Using the sdgm Package

Khaled El Emam, Dan Liu, and Juan Li

2024-01-12

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. The main purposes for developing this package were:

- Teaching an introductory course in machine learning
- Use it as part of machine learning research projects so that we would have a documented and stable set of modeling functions
- Perform experiments to evaluate different techniques to improve the performance of different types of models

Note that the models in this package can take a considerable amount of computation to complete. Therefore, do not be surprised if you need to wait if your computational capacity is limited. It is recommended to run this package on a machine that has many CPU cores to enable the parallelization of the computation. Also, this package is implemented to work on CPUs by default. It will not work on GPUs at this point in time.

Main Installation Steps

The package has a dependency on `keras` and hence `tensorflow`. These packages are needed if the embedding layer is trained for dealing with high cardinality variables (it is one of the categorical variable encoding options in the package).

It is not necessary to have a GPU available as the computations are all CPU-bound. However, these packages need to be installed first even if you do not need that functionality for your projects. This will install Python on the machine. The assumption is that the `reticulate` package is installed. If it is not the it will need to be installed first.

The installation of these packages can be accomplished as follows:

```r
# install reticulate if necessary
install.packages("reticulate")

## start here
reticulate::install_python()
install.packages("tensorflow")
library(tensorflow)
install_tensorflow(envname = "r-tensorflow")

## RESTART
```

```r
install.packages("keras")
library(keras)
install_keras()

## RESTART

## TO TEST IT
library(magrittr)
model <- keras_model_sequential()
model %>%
  layer_dense(units = 256, activation = 'relu', input_shape = c(784)) %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 10, activation = 'softmax')

# this should display a summary of the model specified above
summary(model)
```

After that you can install the `sdgm` package. 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 (assuming you use R Studio).

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 or you are not using R Studio, then you can install directly as follows and this will also install all of the dependencies (which is probably the easiest way to install the package and is my preference):

```r
install.packages(
  "sdgm",
  repos = c(
    cloudsmith = "https://dl.cloudsmith.io/oZuDP9AMif3uYKF7/ehil/sdg/cran/",
    cran = "http://cran.us.r-project.org"
    ),
  dependencies=T
)
```

While these links are open, they are only accessible through this unique URL.

Basic Principles of the Package

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, leaving the user having to experiment a lot or to read the package code to understand what is happening, neither of which is ideal. Also, for users that just want to train models, the flexibility of some of the other packages makes it easier to make subtle mistakes, which we hope we have avoided here.

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 modeling 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

The following binary classification functions are currently supported:

The binary classification functions are:

Modeling Methods	Notes
Logistic Regression	Logistic regression with no feature selection
CART	Standard Classification trees using the `rpart` package
Random Forest	Random forests using the `ranger` package
SVM	Support Vector Machines
LGBM	Light gradient boosted decision trees
xgboost	Extreme gradient boosted trees

More methods are being added so this is not a static list.

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 `sdgm.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 (and taht is the positive class). But that is something important to keep in mind.

Modeling Process

The default modeling 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.

The hyperparameter tuning is performed by default to maximize the AUC. This can be changed by the function parameters to `logloss` or `brier`.

Also, by default the model training will try to apply target encoding on high cardinality variables. For logistic regression this is attempted when a variable has more than ten categories (this is defined in the constant `sdgm::sdgm.s.encode.threshold`). For the other model types this is attempted when a variable has more than 100 categories (this is defined in the constant `sdgm::sdgm.l.encode.threshold`).

Even though encoding is attempted, that does not mean that it will be implemented as this becomes another hyperparameter that is included in tuning (whether to encode or not). Therefore, if the encoding does not improve the AUC (by default) then it will not be kept.

Encoding will only be implemented if the tuning is turned on. This is a general feature of the package in that some types of functionality are only considered in tuning (such as encoding and rebalancing).

The user can switch the encoding scheme to `embedding` where multidimensional embedding layer will be trained and used. The number of dimensions follows the Jeremy Howard / `fast.ai` rule of thumb which is the minimum of 50 dimensions or the number of categories divided by two (rounded).

