McCOMBS SCHOOL OF BUSINESS

Salem Center for Policy

The bias-variance tradeoff and new modeling techniques

David Puelz

Prediction

Let's go back to supervised learning aka prediction.

There was a lingering problem of which subset of variables I use for my regression model. It is closely related to model selection, and we will cover important ideas related to it here.

Remember our supervised learning goal

Predict a target variable Y with input variables X.

Remember our supervised learning goal

Predict a target variable Y with input variables X.

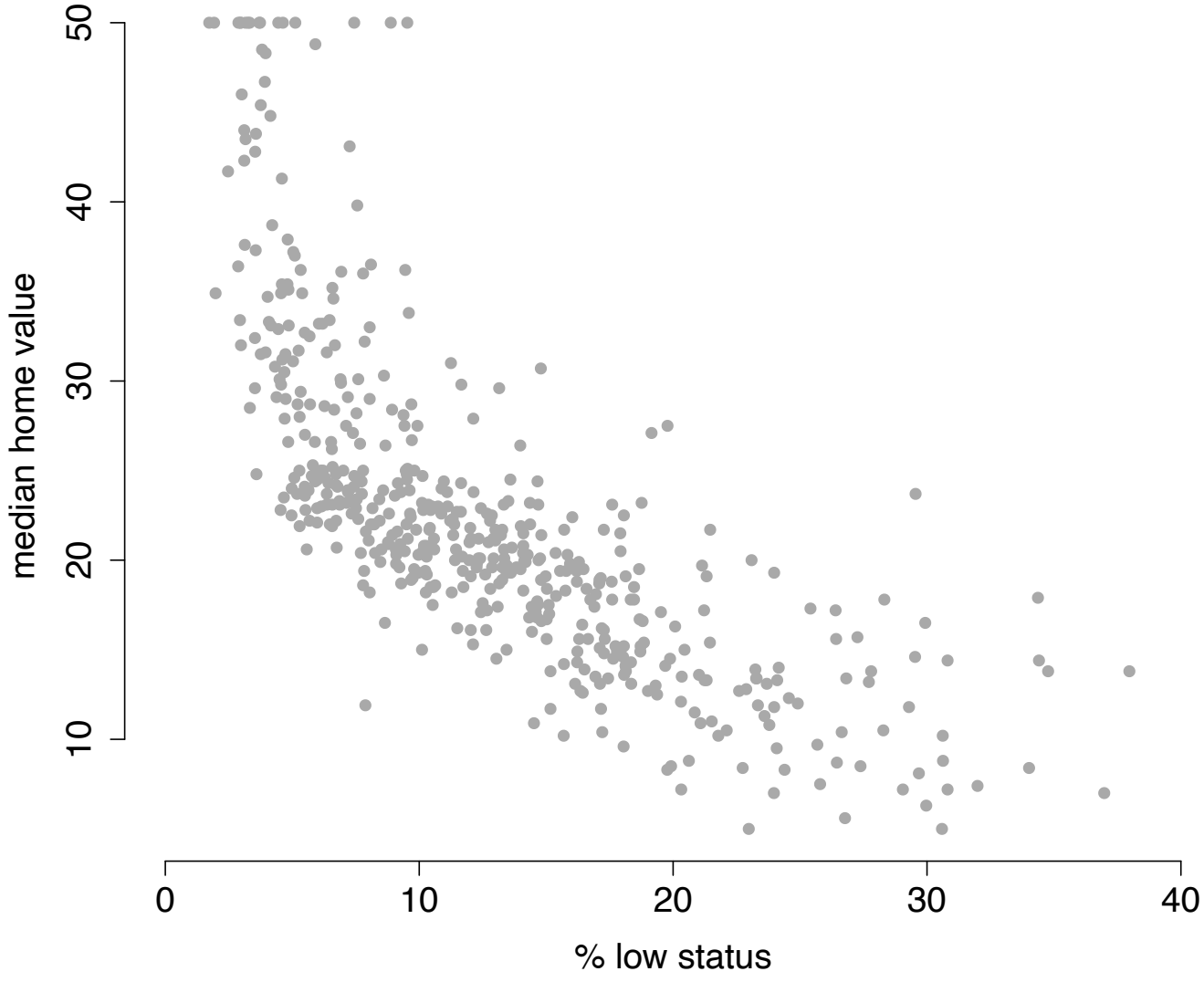
We can frame the problem by supposing Y and X are related in the following way:

$$Y_i = f(X_i) + \epsilon_i$$

To achieve our goal, we need to: *Learn or estimate* $f(\cdot)$ from data.

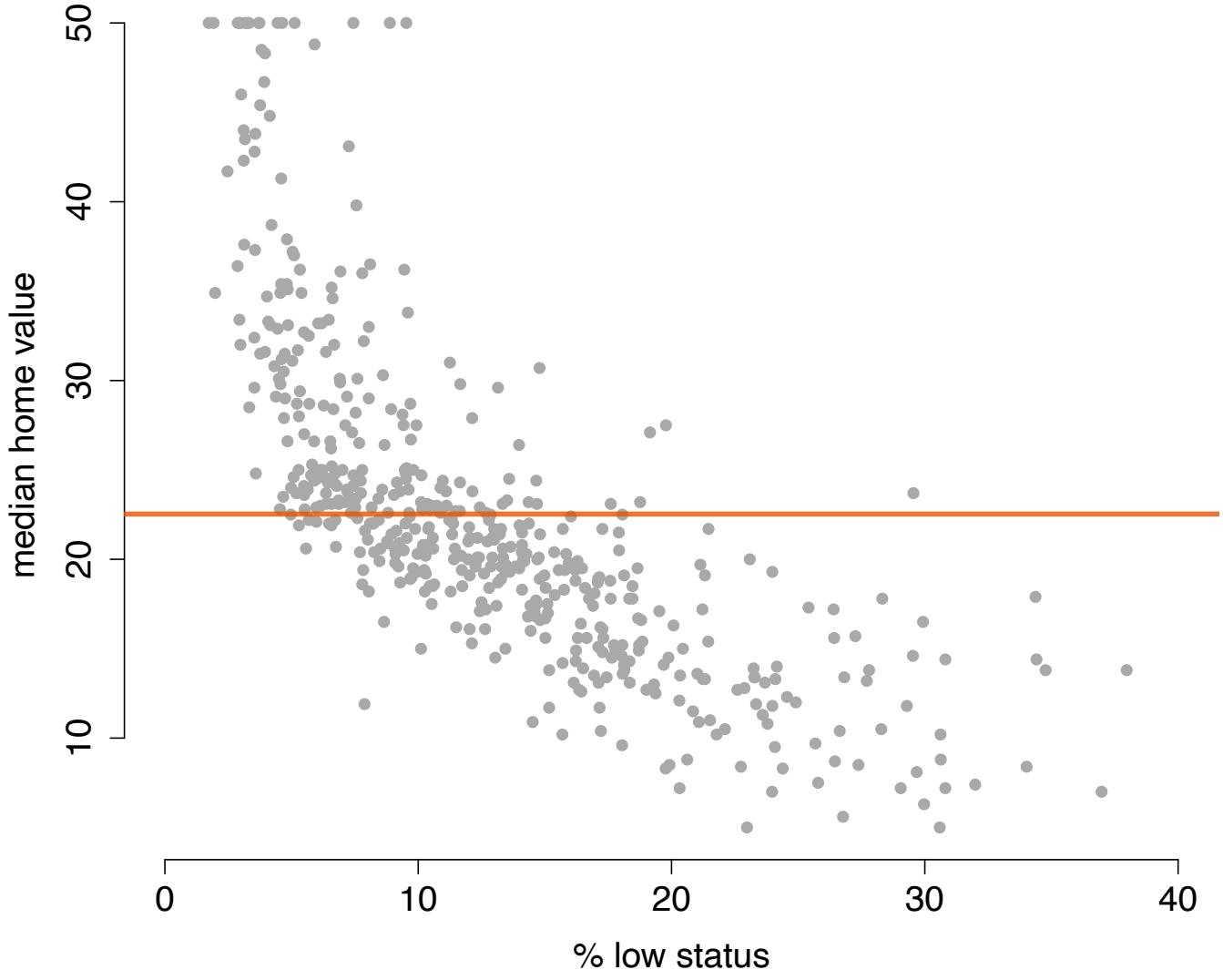
Boston housing data

Predict median home value with percent low economic status.



