---

title: “Using the sdgm Package”
author: “Khaled El Emam”
date: “2023-10-03”
output: rmarkdown::pdf_document

This document describes the basic operations of the sdgm package. This package is intended to provide some basic modeling functions that can be used in other projects and within other packages.

Main Installation Steps

The following process should make it somewhat easier to install the package for the first time and also to update it over time. The installation is directly from Cloudsmith. The easiest way is to include the Cloudsmith repositories that are used by EHIL in your R Studio profile, and then you can just do a regular install afterwards.

Run the following to save the profile information:

```r
r = getOption("repos")
r["ehil"] = "https://dl.cloudsmith.io/oZuDP9AMif3uYKF7/ehil/sdg/cran/"
options(repos = r)
rm(r)
```

Then either through the R Studio menu or the following code you can install this package:

```r
install.packages("sdgm")
```

If you do not want to save this Cloudsmith repository in your profile, then you can install directly as follows:

```r
install.packages(
  "sdgm",
  repos = c(cloudsmith = "https://dl.cloudsmith.io/oZuDP9AMif3uYKF7/ehil/sdg/cran/")
)
```

While these links are open, they are only accessible through this unique URL. Therefore do not share the link outside the EHIL team / teaching as these packages are not intended for dissemination outside our group.

Basic Principles

The `sdgm` package was designed to make correct machine learning (ML) modeling easier. There are other packages that do this but we have found them not to be thoroughly documented with many methodology and practical details not clear, leavign the user having to experiment a lot or to readt he package code, neither of which is ideal.

The package is intended to be used in other larger ML projects and simulations. Our lab’s focus is synthetic data generation (hence the name `sdg`), however, the modeling tools can be used in other applications as well.

This section provides some of the basic principles for the `sdgm` package.

Debug Output

If you want to get debug messages (which can sometimes be useful), set the following at the top of your notebooks:

```
sdgm.verbose<<-TRUE
```

This is different than the `verbose` parameter which is used in function calls.

Binary Classification

Type of Outcome

For the binary classification models the outcome is expected to be a binary variable of some sort. If it is not a binary variable then an error will be generated. If you can make the binary variable 0/1 then everything is clear. If the outcome variable is a factor or a character, that is OK and the functions should handle that. But it should have two values only for binary classification.

For factors or character, the question is what is the positive class that the predicted probability pertains to. if you set the global verbose to TRUE the function will tell you what it determines to be the positive class, or you can look at the factor levels to see which is the second factor. But that is something important to keep in mind.

Modeling Process

The default modeing process uses stratified 5-fold cross-validation to tune the hyperparameters for the modeling method of interest. Once the optimal hyperparameters are determined then a model is trained on the full dataset. That is the final model that is delivered at the end of the model training.

Type of Prediction

The binary classifiers predict (using the `predict` function) probabilities or pseudo probabilities. They do not attempt to convert these to actual classes. These conversions are left to the user.

Prediction Result

A generic `predict` function is then used to get the predicted probabilities using the trained model. A new (test) dataset is typically used to predict on. Note that not all methods can handle missing values in the predictors in the test dataset. For methods that cannot handle missing values in the predictors during prediction, the predicted value will be `NA`. Therefore, keep in mind that sometimes you may get an `NA` value instead of a predicted probability, but the prediction result (the vector of probabilities) will always be the same length as the test dataset number of rows.

Hyperparameter Tuning

The tuning algorithm used is Bayesian optimization. This cannot be changed. If the BO algorithm fails for some reason (e.g., lack of variation), the hyperparameters used wll be the default ones. Whatever they are, the model hyperparameters are in the resultant model object.

Example

The following example shows a train/test for the CART model on the Adult census dataset (a random sample from that just for performance reasons).

```r
# this displays debug statements and outputs which is sometimes useful
# yu can switch it off if you do not want the debug output
sdgm.verbose<<-T

full_data<-sdgm::C1

# create train and test data
idx<-splitTools::partition(rep(0,nrow(full_data)), p=c(train=0.7, test=0.3), type="stratified")
train_data <- full_data[idx$train,] %>% dplyr::slice_sample(prop=0.1)
voutcome<-"income"
test_data <- full_data[idx$test,]

# train a model with optimal hyperparameters
best_model<-sdgm::cart.bestmodel.bin(train_data, voutcome)

# predict on the test data; this is a generic predict function
preds<-predict(best_model, test_data)

# # auc
options(warn = 1)
if (!is.null(preds))
{
  # we use our own AUC function as it fixes some bugs in the MLmetrics package version
  test_auc<-sdgm::auc(preds, test_data[,voutcome] )
} else {
  test_auc<-NA
  print("AUC calculation failed because there are no predicted values")
}

print(paste0("AUC on Adult Data: ", test_auc))
```

This template can be used for any of the binary classification functions. The binary classification functions are:

Modeling Methods	Function
Logistic Regression	sdgm::lr.bestmodel.bin
CART	sdgm::cart.bestmodel.bin
Random Forest	sdgm::rf.bestmodel.bin
SVM	sdgm::svm.bestmodel.bin
LGBM	sdgm::lgbm.bestmodel.bin

The parameters are exactly the same, and the behavior is the same in that a model is trained and tuned (where relevant), and the final model is returned that is trained on the full dataset.

The output of the modeling is an object that has a consistent structure.

```
print(best_model)
```

Attribute	Interpretation
model	Model object (which will depend on the algorithm used)
params	A list of the optimal hyperparameters. The actual hyperparameters will depend on the algorithm that is used. There is an additional attribute there which is `perf` and that is the logloss (multiplied by `-1` to make it a positive number) of the optimal hyperparameters.
outcome	A string of the variable name of the outcome
factorList	A list of the factor predictors, with the categories used during training (as character strings). Any categories that are not there during prediction will be removed (that observation converted to NA)
predictors	A string list of all of the predictor variables