Modern Statistics and Machine Learning: Trees and Random Forests

Data Science for Marketing Decision Making
Günter J. Hitsch
Chicago Booth

Winter 2017

Overview

1. **C**lassification **a**nd **r**egression **t**rees (CART)
2. Random forests

Regression trees

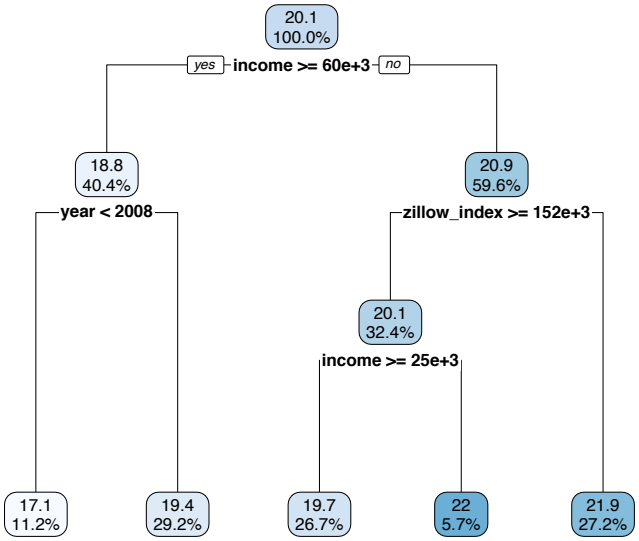
Elements of a tree:

- Root node
- Internal nodes, labeled with one of the inputs
- Terminal nodes or leaves

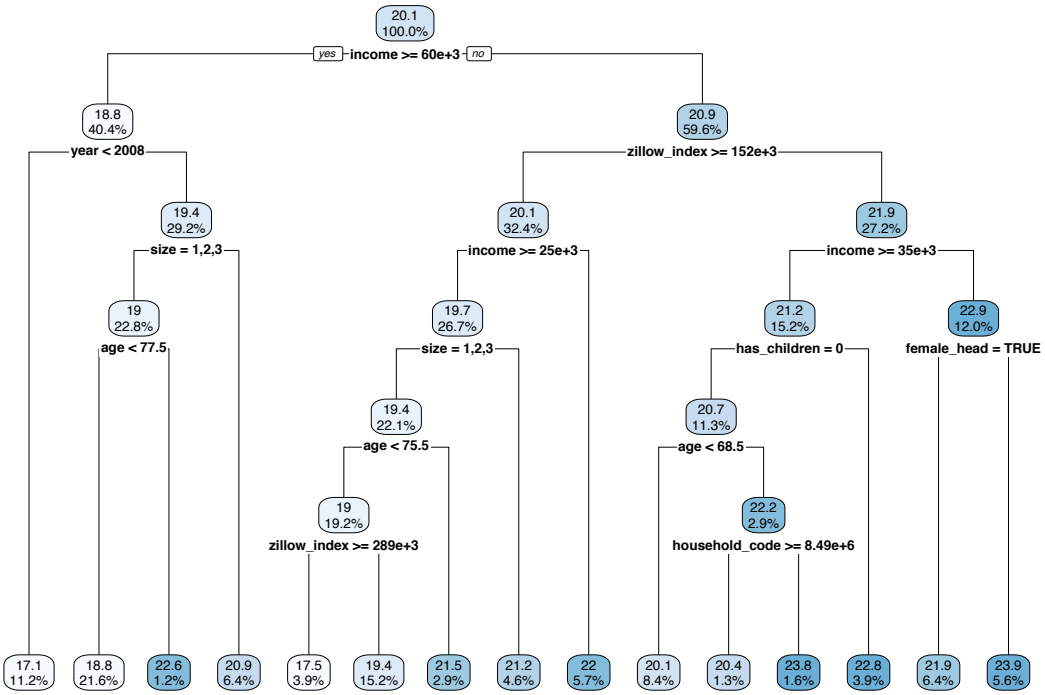
Binary splitting rules (e.g. year < 2008) create branches at the root and at each internal node