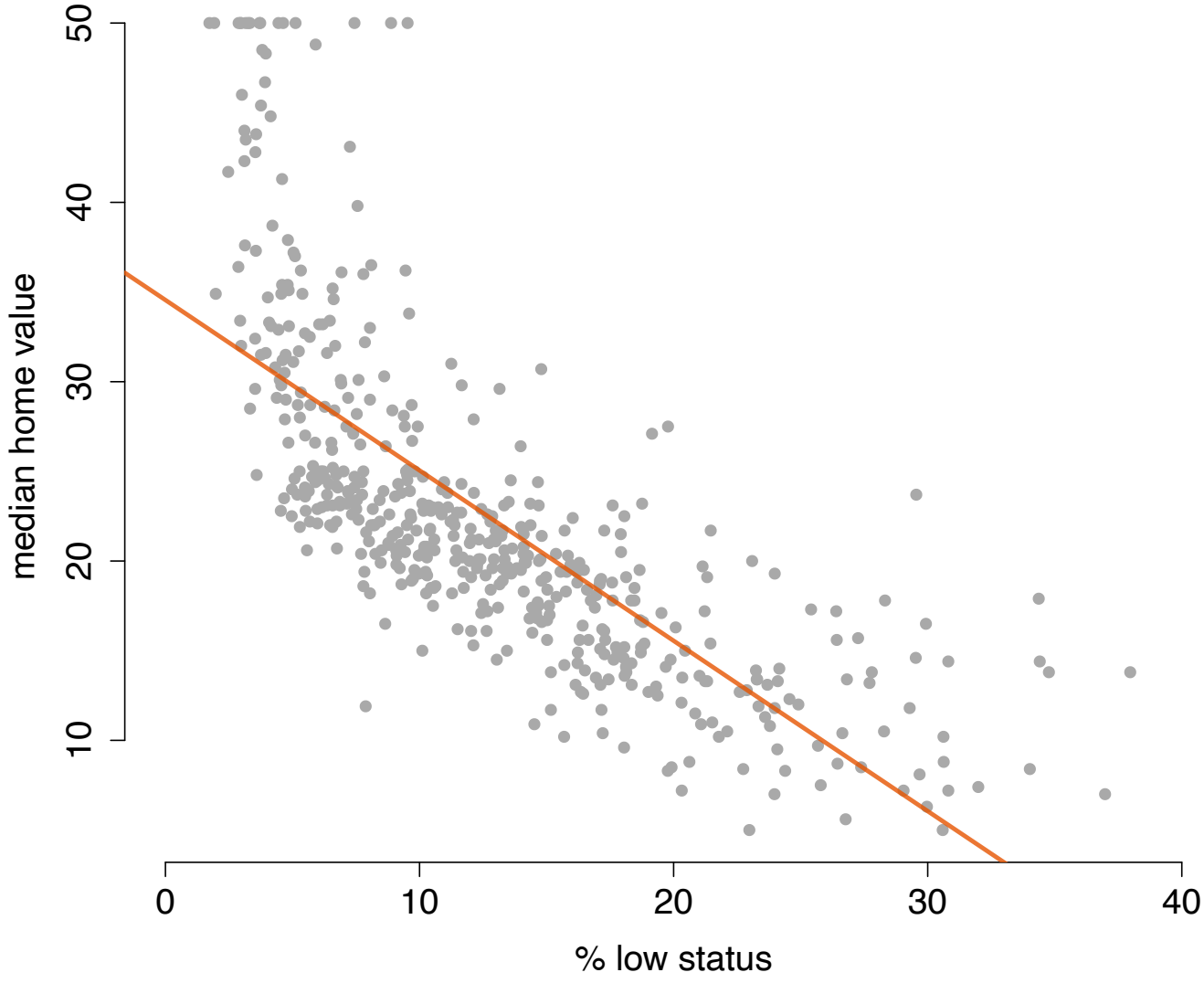
Boston housing data

Prediction at % low status = 30?



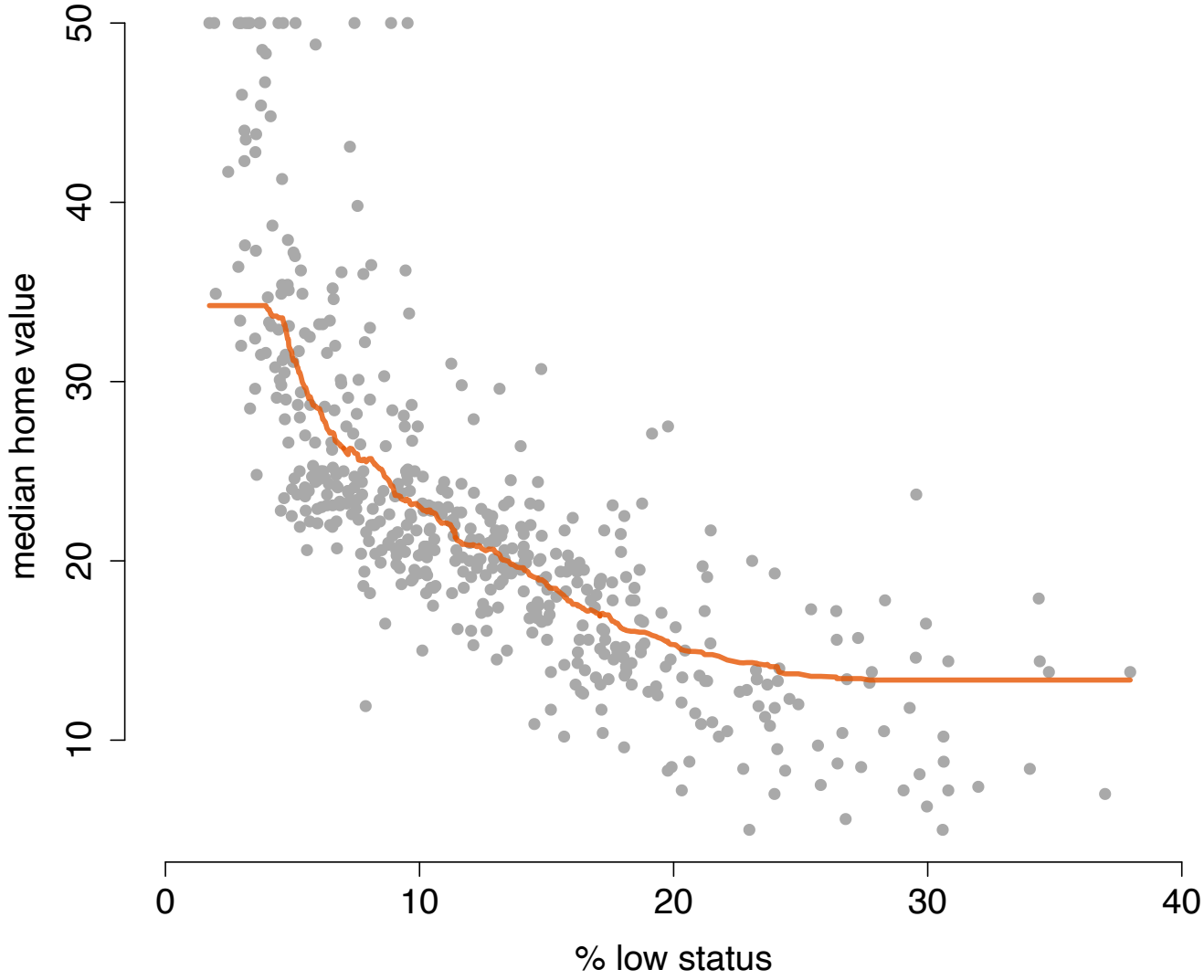
Boston housing data

Prediction at % low status = 30?



Boston housing data

Prediction at % low status = 30?



How do we estimate $f(\cdot)$?

1. Choose set of training data: $(Y_1, X_1), \ldots, (Y_N, X_N)$.

How do we estimate $f(\cdot)$?

1. Choose set of training data: $(Y_1, X_1), \ldots, (Y_N, X_N)$.

2. Fit $f(\cdot)$ to training data using:

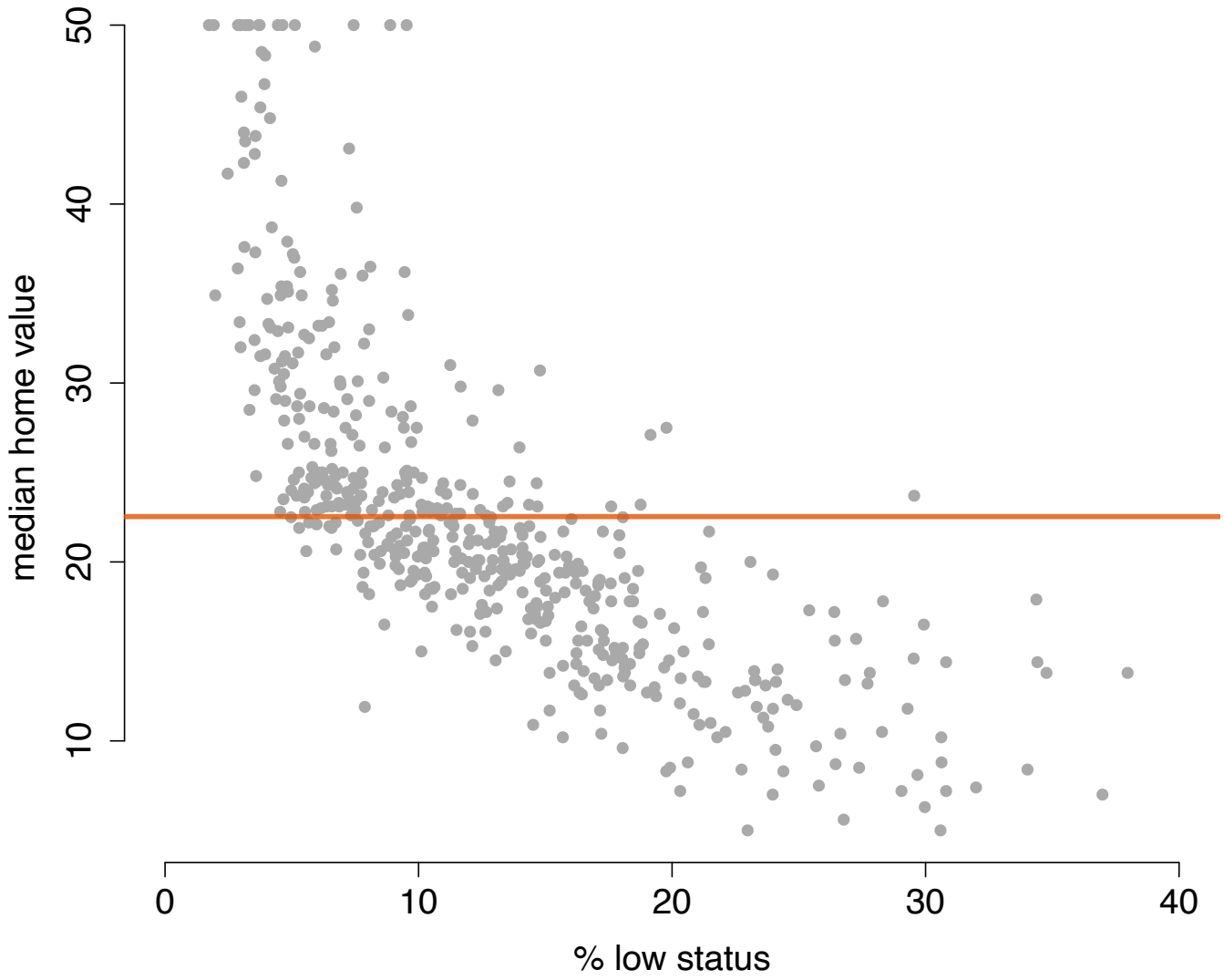
   - Parametric model, or
   - Nonparametric model

How do we estimate $f(\cdot)$?

1. Choose set of training data: $(Y_1, X_1), \ldots, (Y_N, X_N)$.
2. Fit $f(\cdot)$ to training data using:
   - Parametric model, or
   - Nonparametric model
3. Evaluate performance on testing data and *adjust*.

