Linear Regression
Final Report

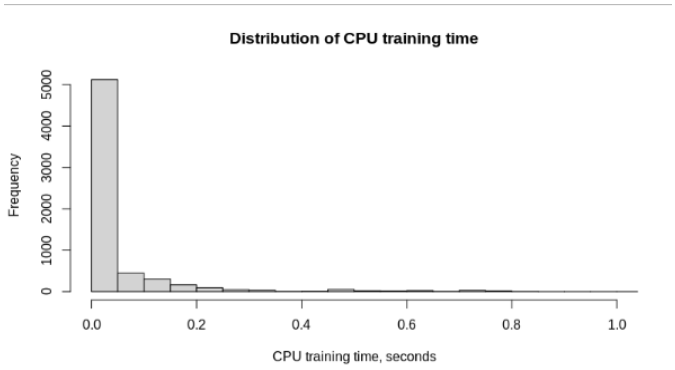
Sanaz Salari

Objectives/Methods

Statistical benchmarking was undertaken to evaluate the computational cost (CPU training time) and accuracy (adjusted R^2) of 10 different regression methods. See later parts of this paper for a full discussion about methods. Measures of central tendency were aggregated at the methods level and applied to 32 data sets. Exploratory analysis was conducted to determine the distributional characteristics of the performance measures and if they were amenable to simple analysis.

Results Summary

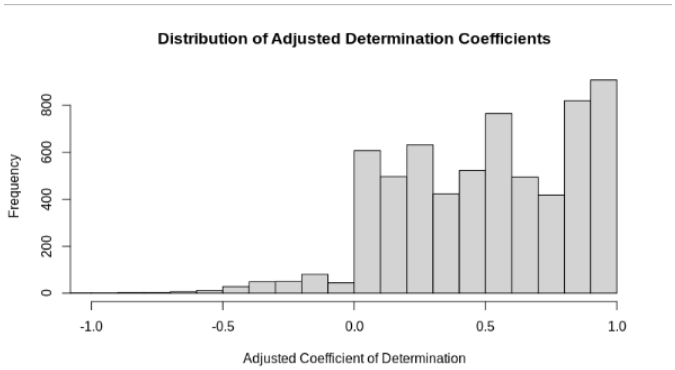
The distributions for CPU training time and adjusted R^2 exhibited strong non-normality as shown in figures below, thus attempts were not made to conduct multiple comparisons between all methods. Principal study results are shown on the next page in Table 1.



Overall, most methods had similar "out of the box" accuracy during training and testing and exhibited minimal bias-variance decomposition. Methods that stood out were tree-based regression, demonstrating greater computational cost with the random forest method being an outlier. The execution times are likely insignificant at smaller scales; however, at larger scales (e.g., methods employed for this paper) the computational cost is non-trivial. The random forest method demonstrated superior training accuracy, but had high variance as evidenced by testing accuracy that was similar to other methods. For unclear reasons, the lqs() method performed poorly overall and may require more complex or nuanced data manipulations or parameter tuning to produce acceptable results;

however, that would be outside the primary scope of this paper. The low degree of decomposition between training and testing metrics seems highly suggestive that the data sets employed had underlying linear relationships.

Table 1. Benchmarking summary: computational cost and accuracy of regression methods.



Method	CPU training time, seconds, mean (SEM)	Adjusted R^2, training, mean (SEM)	Adjusted R^2, testing, mean (SEM)	Adjusted R^2 difference, mean (SEM)*
boost	0.011 (0.0004)	0.54 (0.01)	0.45 (0.01)	-0.089 (0.0046)
elastic	0.003 (0.0003)	0.48 (0.01)	0.44 (0.01)	-0.044 (0.0054)
lasso	0.002 (0.0002)	0.48 (0.01)	0.44 (0.01)	-0.044 (0.0054)
lm	0.002 (0.0002)	0.48 (0.01)	0.44 (0.01)	-0.044 (0.0055)
lqs	0.08 (0.0023)	-0.04 (0.05)	-0.11 (0.06)	-0.072 (0.044)
rf	0.357 (0.0183)	0.77 (0.01)	0.47 (0.01)	-0.293 (0.0099)
ridge	0.003 (0.0003)	0.45 (0.01)	0.41 (0.01)	-0.043 (0.0049)
rlm	0.004 (0.0003)	0.46 (0.01)	0.43 (0.01)	-0.031 (0.0068)
rpart	0.007 (0.0004)	0.56 (0.01)	0.4 (0.01)	-0.166 (0.0066)
svm	0.058 (0.0017)	0.54 (0.01)	0.45 (0.01)	-0.095 (0.0054)

* Negative values indicate worse accuracy with a test set than a training set.

METHOD

This project focused on the utilization of 10+ CRAN regression methods on 30+ datasets. Regression datasets were collected using open libraries such as universities and public research databases. The datasets were chosen based on a criteria of 200+ rows, 2 or more predictor variables, and containing quantitative response variables.

Next an R script was then developed to simultaneously open the datasets, read them, clean them, and have the vital information ready to plug into different methods. A majority of the cleaning focused on renaming columns, converting the data to their proper data types, and formatting numerical categorical variables. Besides that, missing/null data were accounted for. The datasets were then analyzed in order to choose the best response and the predictor variables and placed in a vector of formula strings. The datasets were put into a for-loop as a vector of alternate names for the datasets and indexed like an array to be called on later by the methods.

Originally the datasets were arranged to be opened through a URL format. However, this seemed infeasible, and subsequently, we turned the datasets into environmental objects to minimize miscellaneous issues being present when reading and opening the datasets. Next, an outer and inner for loops were created to compartmentalize the code to be ready for method application. We developed a way to take all these extracted pieces from the datasets and form loops around them to conduct regression testing. We used vectors to store values from the datasets, such as length of datasets, length of methods, and common machine learning tools such as training, testing, and iterations to split the data into 70:30 test: train models to analyze. Inner loops were developed in the for loop which would take the formula strings and data to split it into training and testing data and be ready for method application.

Next, we went further ahead in the outermost for-loop to further extract data information from the formula strings housing the variables, to splitting the data, and finally populating column vectors to store the results of testing methods. This process was developed as a nested for-loop that looked at the outside data while it continued to analyze and extract insights from raw data to applied methods.

Nearing the end of this process, we were researching different methods to work with. Ultimately, we were looking for at least 10 methods that could work well with our code structure, did not have too similar backends as other methods, and ran feasibly with computation and our benchmarks. After much trial and error, we had 10 methods with their backend developed for regression analysis, listed below.

stats::lm() *fitting linear methods for regression*
rpart::rpart() *recursive partitioning and decision trees*
GLMnet
glmnet::glmnet() for ridge *estimating coef. of regression for possible multicollinear predictors*
glmnet::glmnet() for lasso *enhance regression through shrinkage of variable selection and regularization*
glmnet::glmnet() for elastic net *attempts to combine penalties of lasso & ridge to regularize regression*
Robust Regression
MASS::rlm() *fit a linear model for robust regression through M-estimators*
MASS::lqs() *resistant regression to fit good points in a dataset; high breakdown point for an estimator*
Support Vector Machines
svm::svm() *consider points within a decision boundary and develop a line of best fit*
Ensemble
randomForest::randomForest() *deep, independent ensemble regression, using multiple algorithms to fit the most accurate model*
gbm::gbm() *weak, shallow, successive tree*

The methods were then incorporated into the code as the final part of the nested for-loop, with inner code hidden for space. The last of the main source code printed out the 20 iterations of the 10 methods applied to the 30 datasets: We developed an aggregate function to summarize the results of the above iterations in a table.