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	0.84	1.22
	1000	0.9470	2.81
	10000	0.9750	1.05
	100000	0.9869	4.15
	1000000	0.9919	43.03
XGBoost in R – direct use of xgboost() with simple cross-validation	100	0.9	0.23
	1000	0.96	0.1807
	10000	0.97	0.464
	100000	0.985	1.931
	1000000	0.988	18.511
	10000000	0.989779	159.465
XGBoost in R – via caret, with 5-fold CV simple cross-validation	100	0.89	1.51
	1000	0.95	1.31
	10000	0.959	2.61
	100000	0.954	16.941
	1000000	0.954	148.48
	10000000	NAN	2781.30

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

The implementation of XGBoost through xgboost() in R with simple cross-validation yields the most efficient and effective solution according to the provided results. The method delivers reliable predictive outcomes at 0.989779 accuracy with 10 million observations through a training process that runs significantly faster than other methods. The direct implementation of R requires 18.5 seconds to complete with 1 million observations while Python takes 43 seconds

and R with caret needs 148 seconds. The direct R implementation maintains practical execution time at 159 seconds when working with 10 million observations but Python and caret become unusable at 2781 seconds and 10 million observations respectively.

The direct R implementation achieves the best performance efficiency ratio among all alternatives. The accuracy advantage that Python achieves through scikit-learn during specific dataset sizes becomes secondary to its speed performance during large-scale processing. The caret implementation provides convenient 5-fold CV through its interface but its performance declines to 0.954 accuracy while processing time becomes excessively long when working with larger datasets. Production environments requiring accurate predictions along with quick processing times should use the direct XGBoost in R approach because it strikes an optimal balance between predictive power and computational efficiency when working with millions of observations.