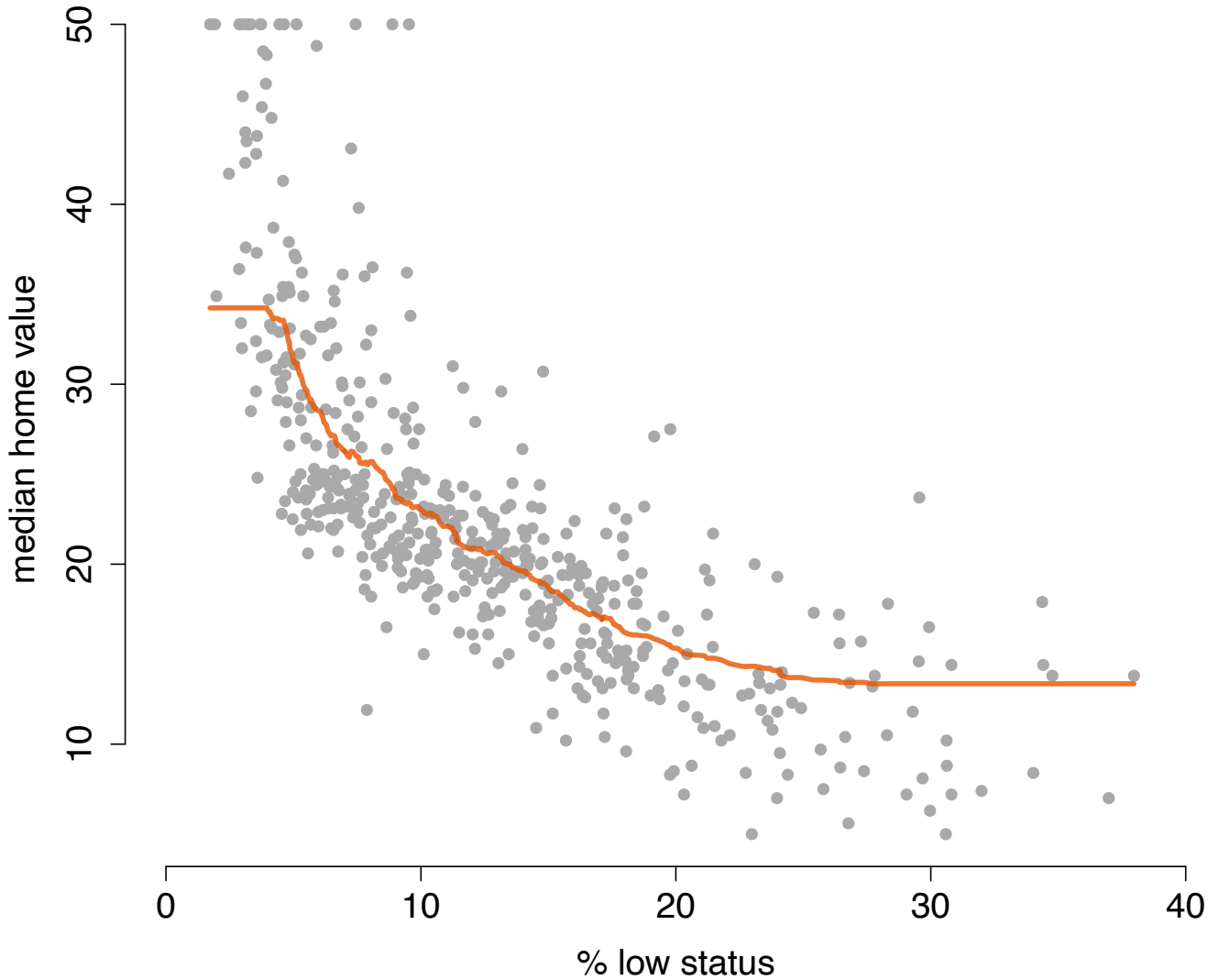
How do we estimate $f(\cdot)$?

Parametric: $Y = \mu + \epsilon$. restrictive assumptions, but simple interpretation.



How do we estimate $f(\cdot)$?

Nonparametric: "Knn" with $k = 100$. flexible assumptions, but complex interpretation.



The challenge when estimating predictions $\widehat{f(\cdot)}$

Balancing *restrictiveness* of assumptions with simplicity of *interpretation*.

Let's look at k-nearest-neighbors (knn)

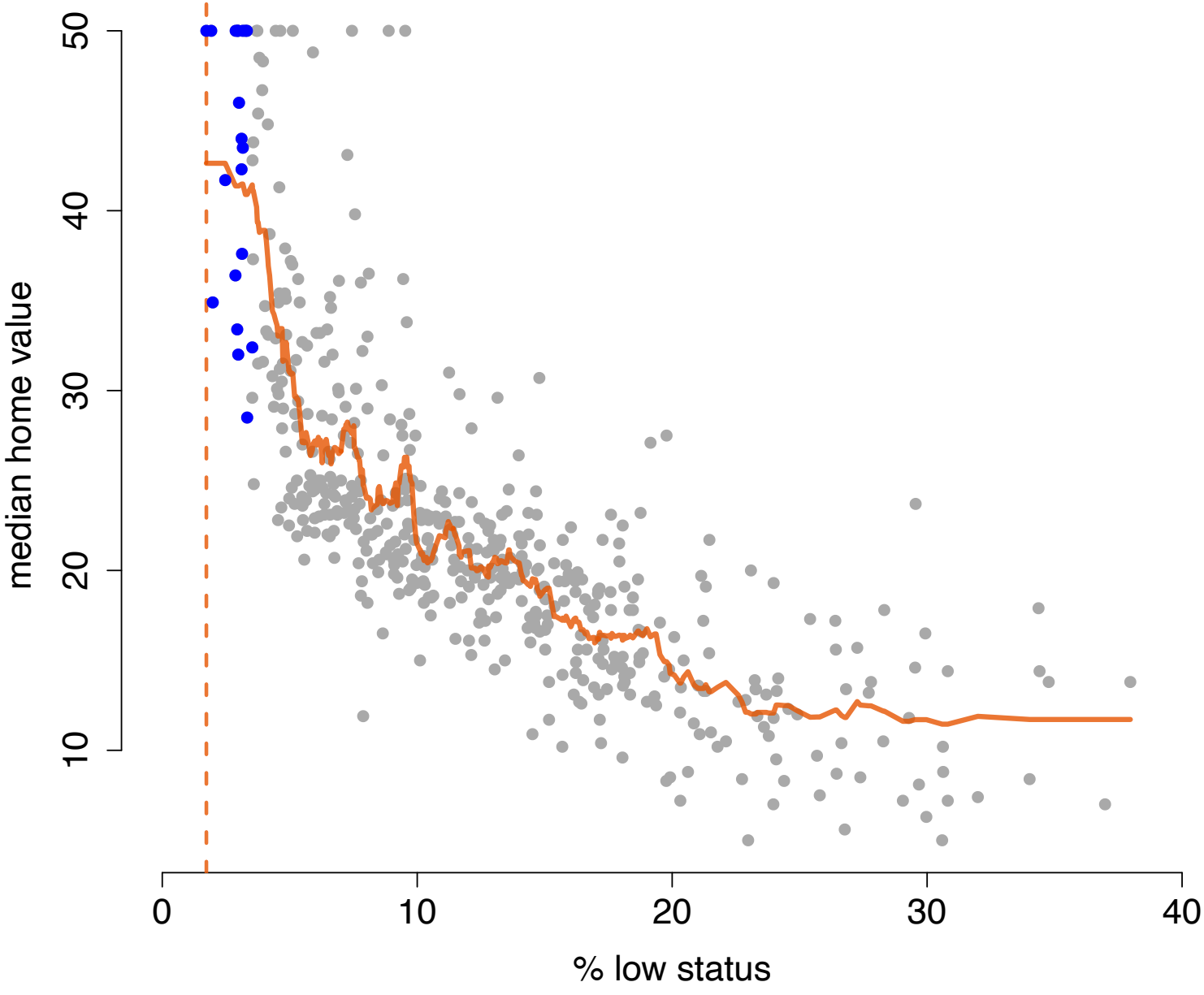
Prediction at point x, $\widehat{f(x)}$ = average of k nearest points around x.

Let's look at k-nearest-neighbors (knn)

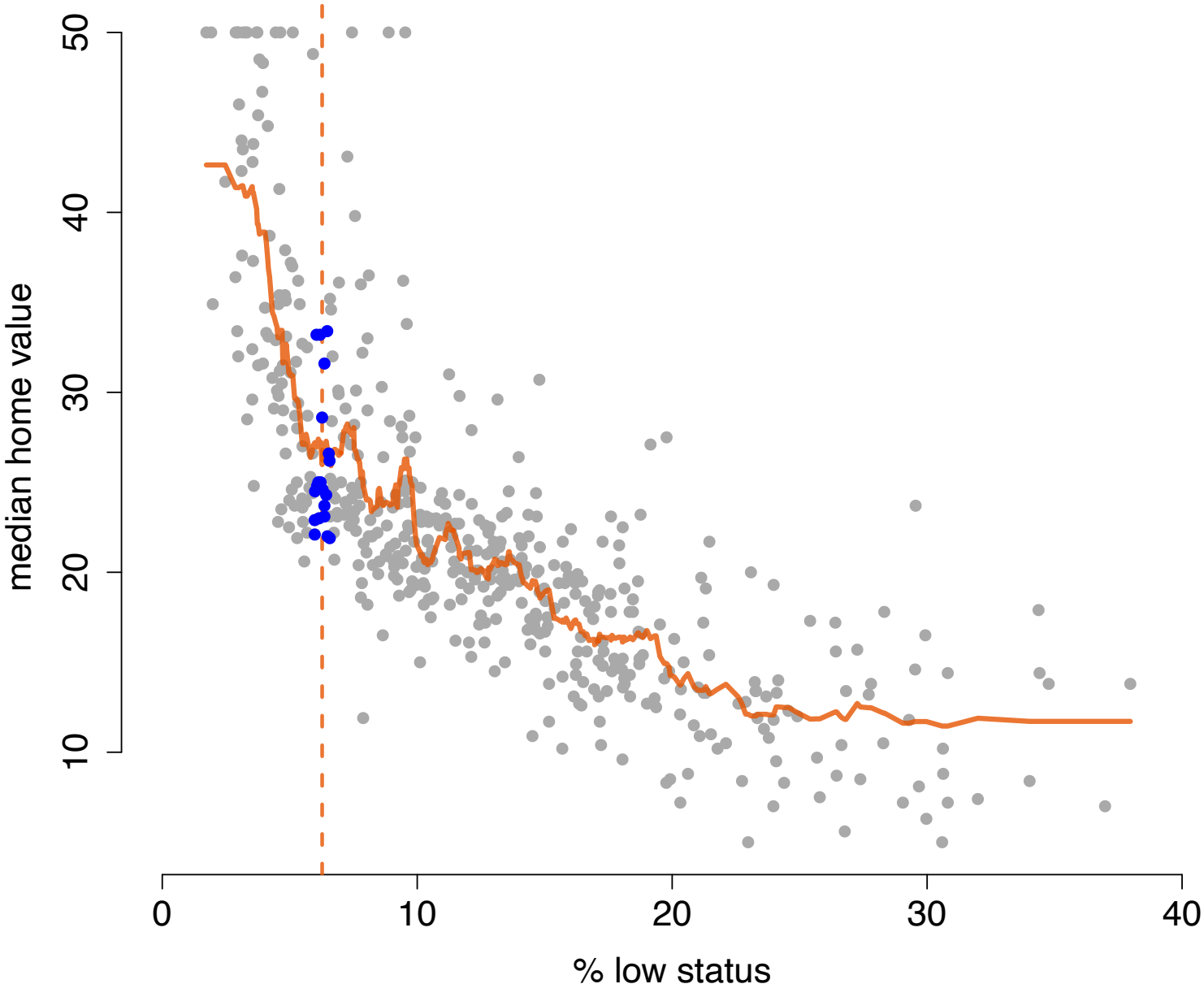
Prediction at point x, $\widehat{f(x)}$ = average of k nearest points around x.

