Machine Learning: An Applied Econometric Approach
Online Appendix

Sendhil Mullainathan
mullain@fas.harvard.edu

Jann Spiess
jspiess@fas.harvard.edu

April 2017

A How We Predict

In this section, we detail the procedure by which we obtain predictions for common machine-learning predictors. Throughout, we use out-of-the box prediction algorithms that are readily implemented in `R`; rather than discussing how any of the algorithms are implemented, we therefore focus solely on how to tune, combine and evaluate them.

Our general approach realizes the main structure that is common to many supervised learning algorithms – namely regularization with empirical choice of the tuning parameters – but leaves many margins of improvement for specific prediction methods, such as strategies to improve regularization (e.g. pruning for trees), tricks to speed up obtaining loss estimates for many tuning parameters simultaneously (e.g. growing larger trees and forests from existing smaller trees and forests), or specific rules to choose the regularization parameter after cross-validation (e.g. the "one standard error rule" for the LASSO and other methods).

A.1 Data management and overall workflow

For the house-price prediction exercise described in the main part of our paper, we want to estimate the performance of different prediction functions, including OLS and several machine-learning predictors. In generating this output, we have followed the following workflow:

1. Divide the data into a training sample and a hold-out sample.
2. On the training sample, run each of the prediction algorithms:
   (a) For prediction algorithms that do not involve any tuning, such as OLS, we simply run the algorithm on the full training sample and store the prediction function.
   (b) For machine-learning predictors that involve a complexity choice (regularization), we tune empirically on the training sample by cross-validation (in our case, we use eight folds in

this step – five to ten folds are standard); once we have chosen a tuning parameter, we rerun the algorithm with this choice of parameters on the full training sample.

(c) We run ensembles last using the tuning parameter choices from the individual algorithms and choose ensembles weights using cross-validation; alternatively, we could choose tuning parameters and weights together for an additional prediction gain (since the optimal tuning parameters within an ensemble are not necessarily the same as those for an algorithm used alone) at the cost of additional computation. We then store the weights along with individual prediction functions fitted on the full training sample.

3. After we have stored all (tuned) prediction functions, we run each on the hold-out sample and produce the statistics we are interested in. For our house-value prediction exercise, for example, we report in Table 1 the overall square-error loss (expressed as R^2) and the performance by quintile of the outcome. On the hold-out sample, we obtain confidence intervals and standard errors using standard analytical formulas for mean estimation or the bootstrap; importantly, these uncertainty estimates represent only variation of the hold-out sample for this *fixed* set of prediction functions, and not the variation of the functions themselves (which would be very hard to estimate for many machine-learning algorithms, as analytical results for the distribution of their prediction functions under general assumptions are scarcely available and the bootstrap is known to fail in many cases that involve empirical tuning).

A.1.1 Clustering and stratification

If data is clustered, we take the clusters into account in constructing the hold-out sample and in choosing fold for cross-validation (that is, we randomize on the level of clusters); note that ignoring clusters in the construction of the hold-out could break the "firewall" between training and hold-out sample and invalidate inference on prediction performance, while ignoring clusters for cross-validation could lead to bad tuning choices (it would likely encourage overly complex model choices since it could reward overfitting). Similarly, one can take strata into account to obtain a balanced hold-out sample and folds.

A.2 Tuning

For individual prediction algorithms, we choose complexity parameters (such as the depth of a regression tree) using cross-validation within the training sample:

1. We divide the training data randomly into eight folds (five to ten are standard).

2. We loop through each fold; for a given fold, we fit the algorithm for every tuning parameter value on *all other* folds and form predictions on the given fold.

3. We obtain one prediction per tuning parameter value for every unit in the training sample; we average the prediction loss over the full training sample for each tuning parameter value.

4. Based on these loss estimates, we choose the tuning parameter (typically by picking the one that minimizes estimated out-of-sample loss).

5. We then fit the algorithm with the chosen tuning parameter on the full training sample.

6. Eventually, we evaluate the function fitted on the full training sample on the hold-out sample; at that point, the tuning parameter is fixed.

A.2.1 Tuned vs untuned out-of-the-box procedures

We use out-of-the box prediction algorithms, such as `rpart` for regression trees in `R`. These algorithms usually come with default choices of tuning (complexity) parameters, often based on some heuristics; without any further arguments, `rpart`, for example, fits a regression tree with at least 20 units in each non-terminal node and a minimal improvement of 1% (in terms of R^2) per split.