Predict output based on mean of y_i among all observations i in a leaf (percent of observations in each leave also indicated)

- Household-level private label shares



A more complex tree



Partitioning

Observation indices: $i \in \mathcal{N} = \{1, \ldots, n\}$

Labels for the leaves (terminal nodes): $l \in \mathcal{L} = \{1, \ldots, L\}$

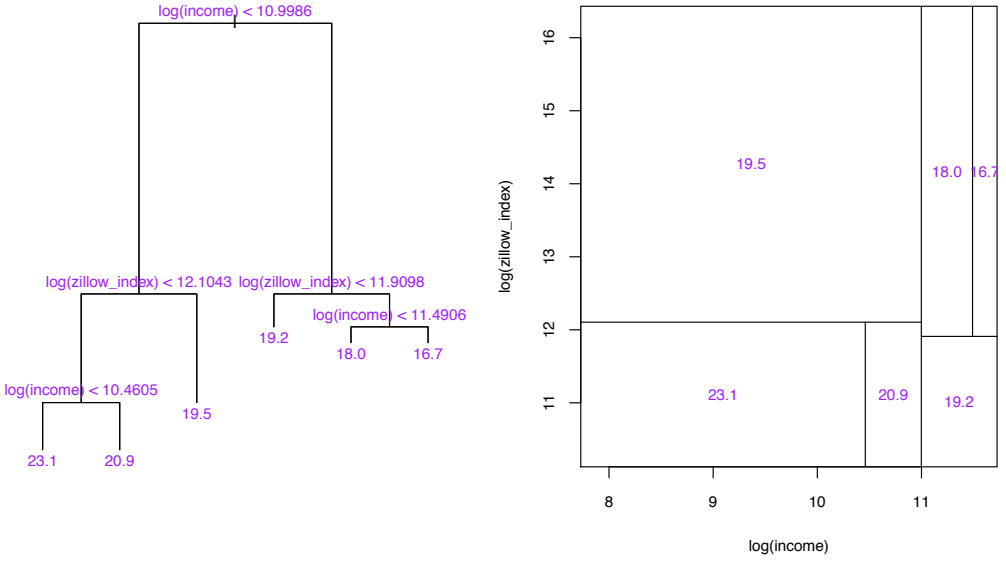
Using a tree and its splitting rules, each observation $i \in \mathcal{N}$ is assigned to exactly one of the leaves (terminal nodes) l. Hence, each leaf is a collection of observations, $\mathcal{R}_l = \{i \in \mathcal{N} : i \text{ is assigned to leave } l\}$.

Because each observation is assigned to exactly one $\mathcal{R}_l$, the leaves form a **partition of the data**. The predicted output for observation i in leaf l is

$$\hat{y}_{\mathcal{R}_l} = \frac{1}{N_l} \sum_{j \in \mathcal{R}_l} y_j.$$

This is simply the mean output over all observations in leaf l. Notation: N_l is the number of observations assigned to leaf l.

Example: Each observation i with inputs $(\texttt{log(income)}_i, \texttt{log(zillow\_index)}_i)$ is assigned to one of six leaves. In the leaf defined by $\texttt{log(income)} \geq 10.9986$ and $\texttt{log(zillow\_index)} < 11.9098$ the predicted output is $\hat{y} = 19.2$.



Growing a regression tree

Algorithm: Start at the root node, and successively add splits and thus additional nodes.

How to add splits: Focus on one terminal node in the *current* tree. $\mathcal{R}$ is the set of observations i in this current terminal node. The RSS (sum of squared residuals) in $\mathcal{R}$ is

$$RSS(\mathcal{R}) = \sum_{i \in \mathcal{R}} (y_i - \hat{y}_{\mathcal{R}})^2$$

Now we attempt to decrease the RSS by splitting $\mathcal{R}$. Pick one of the inputs, k, and some cutpoint, s. Input k and cutpoint s split $\mathcal{R}$ into the two regions

$$\mathcal{R}_a(k, s) = \{i : x_{ik} < s\}, \quad \mathcal{R}_b(k, s) = \{i : x_{ik} \geq s\}$$