Let's look at $k = 20$ ...

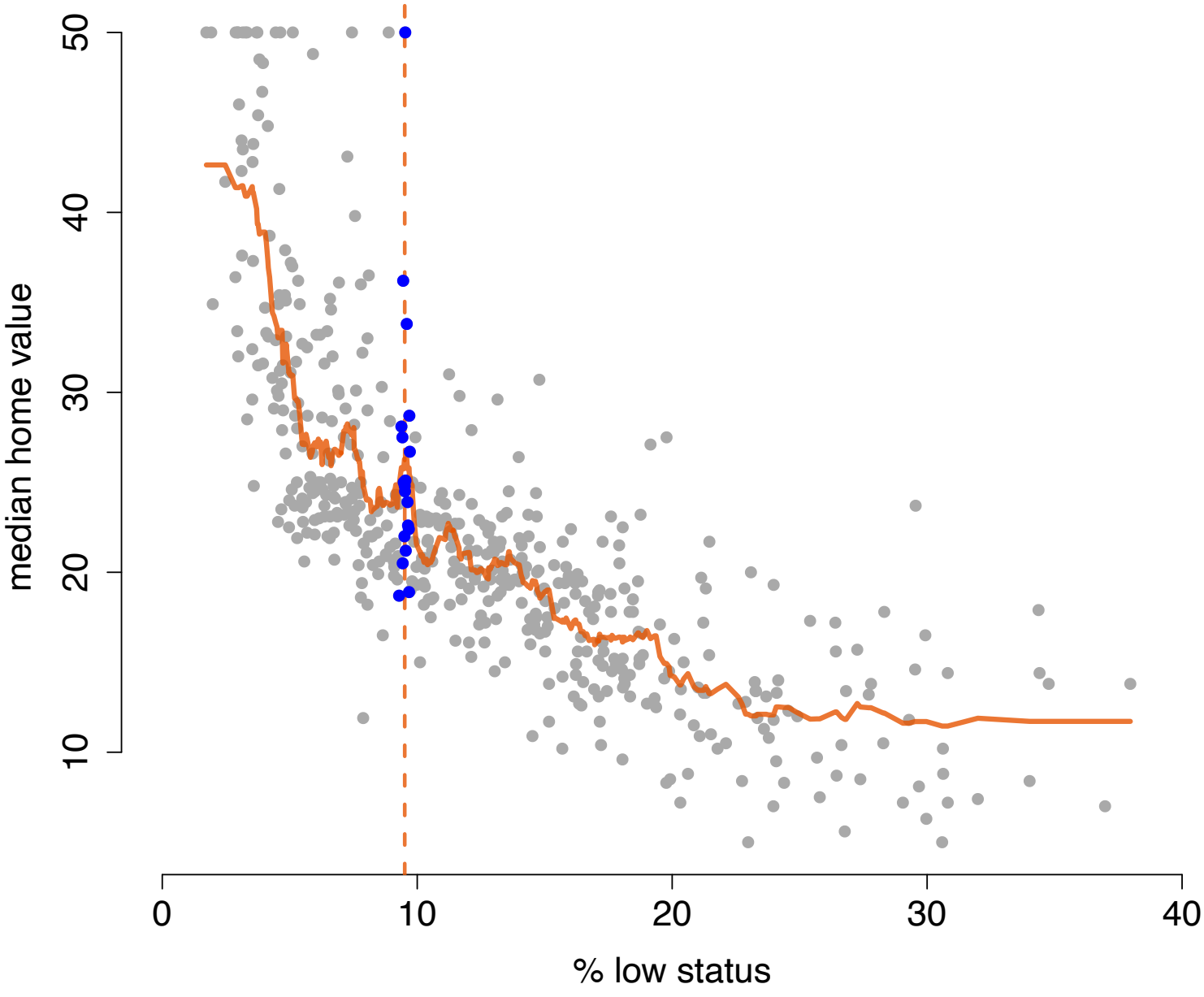
knn with $k = 20$



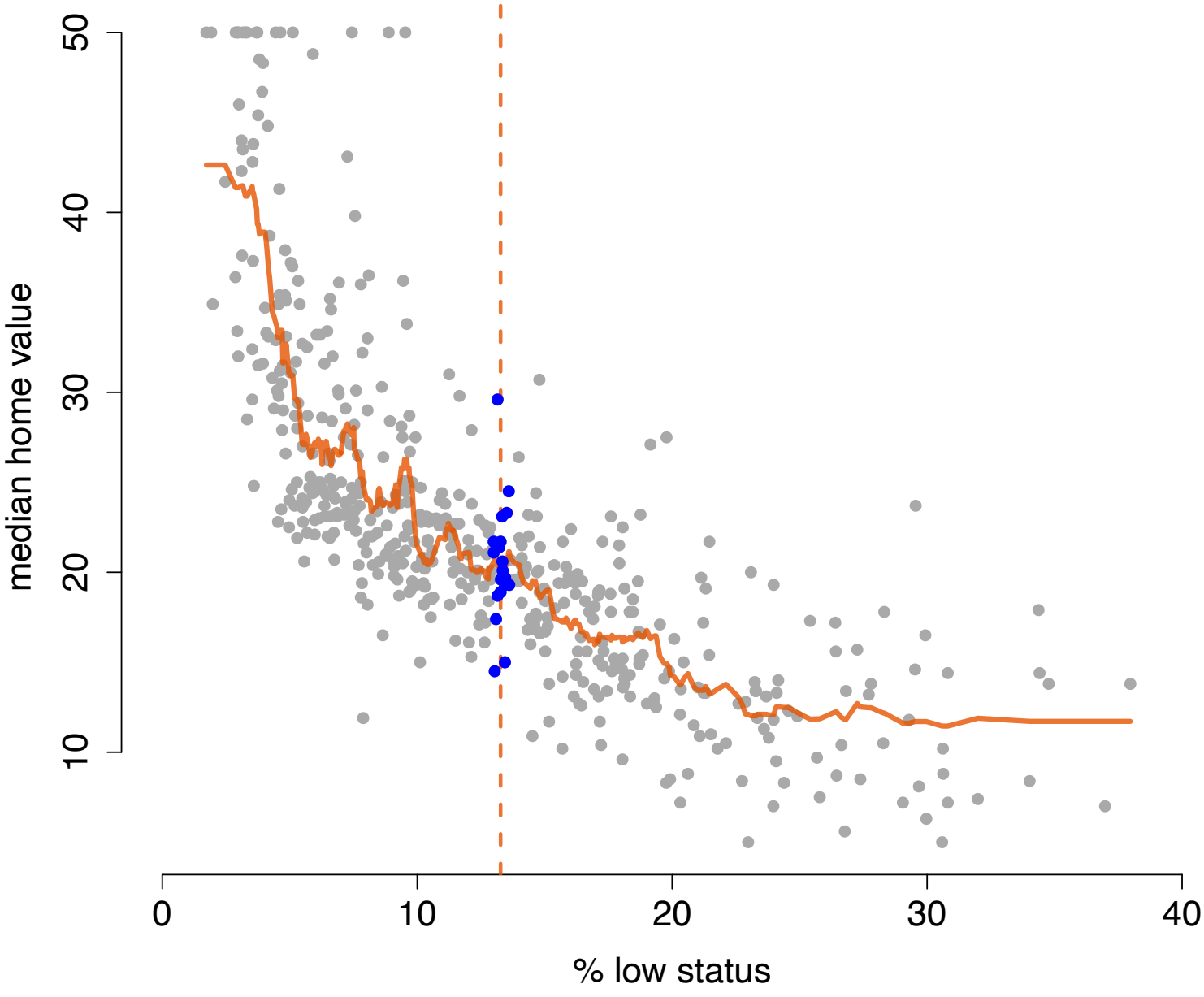
knn with $k = 20$



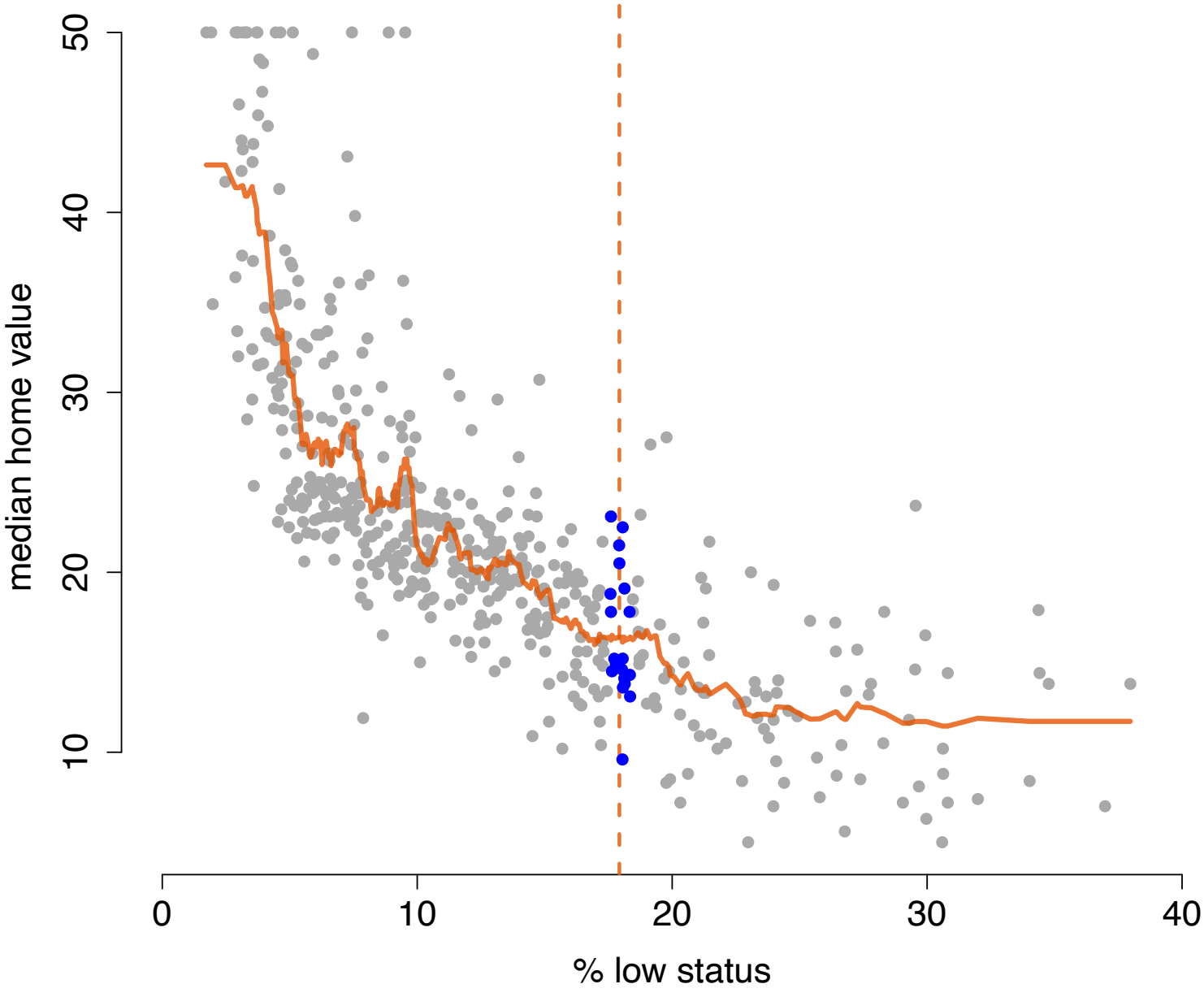