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 will be the default ones. Whatever they are, the model hyperparameters are in the resultant model object and can be inspected. The values chosen were informed by the following analyses and recommendations:

1. E. Bartz, T. Bartz-Beielstein, M. Zaefferer, and O. Mersmann, Eds., Hyperparameter Tuning for Machine and Deep Learning with R: A Practical Guide. Singapore: Springer Nature, 2023. doi: 10.1007/978-981-19-5170-1.
2. B. Bischl et al., "Hyperparameter Optimization: Foundations, Algorithms, Best Practices and Open Challenges," arXiv.org. Accessed: Dec. 09, 2023. [Online]. Available: https://arxiv.org/abs/2107.05847v3

3. M. Binder, F. Pfisterer, and B. Bischl, "Collecting Empirical Data About Hyperparameters for Data Driven AutoML," 7th ICML Workshop Autom. Mach. Learn., 2020.

4. D. Kühn, P. Probst, J. Thomas, and B. Bischl, "Automatic Exploration of Machine Learning Experiments on OpenML," arXiv.org. Accessed: Dec. 09, 2023. [Online]. Available: https://arxiv.org/abs/1806.10961v3

CART

Hyperparameter	Default	Lower Bound	Upper Bound	Transform
`min.split`	4	1	7	value is round(2^`min.split`)
`min.bucket`	3	0	6	value is round(2^`min.bucket`)
`max.depth`	30	1	30	
`cp`	-2	-4	-1	value is 10^`cp`

SVM

Hyperparameter	Default	Lower Bound	Upper Bound	Transform
`kernel`	1	1 (radial)	2 (sigmoid)	
`gamma`	1/nFeatures	-10	10	2^`gamma`
`cost`	1	-10	10	2^`cost`

Random Forest

Hyperparameter	Default	Lower Bound	Upper Bound	Transform
`num.trees`	500	1	2000	
`min.node.size`	0.5	0	1	round(n^`min.node.size`) where n is the number of observations
`max.depth`	15	1	50	
`min.bucket`	10	1	60	

Other hyperparameters were not tuned because they were not improving the performance on the models that we tested on.

LGBM

Hyperparameter	Default	Lower Bound	Upper Bound	Transform
`booster`	1 (gbdt)	1 (gbdt)	2 (goss)	
`max_depth`	6	1	15	
`learning_rate`	log2(0.3)	-10	0	2^`learning_rate`
`early_stopping_rounds`	7	7	30	

Hyperparameter	Default	Lower Bound	Upper Bound	Transform
`min_data_in_leaf`	10	1	60	
`num_leaves`	15	4	60	

xgboost

Hyperparameter	Default	Lower Bound	Upper Bound	Transform
`gamma`	0	-15	3	`2^gamma`
`eta`	log2(0.3)	-10	0	`2^eta`
`max_depth`	6	1	15	
`early_stopping_rounds`	7	7	30	
`max_leaves`	15	4	60	
`min_child_weight`	1	0	7	`2^min_child_weight`

Example

The following example shows a basic train/validate/test for the CART model on the Adult census dataset (a random sample from that just for performance reasons). The train/validate is within the `cart.bestmodel.bin` function which performs a 5-fold cross-validation to find the best hyperparameters and returns the best hyperparameters and the model trained on the full dataset using these best hyperparameters. This type of function is the basic building block of the package.

```r
# this displays debug statements and outputs which is sometimes useful
# you 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$train,] %>% dplyr::slice_sample(prop=0.1) # take sample to speed it up
test_data <- full_data[idx$test,]
voutcome<-"income"

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

# predict on the test data
preds<-predict(best_model, test_data)

# auc
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))
```

Binary Classification Functions

The above template can be used for any of the binary classification functions when implementing a simple train/validate/test evaluation with 5-fold CV. 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
xgboost	sdgm::xgb.bestmodel.bin

The function 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. For model-specific behavior, please see the help files for the model.

