DL (1978). Hedonic housing prices and the demand for clean air. *Journal of Environmental Economics and Management*, 5(1): 81–102. https://doi.org/10.1016/0095-0696(78)90006-2.

Hastie T, Efron B (2022). *lars: Least Angle Regression, Lasso and Forward Stagewise*. R package version 1.3. https://CRAN.R-project.org/package=lars.

Hastie T, Tibshirani R, Friedman JH, Friedman JH (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, volume 2. Springer, New York, NY USA.

Haycock S (2023). stressor: An R package for benchmarking machine learning models. *Utah State University Digital Commons*. 1-75. https://doi.org/10.26076/2am5-9f67.

Hengl T, Miller MA, Križan J, Shepherd KD, Sila A, Kilibarda M, et al. (2021). African soil properties and nutrients mapped at 30 m spatial resolution using two-scale ensemble machine learning. *Scientific Reports*, 11(1): 1–18. https://doi.org/10.1038/s41598-021-85639-y.

Hothorn T (2023). CRAN task view: Machine learning & statistical learning. Version 2023-07-20. https://CRAN.R-project.org/view=MachineLearning.

Ke G, Meng Q, Finley T, Wang T, Chen W, Ma W, et al. (2017). Lightgbm: A highly efficient gradient boosting decision tree. In: *Advances in Neural Information Processing Systems*. volume 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf (Accessed May 17, 2023).

Krueger T, Braun M (2022). *CVST: Fast Cross-Validation via Sequential Testing*. R package version 0.2-3. https://CRAN.R-project.org/package=CVST.

Kuhn M (2022). *caret: Classification and Regression Training*. R package version 6.0-93. https://CRAN.R-project.org/package=caret.

Kuhn M, Wickham H (2020). *Tidymodels: a collection of packages for modeling and machine learning using tidyverse principles*. https://www.tidymodels.org.

Lang M, Binder M, Richter J, Schratz P, Pfisterer F, Coors S, et al. (2019). mlr3: A modern object-oriented machine learning framework in R. *Journal of Open Source Software*. 1903. https://doi.org/10.21105/joss.01903.

Le HM, Eriksson A, Do TT, Milford M (2019). A binary optimization approach for constrained k-means clustering. In: *Computer Vision–ACCV 2018: 14th Asian Conference on Computer Vision, Revised Selected Papers, Part IV*. Perth, Australia. December 2–6, 2018, 383–398. Springer. https://doi.org/10.1007/978-3-030-20870-7_24.

Leisch F, Dimitriadou E (2021). *mlbench: Machine Learning Benchmark Problems*. R package version 2.1-3.1. https://cran.r-project.org/package=mlbench.

Liang XZ, Wu Y, Chambers RG, Schmoldt DL, Gao W, Liu C, et al. (2017). Determining climate effects on US total agricultural productivity. *Proceedings of the National Academy of Sciences*, 114(12): E2285–E2292. https://doi.org/10.1073/pnas.1615922114.

Lovelace R, Nowosad J, Muenchow J (2019). *Geocomputation with R*. CRC Press. https://r.geocompx.org/spatial-cv.html (Accessed: Dec 29, 2023).

Lundell JF (2017). There has to be an easier way: A simple alternative for parameter tuning of supervised learning methods. In: *JSM Proceedings, Statistical Computing Section*, 3028–3036. American Statistical Association, Alexandria, VA. https://cran.r-project.org/package=EZtune.

Majka M (2019). *naivebayes: High Performance Implementation of the Naive Bayes Algorithm in R*. R package version 0.9.7. https://CRAN.R-project.org/package=naivebayes.

Meyer D, Dimitriadou E, Hornik K, Weingessel A, Leisch F (2022). *e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*. R package version 1.7-12. https://CRAN.R-project.org/package=e1071.

Neunhoeffer M, Sternberg S (2019). How cross-validation can go wrong and what to do about it. *Political Analysis*, 27(1): 101–106. https://doi.org/10.1017/pan.2018.39.

Papadakis M, Tsagris M, Fafalios S (2023). *Rfast: A Collection of Efficient and Extremely Fast R Functions*. R package version 2.1.0. https://CRAN.R-project.org/package=Rfast.

Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12: 2825–2830. https://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf?ref=https:/ (Accessed Dec 29, 2023).

Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. (2013). sklearn.model_selection.randomizedsearchcv. *Journal of Machine Learning Research*, 12: 2825–2830. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html (Accessed May 17, 2023).

Ploton P, Mortier F, Réjou-Méchain M, Barbier N, Picard N, Rossi V, et al. (2020). Spatial validation reveals poor predictive performance of large-scale ecological mapping models. *Nature Communications*, 11(1): 1–11. https://doi.org/10.1038/s41467-020-18321-y.