knn with $k = 20$



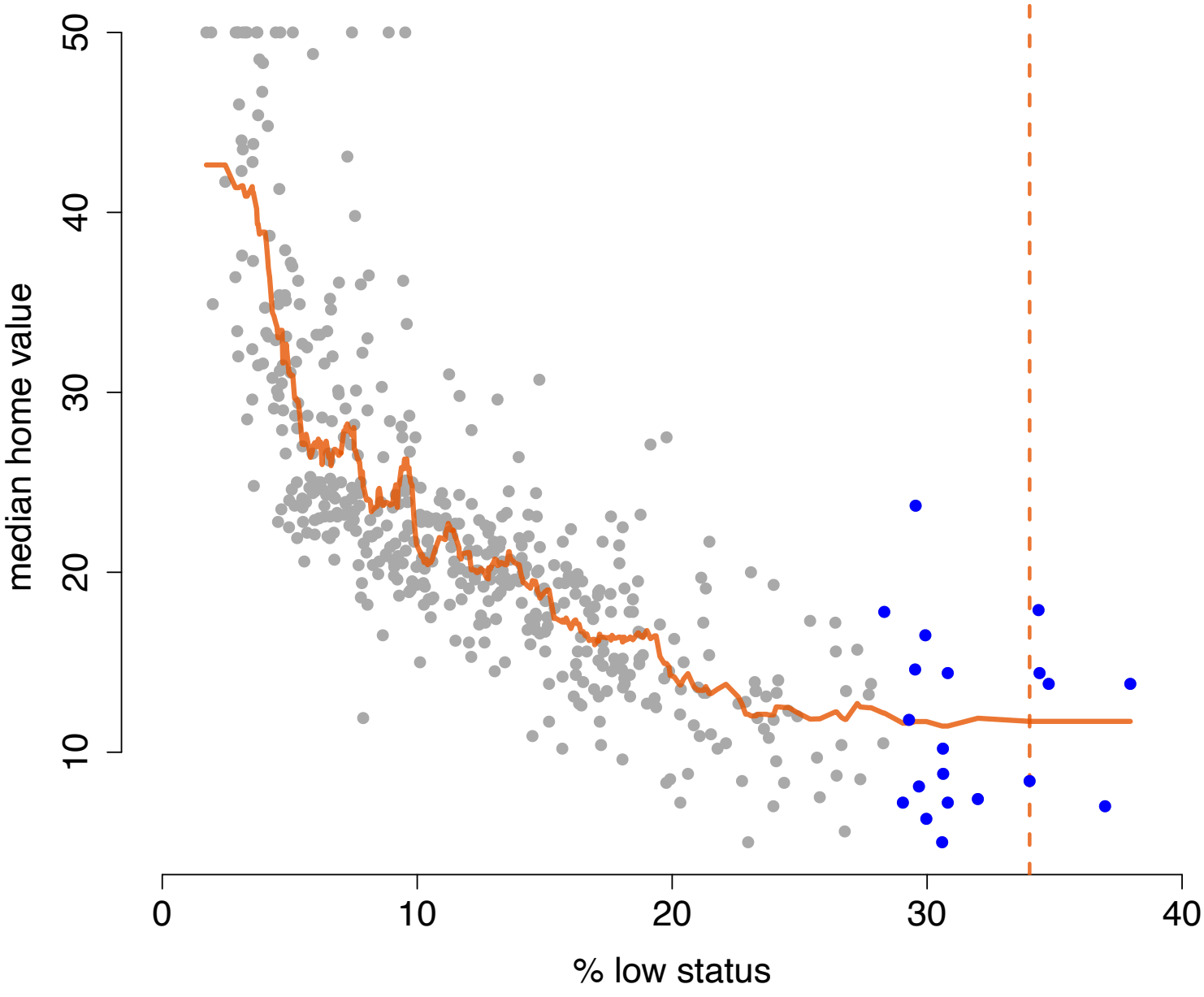
knn with $k = 20$



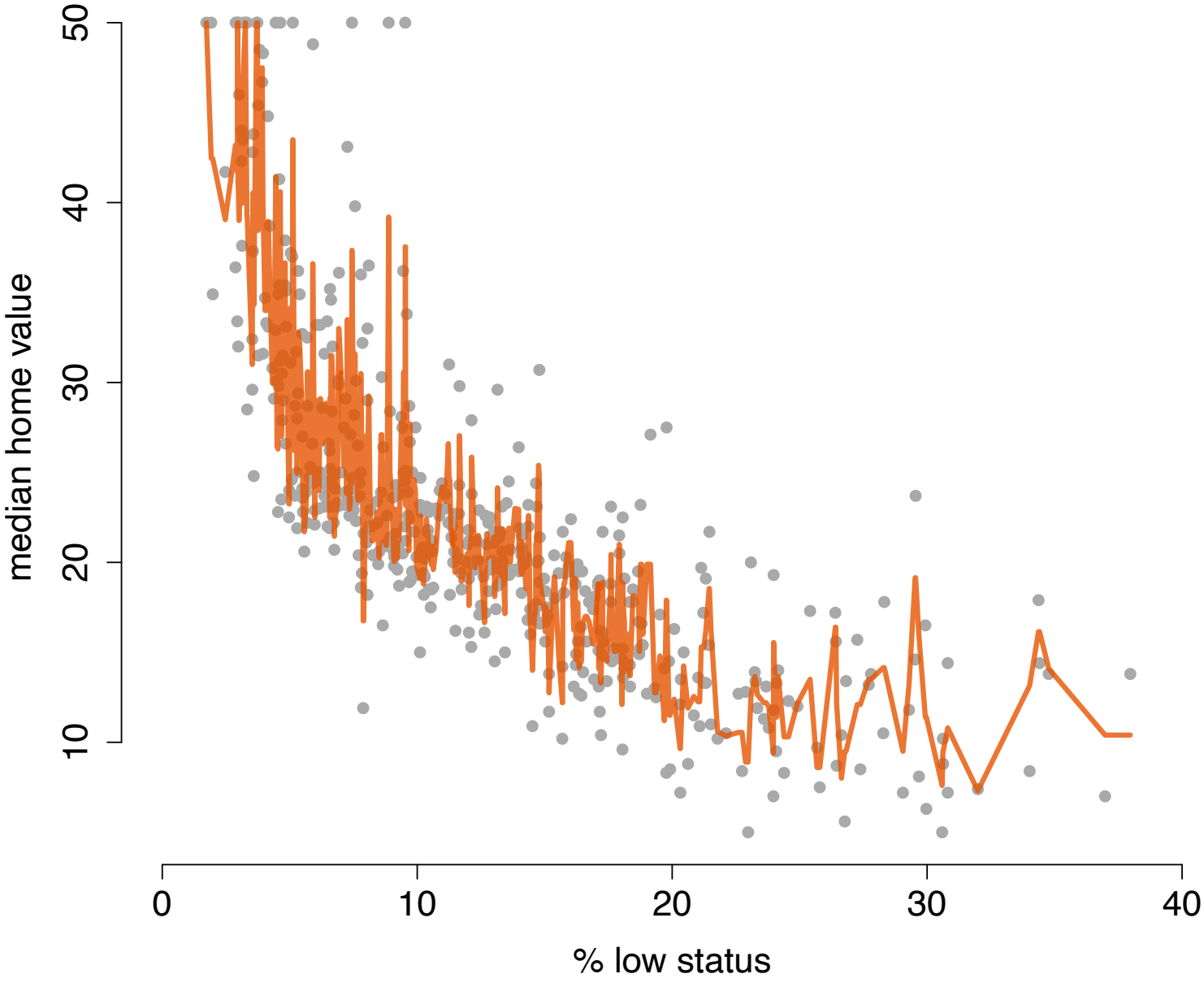
knn with $k = 20$



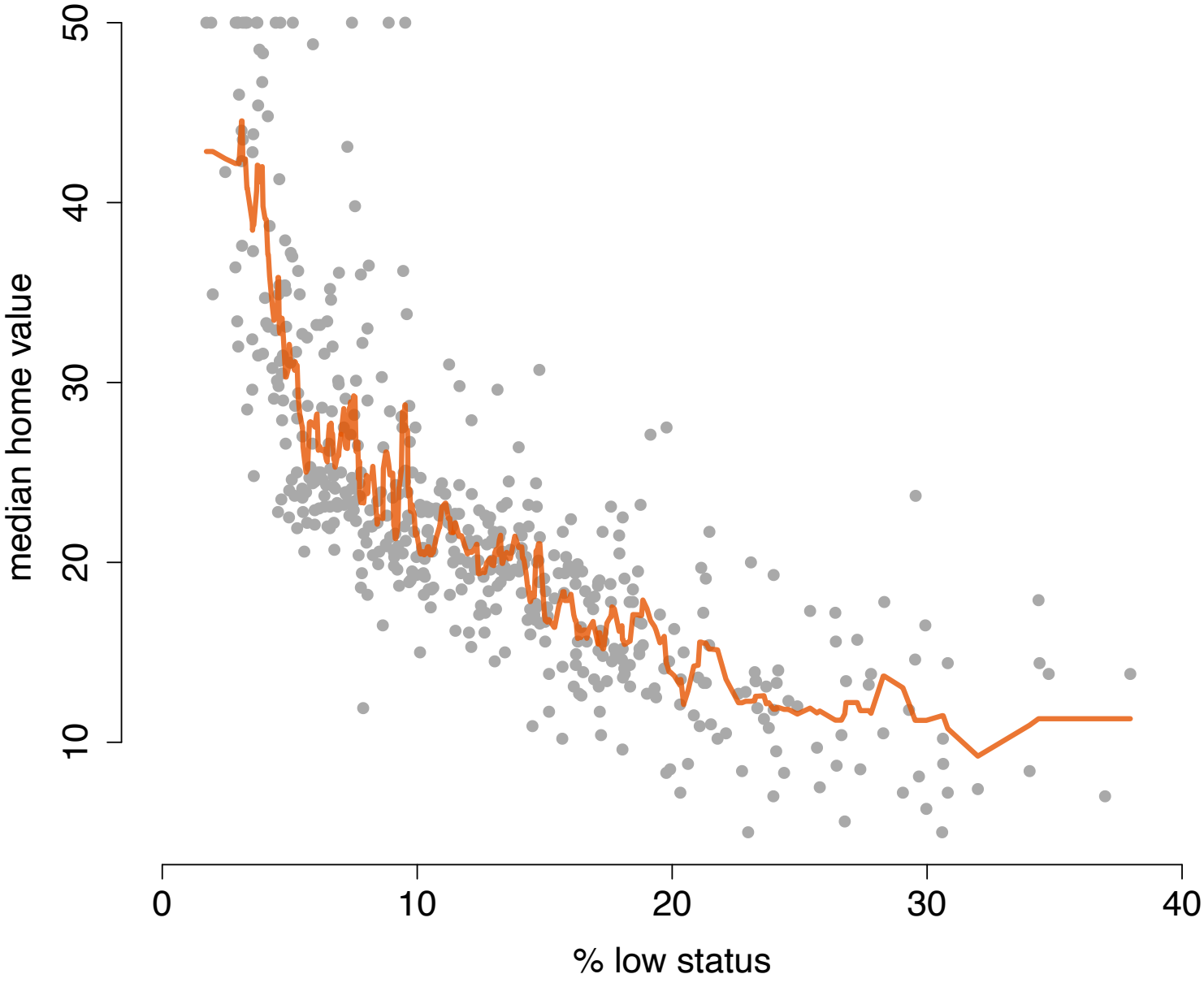