The new RSS for the observations in $\mathcal{R}$ when adding this split is

$$RSS(k, s) = \sum_{i \in \mathcal{R}_a} (y_i - \hat{y}_{\mathcal{R}_a})^2 + \sum_{i \in \mathcal{R}_b} (y_i - \hat{y}_{\mathcal{R}_b})^2$$

Now choose one of the inputs $k = 1, \ldots, p$ and a cutpoint s to minimize $RSS(k, s)$. We thus add a split to reduce the sum of squared residuals, $RSS(\mathcal{R})$, by the largest amount. Correspondingly, we grow the tree: The current terminal node becomes an interior node with two branches.

This is a so-called *greedy algorithm*, and it will not generally find the partition that provides the best fit to the data. But it is a computationally feasible algorithm!

Now repeat this process along all terminal nodes in the *current* tree.

Use a *stopping criterion* to decide when to stop growing the tree. For example, stop if any additional split would add a leaf with number of observations below a minimum node size (e.g. 10).

Note: Split needs to be modified for categorical variables (the two regions need to be defined based on the category labels).

Regression trees are regression models

To see this, define the indicator (dummy) variable corresponding to leaf l

$$D_{il} = \begin{cases} 1 & \text{if } i \text{ is in leaf } l, \\ 0 & \text{otherwise} \end{cases}$$

Then the predictions obtained from a regression tree are identical to the predictions from the regression

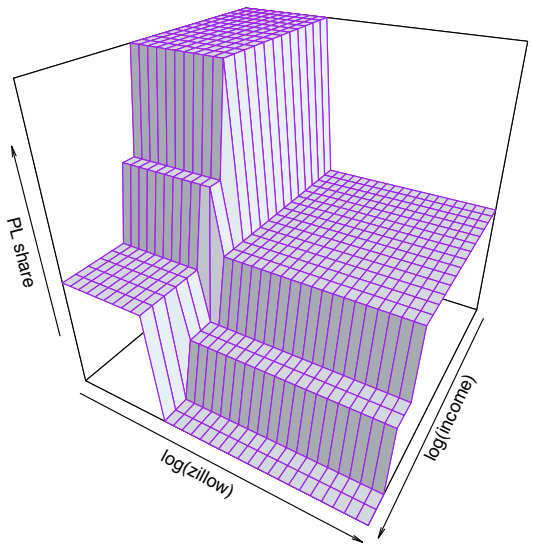
$$y_i = \sum_{l=1}^{L} \beta_l D_{il} + \epsilon_i$$

Indeed: $\hat{\beta}_l = \hat{y}_{\mathcal{R}_l}$.

This is similar to RFM analysis. However, unlike in RFM analysis, the tree (and thus the partition = segments) is grown in order to provide a good fit to the data!

The regression function estimated using a tree, $\mathbb{E}(y|X = x)$, is a **step-function** approximation.

Note the *non-linearities* in the two inputs and the i*nteraction* between income and the Zillow index: The effect of income depends on the value of `log(zillow)`.



Regression trees: Discussion

Benefits of regression trees:

- Flexible (non-parametric) regression functions
- Allows for interactions between inputs
- Can be easily interpreted if the tree size is small
- Trees can be displayed graphically — this is great to communicate the estimation results (if the tree size is not too large)

Prediction using regression trees

Predict output for a new observation with inputs $\boldsymbol{x}$:

1. Use the tree to find the leave l that $\boldsymbol{x}$ belongs to
2. Predicted the output value based on $\hat{y}_{\mathcal{R}_l}$

Out-of-sample fit and the bias-variance tradeoff

Prediction in a regression tree is straightforward, but especially in a large regression tree (small node size) the prediction quality will be poor because of over-fitting.

Why this happens is easy to see: In the limit, if each node is based on only one observation, the in-sample fit will be perfect! Out-of-sample, not so much — this is the *bias-variance tradeoff*.

Pruning is a common method to reduce the variance of the prediction by adding a cost for growing a tree with many leaves.