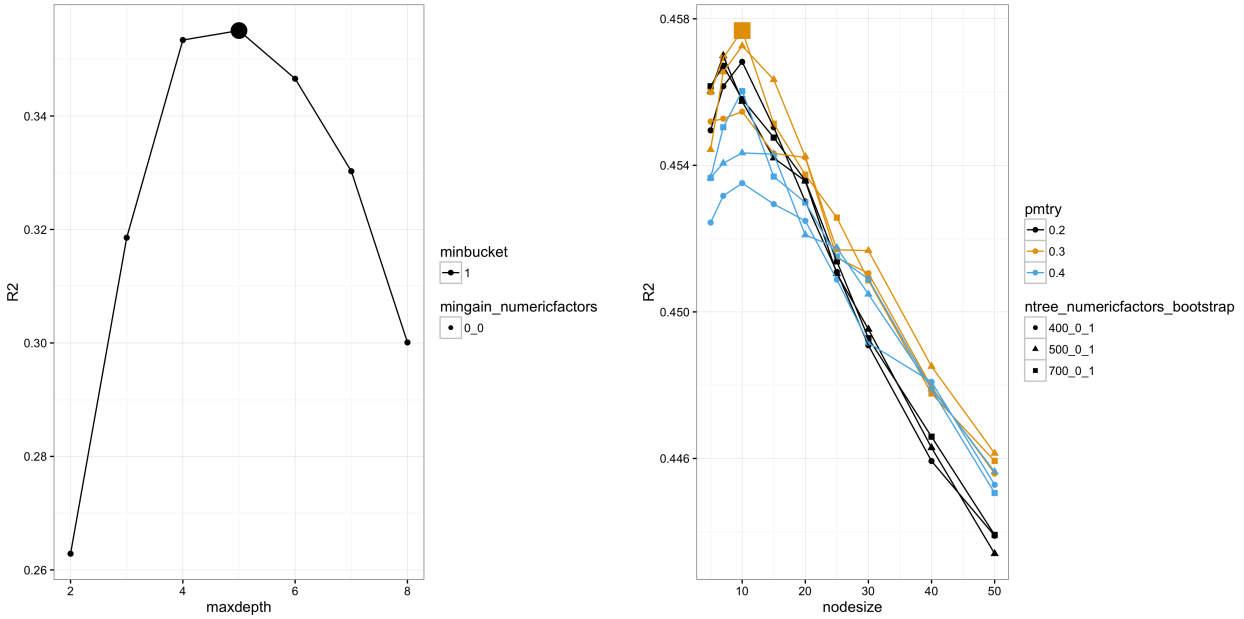
While we use these algorithms "as-is", we do *not* simply run them with their default parameters, but choose tuning parameter values by cross-validation. While the default rules usually perform quite well on average, they do not necessarily perform well on any given sample. To illustrate this point, we report in Table B.1 below the performance of `rpart` in our prediction exercise without empirical tuning, and compare it with a fully tuned regression tree. Tuning alone – holding the data and the function class fixed – accounts for a gain in over five percentage points R^2 over the default settings in this example.

A.2.2 Multiple tuning parameters

When there are multiple tuning parameters, such as the depth of the tree or the minimal size of a non-terminal node, we choose these parameters jointly; that is, we run the above procedure over a grid of multi-dimensional parameter values. In particular, these parameters will usually interact with each other, and a less complex choice in one dimension may allow for a more complex choice in another.

A.2.3 Inspecting tuning results

The range of parameter values we search over is a central input to the empirical tuning exercise, and tuning may do badly if the optimal parameter is outside this range. To get a sense for whether this could be the case, we inspect the cross-validated performance by parameter values and check whether the chosen parameter value is in the interior of the range we provided. Figure A.1a shows the tuning plot for a regression tree tuned by depth; here, the chosen depth (five) is in the interior – if it were not, we would rerun the tuning step with a different range of parameter values.



(a) Regression tree tuned by depth.

(b) Random forest tuned by three parameters.

Figure A.1: Tuning plots with cross-validation estimates of out-of-sample R^2.

While it is easy to see whether a one-dimensional parameter is in the interior, that may be harder to assess for multi-dimensional tuning parameters. Figure A.1b shows a tuning plot for a random forest. While the chosen complexity is interior with respect to `nodesize` (the minimal node size of each tree) and `pmtry` (the proportion of variables used in each tree), it is not interior with respect to `ntree` (the number of trees).