knn with $k = 20$



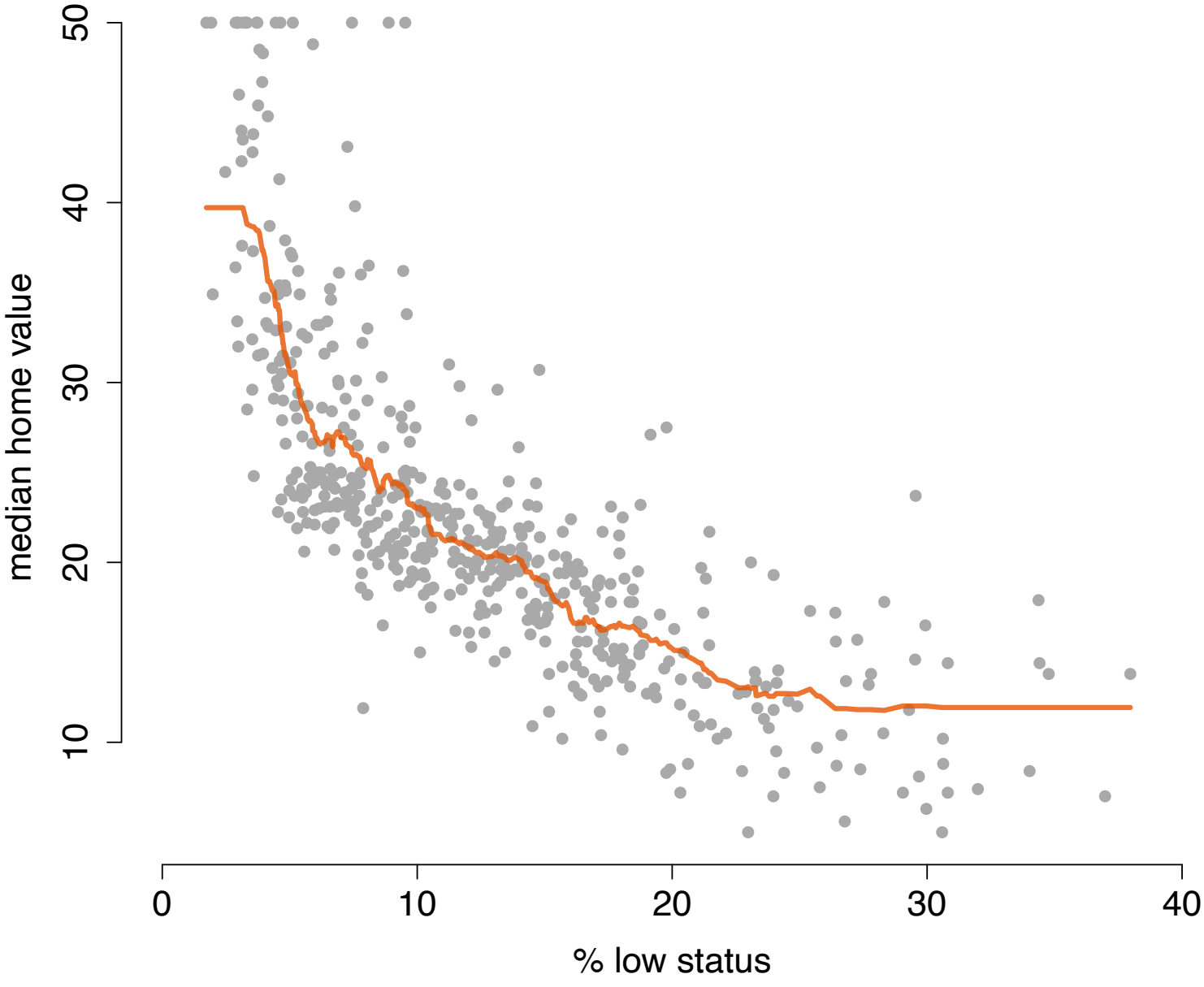
Why don't I choose $k = 2$ instead?



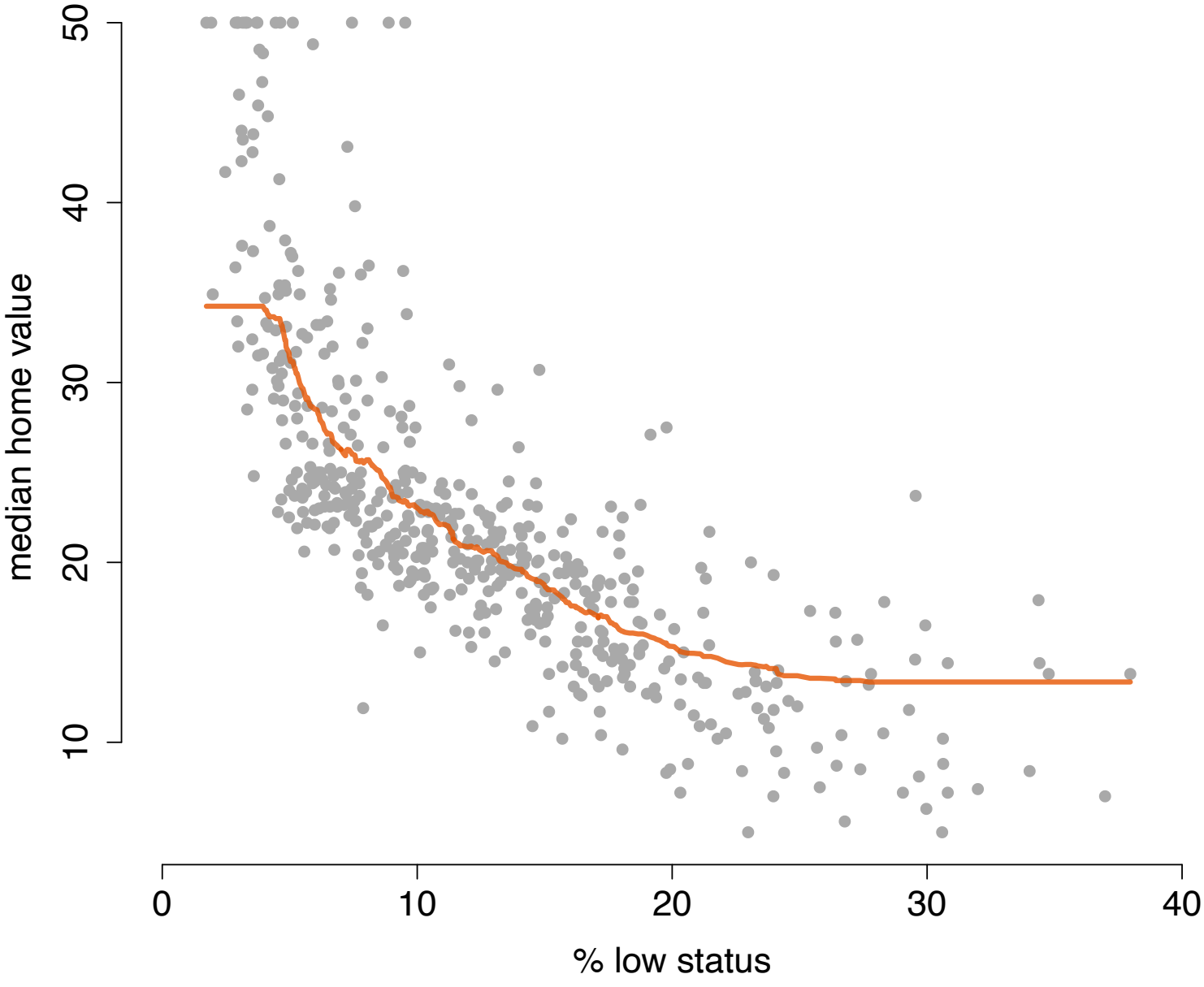
or $k = 10$ ...