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

```r
print(best_model)
```

The explanation of the attributes of the resultant model are:

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 loss (multiplied by -1 to make it a positive number if logloss) 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 may be removed (that observation converted to NA - depending on the algorithm).
predictors	A string list of all of the predictor variables prior to encoding
hicar	A string list of all high cardinality variables that may be considered for additional encoding. If this is NULL it means that no encoding has been performed for this model (i.e., no encoding was more favourable or tuning was off).
encoders	This is a list of encoders that were created for the high cardinality variables. It is for internal use.
encode_method	The specific method that was used for encoding. This method may not have actually been applied if the hicar attribute is NULL. This attribute reflects what the user wanted and not necessarily what was implemented.
cfit	Calibration fit if it was deemed that beta calibration improved the predicted probabilities.

Attribute	Interpretation
logicalList	A string list of field names that were logical in the original data. This is only used for some of the models where consistency between the logical status in fit for predict is important (such as CART).

Nested Cross-Validation

Nested cross-validation is a general purpose function that has a 5-fold outer loop. By default it tries to parallelize the outer loop but this can be disabled through the parameters.

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

full_data<-sdgm::C1 %>% dplyr::slice_sample(prop=0.1)

# create train and test data
voutcome<-"income"

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

# auc
model_auc<-best_model$auc

print(paste0("AUC on Adult Data: ", model_auc))