Tree pruning

First, grow a very large, complex tree. Then prune the tree back by eliminating branches to obtain a **subtree**. For any such pruned tree $\mathcal{T}$ with number of leaves $L_{\mathcal{T}}$ we evaluate the following **cost complexity criterion**:

$$\sum_{l=1}^{L_{\mathcal{T}}} \sum_{i \in \mathcal{R}_l} (y_i - \hat{y}_{\mathcal{R}_l})^2 + \alpha L_{\mathcal{T}}$$

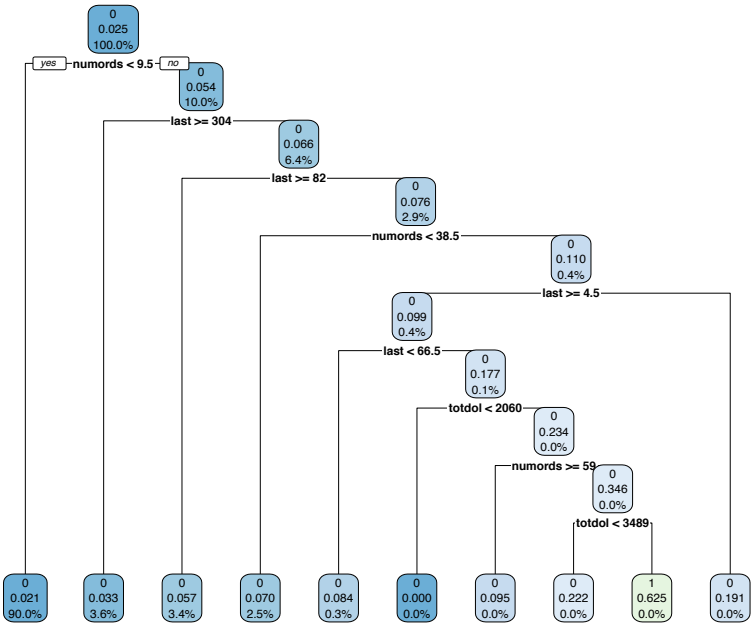
α is the *cost complexity parameter*. This criterion introduces a tradeoff between in-sample fit and the cost of a complex tree with many leaves. Note that this is very similar to the penalty term in a ridge regression or LASSO that allows for regularization.

For any given cost complexity parameter α we can find the subtree (of the original, big tree) that minimizes the cost complexity criterion.

How do we determine a good value of the tuning parameter α? — Cross-validation!

Classification trees

Example: RFM data set



Classification trees are used to predict categorical outcomes, $y_i \in \{1, 2, \ldots, K\}$

The tree-building algorithm is similar to the approach used for regression trees, but the RSS (sum of squared residuals) criterion to evaluate fit cannot be applied. Instead, build trees to maximize the **purity** of a node.

Example: Consider a binary classification problem, as in the RMF example. Consider all observations i in a leaf $\mathcal{R}$ of the tree, and calculate the proportion of outcomes such that $y_i = 1$:

$$\hat{p}_{\mathcal{R}} = \frac{1}{N_{\mathcal{R}}} \sum_{i \in \mathcal{R}} y_i$$

The *Gini index* is a common measure of node purity:

$$G_{\mathcal{R}} = \hat{p}_{\mathcal{R}}(1 - \hat{p}_{\mathcal{R}})$$

You can verify that $G_{\mathcal{R}} = 0$ if $\hat{p}_{\mathcal{R}} = 0$ or $\hat{p}_{\mathcal{R}} = 1$, and that the value of $G_{\mathcal{R}}$ is at its maximum when $\hat{p}_{\mathcal{R}} = 1/2$. Hence, maximization of node purity can be achieved if a split creates two new leaves with outcomes that are as homogenous as possible.

Interlude: The bootstrap

The bootstrap is a **resampling method** (like cross-validation) that is used to simulate the sampling distribution of an estimator. This is useful, for example, to estimate the standard error or to create a confidence interval **if the true distribution of the estimator is not known** (or hard to calculate).