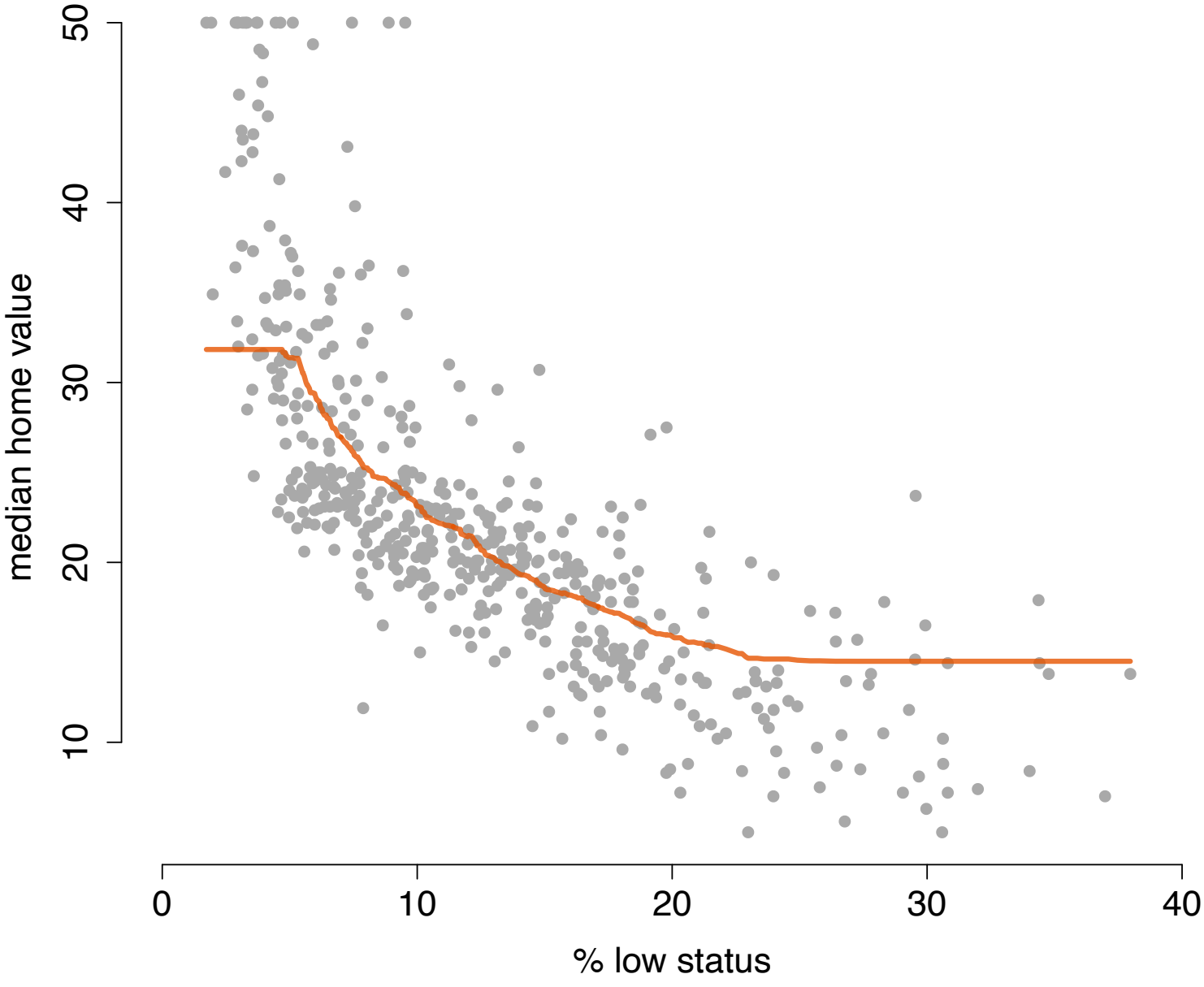
or $k = 50$ ...



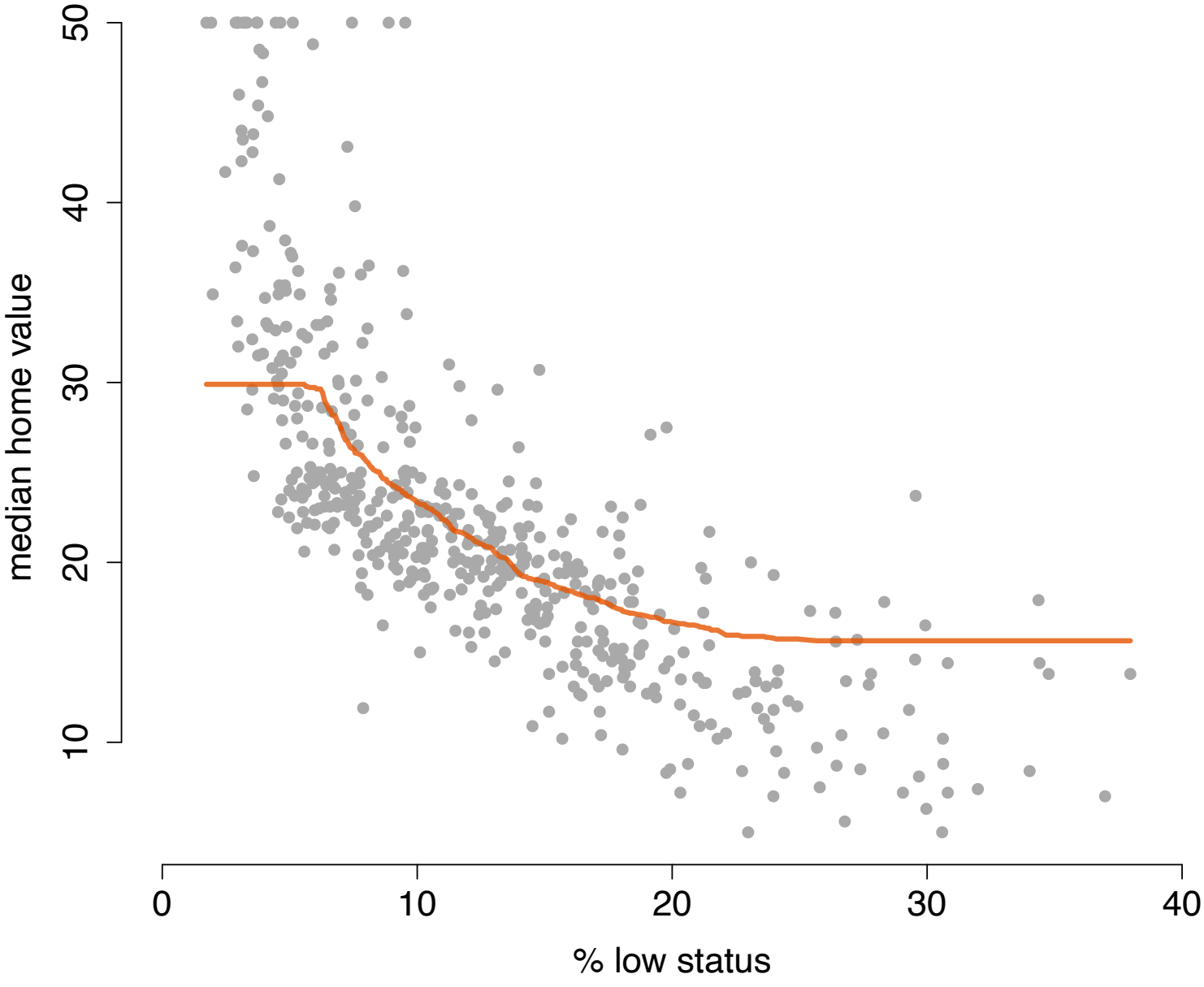
or $k = 100$ ...