# there is a predict() function for the model trained from the nested CV
# so you can use the "best_model" on unseen data as well
```

For model evaluation it is generally recommended to used nested CV (see recommendations below).

Calibration

Calibration is defined, most commonly, as when we observe a p% probability of an outcome among patients with a predicted p% probability of the event [1]. For binary prediction with boosted trees, the predicted values are not true probabilities and the error grows as the number of iterations increases [2], [3]. Therefore, these pseudo probabilities need to be calibrated after the fact. Calibration methods, such as isotonic regression overfit with small datasets. Another commonly used method, Platt scaling, is also not suitable because boosting tends to produce extreme probability estimates (close to zero or one), and Platt scaling works best when the probabilities are closer to the mid-point [4]. Furthermore, the logistic function used in Platt scaling does not support the identity function, which can result in calibrated models having worse calibration after scaling. We use beta calibration, which provides a better solution with a more flexible functional form [5].

When oversampling methods are used to balance datasets, that results in incorrect predicted probabilities across different machine learning algorithms [6]. These predicted probabilities must be corrected using calibration, and in that case we also use beta calibration. The integrated calibration index (ICI) is used to determine whether beta calibration has improved the accuracy of the predicted probabilities [7]. This means that in some instances the calibration does not add value and the uncalibrated model is used.

The dataset is divided into three partitions: training, calibration, and test. The ML model is trained on the training partition using the optimal hyperparameters. A calibration model is fitted on the calibration dataset using the predicted and observed values from the trained model. The ICI is computed on the test dataset to determine if the calibration should be retained or not. If calibration is retained then the final model is trained on the concatenation of the training and test datasets. If calibration is not retained then the final model is trained on the full dataset. Note that the encoding partition is treated separately from the above (i.e., if encoding is performed then that partition is not included in the calibration).

For imbalanced data, rebalancing is performed on the training set only to avoid leakage of information in the test dataset [8]. We use a sequential decision tree generative model for rebalancing [9]. Once a synthetic dataset is generated, rejection sampling is used to add minority group records. This type of generative model has been used to synthesize health and social sciences data [10]–[18], and applied in research studies on synthetic data [10], [19], [20].

The rebalancing decision is made during hyperparameter tuning and is only retained if it improves on the chosen loss metric. This is how calibration is selected:

Method	Calibration Decision
Logistic Regression	if rebalancing is selected
Random Forest	always calibrate
CART	if rebalancing is selected
SVM	always calibrate
LGBM	always calibrate
xgboost	always calibrate

Note that the algorithm and approach used for rebalancing are experimental and will likely evolve over time as we make improvements.

References for Calibration Subsection

1. E. W. Steyerberg, Clinical Prediction Models: A Practical Approach to Development, Validation, and Updating. in Statistics for Biology and Health. Cham: Springer International Publishing, 2019. doi: 10.1007/978-3-030-16399-0.

2. A. Niculescu-Mizil and R. A. Caruana, “Obtaining Calibrated Probabilities from Boosting,” ArXiv12071403 Cs Stat, Jul. 2012, Accessed: Oct. 21, 2020. [Online]. Available: http://arxiv.org/abs/1207.1403

3. B. Zadrozny and C. Elkan, “Obtaining calibrated probability estimates from decision trees and naive Bayesian classifiers,” in ICML, 2001.

4. Y. Huang, W. Li, F. Macheret, R. A. Gabriel, and L. Ohno-Machado, “A tutorial on calibration measurements and calibration models for clinical prediction models,” J. Am. Med. Inform. Assoc. JAMIA, vol. 27, no. 4, pp. 621–633, Apr. 2020, doi: 10.1093/jamia/ocz228.

5. M. Kull, T. S. Filho, and P. Flach, “Beta calibration: a well-founded and easily implemented improvement on logistic calibration for binary classifiers,” in Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, PMLR, Apr. 2017, pp. 623–631. Accessed: Dec. 30, 2022. [Online]. Available: https://proceedings.mlr.press/v54/kull17a.html

6. R. van den Goorbergh, M. van Smeden, D. Timmerman, and B. Van Calster, “The harm of class imbalance corrections for risk prediction models: illustration and simulation using logistic regression,” J. Am. Med. Inform. Assoc. JAMIA, vol. 29, no. 9, pp. 1525–1534, Aug. 2022, doi: 10.1093/jamia/ocac093.

7. P. C. Austin and E. W. Steyerberg, “The Integrated Calibration Index (ICI) and related metrics for quantifying the calibration of logistic regression models,” Stat. Med., vol. 38, no. 21, pp. 4051–4065, Sep. 2019, doi: 10.1002/sim.8281.

8. G. Vandewiele et al., "Overly optimistic prediction results on imbalanced data: a case study of flaws and benefits when applying over-sampling," Artif. Intell. Med., vol. 111, p. 101987, Jan. 2021, doi: 10.1016/j.artmed.2020.101987.

9. K. El Emam, L. Mosquera, and C. Zheng, "Optimizing the synthesis of clinical trial data using sequential trees," J. Am. Med. Inform. Assoc. JAMIA, Nov. 2020, doi: 10.1093/jamia/ocaa249.

10. J. Drechsler and J. P. Reiter, "An empirical evaluation of easily implemented, nonparametric methods for generating synthetic datasets," Comput. Stat. Data Anal., vol. 55, no. 12, pp. 3232–3243, Dec. 2011, doi: 10.1016/j.csda.2011.06.006.

11. R. C. Arslan, K. M. Schilling, T. M. Gerlach, and L. Penke, "Using 26,000 diary entries to show ovulatory changes in sexual desire and behavior," J. Pers. Soc. Psychol., vol. 121, no. 2, pp. 410–431, 2021, doi: 10.1037/pspp0000208.

12. D. Bonnéry et al., "The Promise and Limitations of Synthetic Data as a Strategy to Expand Access to State-Level Multi-Agency Longitudinal Data," J. Res. Educ. Eff., vol. 12, no. 4, pp. 616–647, Oct. 2019, doi: 10.1080/19345747.2019.1631421.