Indeed, the exact distribution of most estimators can only be derived (if at all) for $n \to \infty$ — arbitrarily many observations, but not for a given n. This is true even for OLS, unless the error term is normally distributed.

The **ideal solution** to derive the sampling distribution of an estimator $\hat{\theta}$ based on a sample with n data points:

1. Obtain B (say $B = 1000$) independent samples with n data points
2. Calculate the estimator, $\hat{\theta}_b$, for each of the samples $b = 1, \ldots, B$
3. Predict the standard error of $\hat{\theta}$ using the standard deviation of the B $\hat{\theta}_b$ values, calculate a 95-percent confidence interval based on the 2.5th and 97.5th percentile of the $\hat{\theta}_b$ values, etc.

The ideal solution is infeasible because we only have one data set.

The **bootstrap** is based on B data sets from the original population:

1. Obtain B data sets by randomly sampling n observations from the original data (which has n observations) *with replacement*
2. Calculate the estimator, $\hat{\theta}_b$, for each of the bootstrap samples $b = 1, \ldots, B$
3. Predict the standard error of $\hat{\theta}$ using the standard deviation of the $\hat{\theta}_b$ values, calculate a 95-percent confidence interval based on the 2.5th and 97.5th percentile, etc.

The bootstrap is easy to implement (on a modern computer) and tends to provide better estimates (e.g. a confidence interval) than an asymptotic approximation.

Random forests

We discussed the advantages of trees (flexibility, tool to communicate and visualize the results), but in practice they are a poor tool for prediction.

A random forest is an estimator of a conditional expectation function $\mathbb{E}(Y|X = \boldsymbol{x})$ that inherits the flexibility of trees (non-parametric estimator) while providing much better out-of-sample fit.

Random forests are an **ensemble method**: Instead of one algorithm to predict, we use an ensemble of B algorithms. To predict, we take a weighted average of the prediction (vote) of each algorithm.

Construction of the ensemble:

1. Bagging
2. Selection of random subset of inputs

Bagging

Bagging means "**b**ootstrap **agg**regation"

The bagging algorithm:

1. Create B training data sets using the bootstrap
2. Estimate the prediction algorithm for each training set b
3. Average over all B predictions

The random forest algorithm trains an ensemble of B (say 500 or 1000) trees, $\mathcal{T}_b(\boldsymbol{x})$. Then predict or approximate the conditional expectation by averaging over the predictions of each tree:

$$\mathbb{E}(Y|X = \boldsymbol{x}) \approx \frac{1}{B}\sum_{b=1}^{B}\mathcal{T}_b(\boldsymbol{x})$$

Purpose of bagging: Reduce variance of prediction!

Selection of random subset of inputs

When building a tree: At each attempted split the random forest algorithm will only split based on a randomly chosen subset of all p inputs (e.g. $\sqrt{p}$ or $p/3$ randomly chosen inputs).

This is sometimes called "feature bagging" or a "random subspace method".

Goal: Break reliance of prediction on a possible small number of "dominant" features and explore a wider set of inputs to determine splits.

Reduces overfitting and the variance of the prediction, and improves what we care about, the out-of-sample predictive power of the estimator.