A.2.4 Picking the tuning parameter after cross-validation

In our house-value prediction exercise, we pick the tuning parameter values that minimize the cross-validation out-of-sample loss estimates. However, there are heuristics available that suggest the choice of a different parameter value after cross-validation that is not necessarily the one that minimizes estimated loss directly; for example, the "one standard error rule" (Friedman et al., 2001), which is frequently applied for the LASSO, picks the value for the tuning parameter corresponding to the least complex model for which the loss estimate falls within one standard error of the empirical minimizer. Similarly, one may want to account for the fact that the function will eventually be fitted on a larger sample than during cross-validation and thus adjust tuning parameters accordingly.

A.2.5 Cross-validated vs hold-out performance after tuning

It is tempting to take the actual value of the cross-validated loss estimate at the chosen tuning parameter as an estimate of out-of-sample performance of the final function – in particular, why

would we need a hold-out if we have already estimated out-of-sample performance during tuning? Unlike the hold-out loss estimate of the final prediction function, the cross-validation estimate obtained during tuning may be biased and misleading:

1. It may be too optimistic because it is the realized empirical minimizer of a number of loss estimates by the choice of tuning parameter; if we try many tuning parameter values, we may end up overfitting even in cross-validation.

2. It may be too pessimistic because the final function is fitted on the full training sample, while the functions are fitted on smaller sub-samples during cross-validation.

3. It is not an estimate of loss of the fixed prediction function we obtain from the training sample, but an average over the performance of multiple different prediction functions.

For an unbiased estimate of prediction performance of the final prediction function, we look into the hold-out instead.

A.3 Fitting ensembles

Ensembles – that is, weighted combinations of different prediction algorithms – consistently come out on top of prediction competitions. Fitting ensembles can be seen as another tuning step, where the ensemble weights are tuning parameters that can be chosen by cross-validation. For simple weighted linear combinations and square-error loss, the calculation of weights becomes particularly easy – in particular, we construct the ensembles in the main paper using the following procedure:

1. We divide the training data randomly into eight folds (as above) to perform cross-validation; for every fold, we fit each of the algorithms that make up the ensemble on all other folds, and predict on the current fold.

2. We have thus obtained one prediction per unit in the training sample for every algorithm. To fit weights, we simply run a linear regression of the outcome on these predicted values in the full training sample, and store the resulting linear model.

3. To predict in the hold-out, we fit each algorithm on the full training sample, obtain predictions for each algorithm on the hold-out, and then predict with the linear model obtained in the previous step from these fitted values.

Note that running OLS does not necessarily produce weights that add up to one, and may also produce an intercept. While we could exclude the intercept and search instead for weights that add up to one, in our experience the OLS ensemble fit (with intercept) performs as well or even better than such an ensemble.

A.3.1 Ensembles and the tuning of individual algorithms

The fact that we use an algorithm inside an ensemble with other prediction functions likely changes the optimal tuning parameters; hence, there may be a gain from choosing all tuning parameters and the weights together in a single cross-validation loop. However, for computational purposes we practically stick with the tuning parameters that we have chosen by cross-validation for each algorithm separately, and simply use the algorithms with these parameter values.

That said, we use a separate cross-validation split for the ensemble, and do not rely on in-sample fitted values in the training sample:

1. If we used the same cross-validation split as for cross-validated tuning, we would give an advantage to those algorithms with many or high-dimensional tuning parameter values: since we choose the empirical loss minimizer, an algorithm that is tuned over more choices of tuning parameters may look better at the empirical minimizer because it overfits more than one with less (or no) choices.

2. More importantly, we do not use the in-sample fitted values in the training sample to choose ensemble weights, as this would produce larger weights for algorithms that overfit more; the random forest, for example, is know to overfit heavily even at optimal choices of tuning parameters.

As a consequence, we always draw new folds (from the same training sample) and perform a new round of cross-validation to choose the ensemble weights.

A.4 Feature representations

The representation of variables is essential to obtaining high-quality predictions, especially since it interacts with the choice of function class and regularizer. In particular, making informative structure "easy to find" for a complexity-constrained machine learning algorithm can create large prediction gains.

A.4.1 Normalization and the hold-out

Especially when penalized linear predictors like the LASSO or ridge regression are used, covariates are often normalized (for example to zero mean and unit variance) before fitting the prediction function. When we perform such normalization, we see it as part of the prediction algorithm itself and are thus careful to not use the data in the hold-out for it so that our ability to obtain unbiased loss estimates is not jeopardized. To be concrete, we transform only the data the algorithm is fitted on, and then transform coefficient estimates back to work on the original data before forming predictions out of sample. This way, different prediction algorithms – may they normalize their data or not – remain comparable in cross-validation and hold-out evaluation.

