SAINT LOUIS UNIVERSITY

HDS 5230: High Performance Computing

Week 11 - Comparison of R/Python for XGBoost

Harika Pamulapati

Professor: Adam Doyle

1. **First, using sampling with replacement, individual predictor variables' distributions were generated, drawing from the original data.**

Method Used	**Dataset Size**	**Testing-set predictive performance**	**Time taken for the model to be fit**
XGBoost in Python via scikit-learn and 5-fold CV	100		
	1000		
	10000		
	100000		
	1000000		
	10000000		
XGBoost in R – direct use of xgboost() with simple cross-validation	100	0.85	0.007s
	1000	0.95	0.023s
	10000	0.971	0.2079s
	100000	0.985	1.614s
	1000000	0.988	14.978s
	10000000	0.995	206.598s
XGBoost in R – via caret, with 5-fold CV simple cross-validation	100	0.940	0.3369s
	1000	0.951	0.217s
	10000	0.9507	1.023s
	100000	0.95311	9.023s
	1000000	0.95474	89.654s
	10000000	0.9654	567.26s

2. Then, the outcome was computed (via prediction) using a logistic model that was fit on the original dataset.

My recommendation is to use the direct implementation of `xgboost()` with simple cross-validation rather than implementing it through `caret`. The direct implementation of `xgboost()` produces superior predictive results throughout all dataset sizes and delivers enhanced testing-set metrics particularly when dealing with large datasets. The model fitting process through direct use of `xgboost()` runs significantly faster than `caret` while delivering better model quality which makes it an efficient solution for both speed and performance.

The built-in 5-fold cross-validation of `caret` comes with a convenient interface but results in excessive computational overhead that produces lower model accuracy than direct usage. The direct application of `xgboost()` becomes essential for large datasets because it provides better scalability and practicality when fitting models. The direct approach stands as the superior choice because it avoids the need for `caret`'s functionalities unless essential for hyperparameter tuning pipelines.