Example

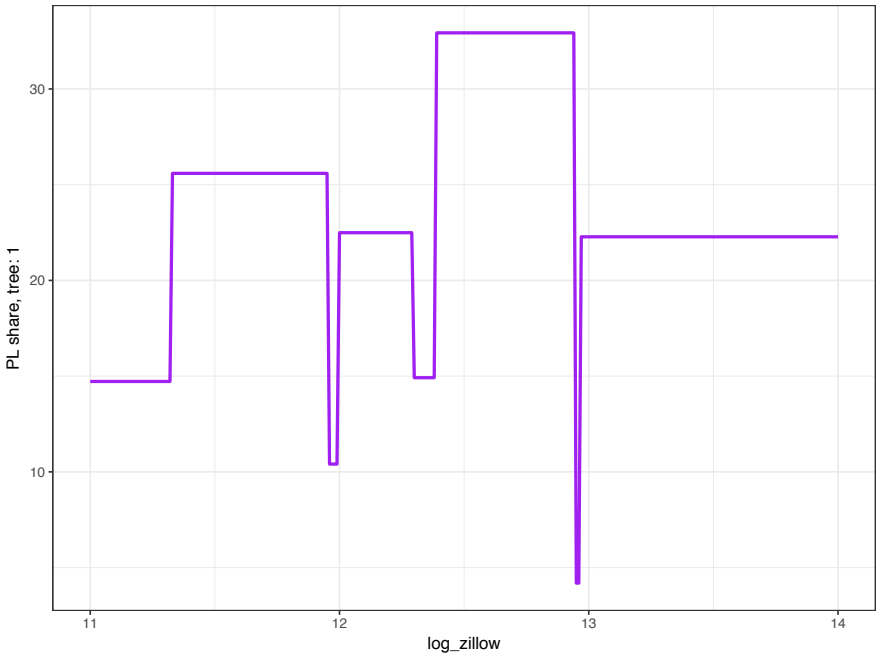
See `Random-Forests.Rmd`

- Example: Private-label share and housing prices (Zillow)

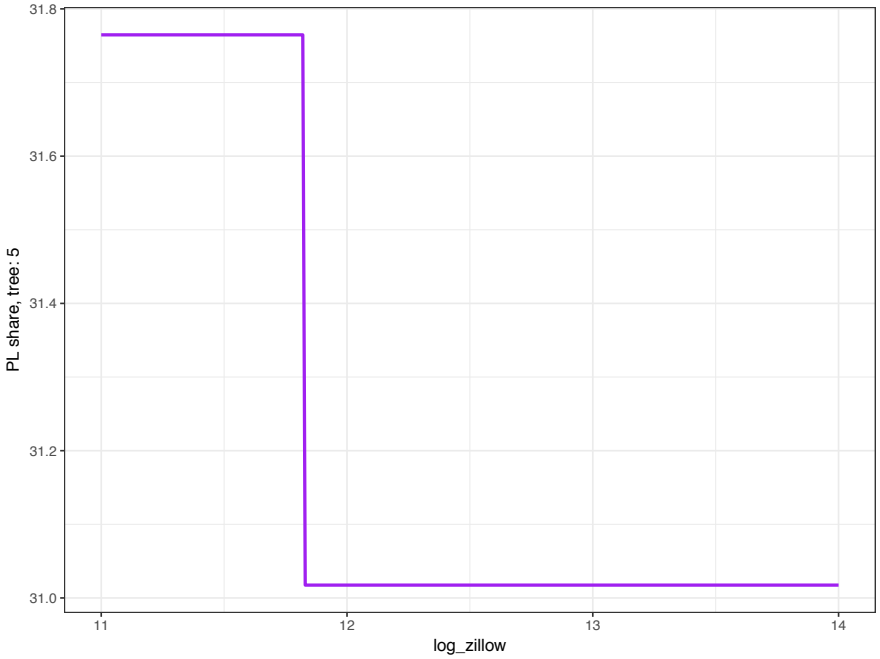
Explore what each individual tree (based on bagging and random feature subset selection) looks like, and how the individual trees aggregate to the random forest prediction:

- `Individual-Trees.pdf`
- `Tree-Averages.pdf`

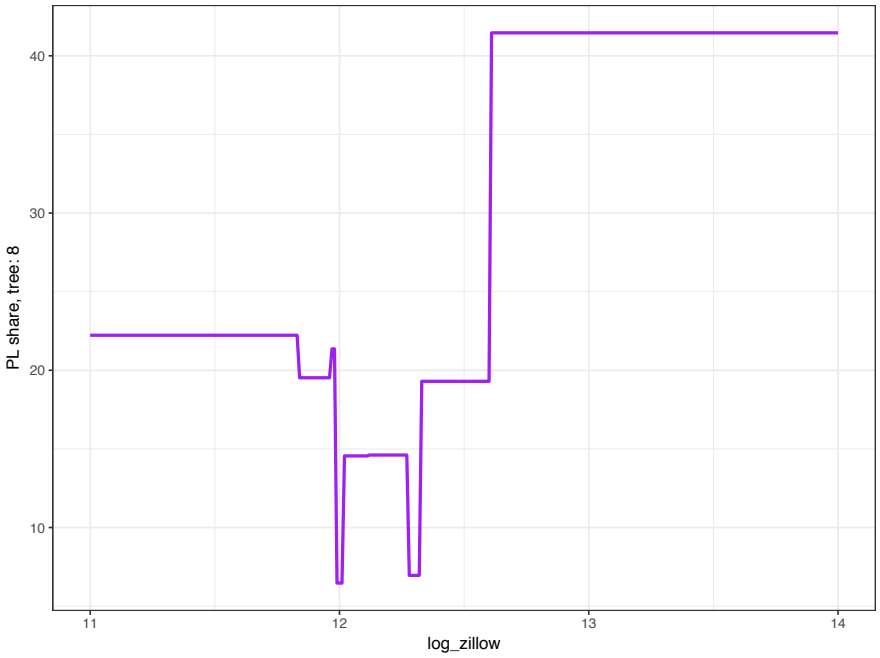
Tree 1


30
20
10
PL share, tree: 1
11
12
13
14
log_zillow

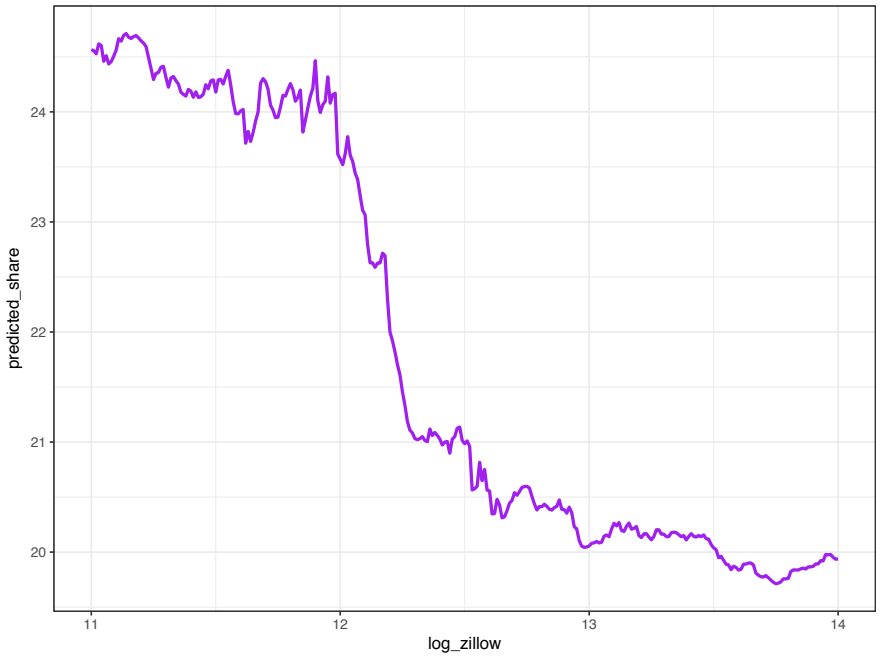
Tree 5


31.8
31.6
31.4
31.2
31.0
PL share, tree: 5
11
12
13
14
log_zillow

Tree 8



Random forest prediction averaged over 2000 trees



Discussion

Note how a random forest can approximate a "continuous" relationship, even though each individual tree can't — because of bagging and random subset of features selection

One "price" of random forests:

- Computational costs — even on a fast computer building 1000 trees may take several hours (or more) if the data size (n and p) is large
- Test algorithm using a small number of trees before "production run", save output or predictions from the final run for later use

Summary

1. Classification and regression trees (CART)
   - Flexible approximation of regression functions (non-linearities, interactions)
   - Good for visualization and communication
   - Often poor out-of-sample predictions
2. Random forests
   - State-of-the-art non-parametric estimator
   - Bagging
   - Random feature subset selection
   - Computational cost can be high, but feasible to implement on a modern computer