R Core Team (2022). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Raschka S, Patterson J, Nolet C (2020). Machine learning in Python: Main developments and technology trends in data science, machine learning, and artificial intelligence. *Information*, 11(4): 1–33. https://doi.org/10.3390/info11040193.

Ray DK, Gerber JS, MacDonald GK, West PC (2015). Climate variation explains a third of global crop yield variability. *Nature Communications*, 6(1). https://doi.org/10.1038/ncomms6989.

Roberts DR, Bahn V, Ciuti S, Boyce MS, Elith J, Guillera-Arroita G, et al. (2017a). Cross-validation strategies for data with temporal, spatial, hierarchical, or phylogenetic structure. *Ecography*, 40(8): 913–929. https://doi.org/10.1111/ecog.02881.

Roberts MJ, Braun NO, Sinclair TR, Lobell DB, Schlenker W (2017b). Comparing and combining process-based crop models and statistical models with some implications for climate change. *Environmental Research Letters*, 12(9): 095010. https://doi.org/10.1088/1748-9326/aa7f33.

Roberts MJ, Schlenker W, Eyer J (2013). Agronomic weather measures in econometric models of crop yield with implications for climate change. *American Journal of Agricultural Economics*, 95(2): 236–243. https://doi.org/10.1093/ajae/aas047.

Rong X (2022). *deepnet: Deep Learning Toolkit in R*. R package version 0.2.1. https://CRAN.R-project.org/package=deepnet.

Rosenzweig C, Jones JW, Hatfield JL, Ruane AC, Boote KJ, Thorburn P, et al. (2013). The agricultural model intercomparison and improvement project (agmip): Protocols and pilot studies. *Agricultural and Forest Meteorology*, 170: 166–182. https://doi.org/10.1016/j.agrformet.2012.09.011.

Schratz P, Becker M (2023). *mlr3spatiotempcv: Spatiotemporal Resampling Methods for 'mlr3'*. https://mlr3spatiotempcv.mlr-org.com/.

Schumacher BL, Burchfield EK, Bean B, Yost MA (2023). Leveraging important covariate groups for corn yield prediction. *Agriculture*, 13(3). https://doi.org/10.3390/agriculture13030618.

Soltani A (2012). *Modeling physiology of crop development, growth and yield*. CABi. https://www.cabidigitallibrary.org/doi/book/10.1079/9781845939700.0000.

Spangler K, Schumacher BL, Bean B, Burchfield EK (2022). Path dependencies in US agriculture: Regional factors of diversification. *Agriculture, Ecosystems & Environment*, 333: 107957. https://doi.org/10.1016/j.agee.2022.107957.

Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1): 1929–1958. https://dl.acm.org/doi/abs/10.5555/2627435.2670313.

Therneau T (2018). *deming: Deming, Theil-Sen, Passing-Bablock and Total Least Squares Regression*. R package version 1.4. https://CRAN.R-project.org/package=deming.

Therneau T, Atkinson B (2022). *rpart: Recursive Partitioning and Regression Trees*. R package version 4.1.19. https://CRAN.R-project.org/package=rpart.

Urban DW, Sheffield J, Lobell DB (2015). The impacts of future climate and carbon dioxide changes on the average and variability of us maize yields under two emission scenarios. *Environmental Research Letters*, 10(4): 045003. https://doi.org/10.1088/1748-9326/10/4/045003.

USDA (2019). 2017 Census of Agriculture. https://www.nass.usda.gov/AgCensus (Accessed Dec 29, 2023).

Ushey K, Allaire J, Tang Y (2022). *reticulate: Interface to Python*. R package version 1.25. https://CRAN.R-project.org/package=reticulate.

van Klompenburg T, Kassahun A, Catal C (2020). Crop yield prediction using machine learning: A systematic literature review. *Computers and Electronics in Agriculture*, 177: 105709. https://doi.org/10.1016/j.compag.2020.105709.

Venables WN, Ripley BD (2002). *Modern Applied Statistics with S*. Springer, New York, fourth edition. ISBN 0-387-95457-0.

Wadoux AMC, Heuvelink GB, De Bruin S, Brus DJ (2021). Spatial cross-validation is not the right way to evaluate map accuracy. *Ecological Modelling*, 457: 109692. https://doi.org/10.1016/j.ecolmodel.2021.109692.

Wang XD, Chen RC, Yan F, Zeng ZQ, Hong CQ (2019). Fast adaptive k-means subspace clustering for high-dimensional data. *IEEE Access*, 7: 42639–42651.

https://doi.org/10.1109/ACCESS.2019.2907043.

Wright MN, Ziegler A (2017). Ranger: A fast implementation of random forests for high dimensional data in C++ and R. *Journal of Statistical Software*, 77(1): 1–17. https://doi.org/10.18637/jss.v077.i01.