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	0.93	0.16
	1000	0.9530	0.19
	10000	0.9766	0.54
	100000	0.9871	2.69
	1000000	0.9918	30.82
	10000000	0.9931	269.85
XGBoost in R – direct use of xgboost() with simple cross-validation	100	0.95	0.02
	1000	0.925	0.03
	10000	0.9735	0.27
	100000	0.9859	2.24
	1000000	0.9884	22.82
	10000000	0.9897	240.39
XGBoost in R – via caret, with 5-fold CV simple cross-validation	100	1.00	0.91
	1000	0.98	0.43
	10000	0.96	1.71
	100000	0.9535	12.61

Method used	Dataset size	Testing-set predictive performance	Time taken for the model to be fit
	1000000	0.9547	122.41
	10000000	0.9544	3695.34

The XGBoost Python application with scikit-learn running 5-fold cross-validation strikes an effective compromise between predictive effectiveness and computer processing speed for datasets in the medium to large scale. Smaller dataset performance speeds up results when using the direct R xgboost() function yet Python delivers better accuracy across larger scales (0.9918 for 1 million samples surpasses the R version at 0.9884). The Python implementation shows acceptable time scalability which makes it suitable for applications that need both high accuracy and performance across different dataset sizes.

The R implementation of XGBoost through caret reaches complete accuracy when working with datasets sized 100 but demonstrates declining performance along with slower execution for larger dataset sizes. The fitting process takes more than an hour to complete at 10 million samples while delivering inferior accuracy than alternative methods. The extra abstraction layer in caret produces performance bottlenecks particularly during scale-up operations. XGBoost through the scikit-learn interface in Python offers the best efficiency and scalability when working with large datasets that require dependable results for practical applications.