A.4.2 Redundant features

Certain variable transformations that would be redundant for an unregularized linear model and create collinearities – such as including census regions in addition to states (which fall into exactly one of the four regions) – can be valuable in penalized linear models or other machine-learning predictors: if meaningful coarser geographic subdivisions are included, the model may obtain better predictions with less complexity if reconstructing them from their components would incur a high complexity penalty. We therefore typically include coarser, redundant measures in addition to the base data when they are available.

B Details on the House Value Prediction Exercise

B.1 Sample selection, data pre-processing and variable encoding

Starting with the 2011 American Housing Survey sample of 186,448 units, we consider all owner-occupied units (`TENURE` $= 1$, `VACANCY` $= -6$) with non-missing and positive value (`VALUE` > 0) from the metropolitan sample (`NATLFLAG` $= 2$), which we divide randomly into a training sample of 10,000 and a hold-out sample of 41,808. In addition to the (natural) logarithm of unit value, we include all available characteristic and quality variables. For geography, we decide to include city/suburban status (`METRO`, `METRO3`) as well as census region (`REGION`). For numeric variables, we add categorical missingness variables that preserve potentially different missingness values. We further merge rare factor levels (including for missingness indicators) as to ensure that no new factor levels appear on the hold-out sample, and further delete variables that are constant on the trainings sample. We obtain a total of 150 right-hand side variables along with twelve missingness indicators from which we predict log house value.

B.2 Detailed results of the prediction exercise

In addition to overall loss (mean-squared error), we also report the MSE by quintile of log(`VALUE`) on the hold-out sample. Table B.1 extends Table 1 from the main paper by the underlying (absolute) MSE and by individual results for additional prediction methods, including an untuned tree using the default settings of `rpart` in `R` and a boosted tree using `xgboost`.

B.3 LASSO barcode graph

Figure 2 visualizes the selected (non-zero) coefficients from repeated LASSO regressions on ten subsets of the full (training and hold-out) data from the original prediction exercise. To obtain the graph, we proceed as follows:

1. We pick a random subset of 10% of the 51,808 units in the combined training and hold-out samples from our empirical exercise; we fit LASSO regressions at varying values of the

Method	Prediction performance					Mean-squared error (MSE) by quintile of unit value				
	Training sample		Hold-out sample							
	MSE	R^2	MSE		R^2	1st	2nd	3rd	4th	5th
OLS	0.589	47.3%	0.674	(0.025)	41.7%	2.180	0.247	0.203	0.203	0.515
Tree (depth)	0.675	39.6%	0.758	(0.026)	34.5%	2.431	0.22	0.19	0.233	0.678
Lasso	0.603	46.0%	0.656	(0.025)	43.3%	2.151	0.218	0.177	0.183	0.525
Forest	0.167	85.1%	0.631	(0.025)	45.5%	2.105	0.189	0.148	0.167	0.517
Ensemble	0.219	80.4%	0.626	(0.025)	45.9%	2.081	0.207	0.167	0.175	0.476
Tree (depth, min. size)	0.608	45.6%	0.696	(0.025)	39.9%	2.206	0.236	0.204	0.22	0.584
Tree (`rpart` defaults)	0.720	35.5%	0.759	(0.026)	34.4%	2.402	0.240	0.193	0.225	0.697
Boosted tree	0.531	52.5%	0.651	(0.025)	43.8%	2.149	0.201	0.167	0.178	0.531

Table B.1: Detailed results from the prediction exercise, extending Table 1 in the main paper. Trees have respective tuning parameters in parentheses. Standard errors represent measurement variation on the hold-out sample, holding the prediction function fixed.

complexity penalty λ on this subsample, and obtain estimates of out-of-sample prediction quality on the remaining units. We then pick the complexity penalty λ^* that minimizes these loss estimates.

2. We divide the full data into ten randomly chosen (non-overlapping) folds, fit LASSO regressions for this fixed λ^* on each, and plot their non-zero coefficients.

We keep the regularization parameter value fixed in order to distinguish between the instability in tuning through cross-validation from that of fitting a model of the same complexity on multiple draws from the same data-generating process; similar results could be obtained by also repeating cross-validation tuning within each fold.

References

Friedman, J., Hastie, T., and Tibshirani, R. (2001). *The Elements of Statistical Learning*, volume 1. Springer series in statistics Springer, Berlin.