13. A. Sabay, L. Harris, V. Bejugama, and K. Jaceldo-Siegl, "Overcoming Small Data Limitations in Heart Disease Prediction by Using Surrogate Data," SMU Data Sci. Rev., vol. 1, no. 3, p. Article 12, Aug. 2018.

14. Michael Freiman, Amy Lauger, and Jerome Reiter, "Data Synthesis and Perturbation for the American Community Survey at the U.S. Census Bureau," US Census Bureau, Working paper, 2017. Accessed: Feb. 24, 2020. [Online]. Available: https://www.census.gov/library/working-papers/2018/adrm/formal-privacy-synthetic-data-acs.html

15. B. Nowok, "Utility of synthetic microdata generated using tree-based methods," presented at the UNECE Statistical Data Confidentiality Work Session, Helsinki, Oct. 2015. Accessed: Feb. 24, 2020. [Online]. Available: https://unece.org/statistics/events/SDC2015

16. G. M. Raab, B. Nowok, and C. Dibben, "Practical Data Synthesis for Large Samples," J. Priv. Confidentiality, vol. 7, no. 3, pp. 67–97, 2016, doi: 10.29012/jpc.v7i3.407.

17. B. Nowok, G. M. Raab, and C. Dibben, "Providing bespoke synthetic data for the UK Longitudinal Studies and other sensitive data with the synthpop package for R 1," Stat. J. IAOS, vol. 33, no. 3, pp. 785–796, Jan. 2017, doi: 10.3233/SJI-150153.

18. D. S. Quintana, "A synthetic dataset primer for the biobehavioural sciences to promote reproducibility and hypothesis generation," eLife, vol. 9, p. e53275, 2020, doi: 10.7554/eLife.53275.

19. C. Little, M. Elliot, R. Allmendinger, and S. Samani, "Generative adversarial networks for synthetic data generation: A comparative study," presented at the UNECE Expert Meeting on Statistical Data Confidentiality, Poznań, Poland: United Nations Economic Commission for Europe, Dec. 2021, p. 17. Accessed: Jan. 17, 2022. [Online]. Available: https://unece.org/statistics/documents/2021/12/working-documents/generative-adversarial-networks-synthetic-data

20. J. Taub, M. Elliot, and W. Sakshaug, "The Impact of Synthetic Data Generation on Data Utility with Application to the 1991 UK Samples of Anonymised Records," Trans. Data Priv., vol. 13, no. 1, pp. 1–23, 2020.

Saving Models

There are specific functions in the package for saving and loading the models that are trained. It is important to use these functions as they will ensure that all auxiliary files that are needed are also saved within the model file. Plus, these functions provide a uniform interface across all algorithms.

To save a model use the `save.model()` function, and to read it back again use the `load.model()` function. More details on these functions are available in the on-line help.

It is important to note that the models that are saved may contain some information from the original training data. Therefore, the risk is not that the saved models are prone to an adverserial attack to recover the training data. The risk is that the models themselves may save (depending on the algorithm) distributional and descriptive information about the training dataset that would make it easier to reconstruct the training dataset. That is to say, if the training data contains personal health information, then the models should also be treated as personal health information. and the procedures for managing the models and access to them should be consistent with that level of risk.

Some Recommendations

The following are some general modeling recommendations:

- As will be evident if you run the models on the smaller datasets included with the package, things do not work well with small datasets. Because of the multiple partitions that are needed to deal with data messiness (e.g., rebalancing, encoding, and calibration), small dataset paritions tend to be quite small and unstable.
- Models trained on small dataset will likely overfit. This means that they will show performance results that are quite good, but these results will not be seen when the model is used for predictions on unseen data. Therefore it is important to be cautious with small datasets in that your evaluation results may not carry over to actual implementation scenarios.
- If you want to tune a model and evaluate its performance then it is better to use nested CV. This is more stable (less variation) than vanilla k-fold cross-validation with a hold-out sample. But it will be more computationally intensive.
- Calibration is done for you automatically so you do not have to worry about that. The predicted probabilities from the models that are trained with this package are already calibrated. That functionality is relatively stable and is not likely to change.
- A decision to rebalance the dataset is also made automatically, so you do not have to worry about that. Rebalancing is done using an oversampling technique.
- For the final result / model, always select tuning on. This makes lots of decisions for you and will in general provide you with better models overall.
- Select the loss that you are most interested in. The choice of "logloss" is a generic one. If discrimination is the primary objective then optimize on "auc". If calibration is also an objective (as well as discrimination) then use "brier", although the models are calibrated anyway by default and therefore you have to see if using this loss adds value.
- Keep in mind that this package uses the scaled Brier score rather than the vanilla Brier. And the scaled brier is truncated at zero so that its range is between zero and one. Negative values will not be observed. The scaled Brier is interpreted differently with higher values meaning a better calibration.

Class Assignments

Given that this package was originally developed to support teaching epidemiology and medical students applied machine learning, the following are some ideas about the types of assignments that can be given and supported by this package:

- Compare the performance of different algorithms on datasets and determine if there is a clear pattern of superior performance.

- Evaluate the performance of ML models as the sample size increases by performing a simulation.
- Given a particular dataset and a model specification, train the best model. The submitted model will be evaluated on a hold-out dataset and the best performing top 3 models will get additional points.