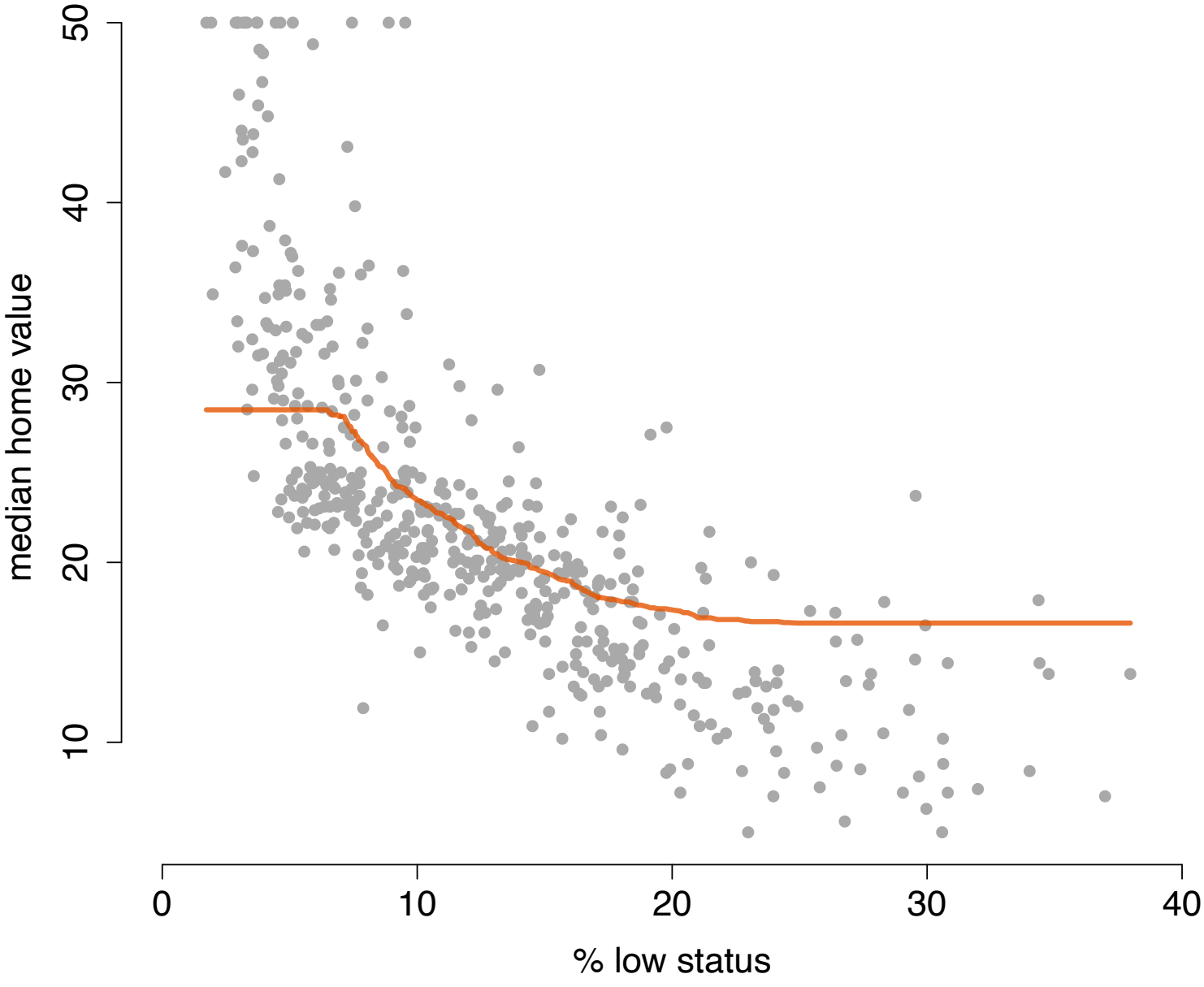
or $k = 150$ ...



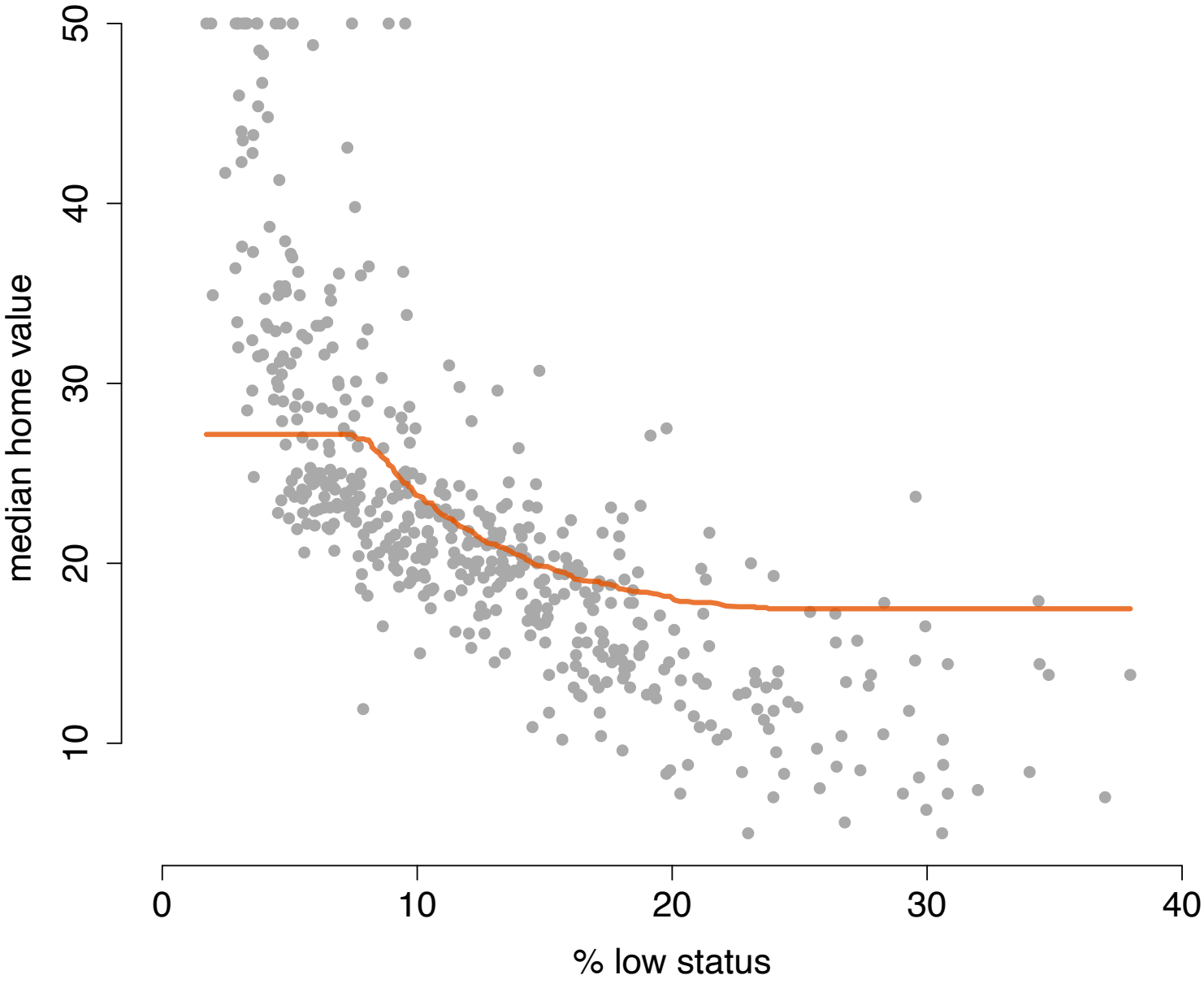
or $k = 200$ ...



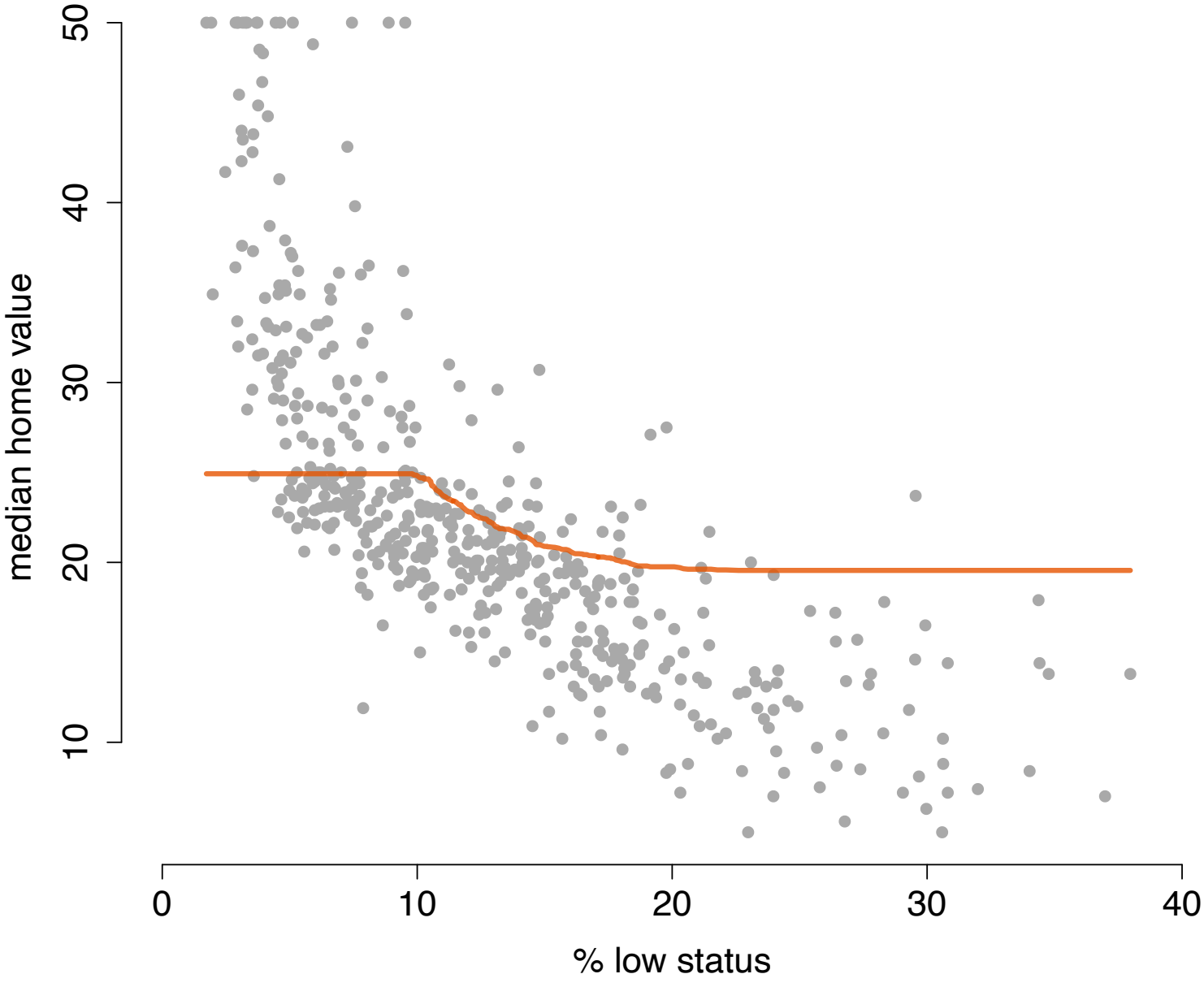
or $k = 250$ ...



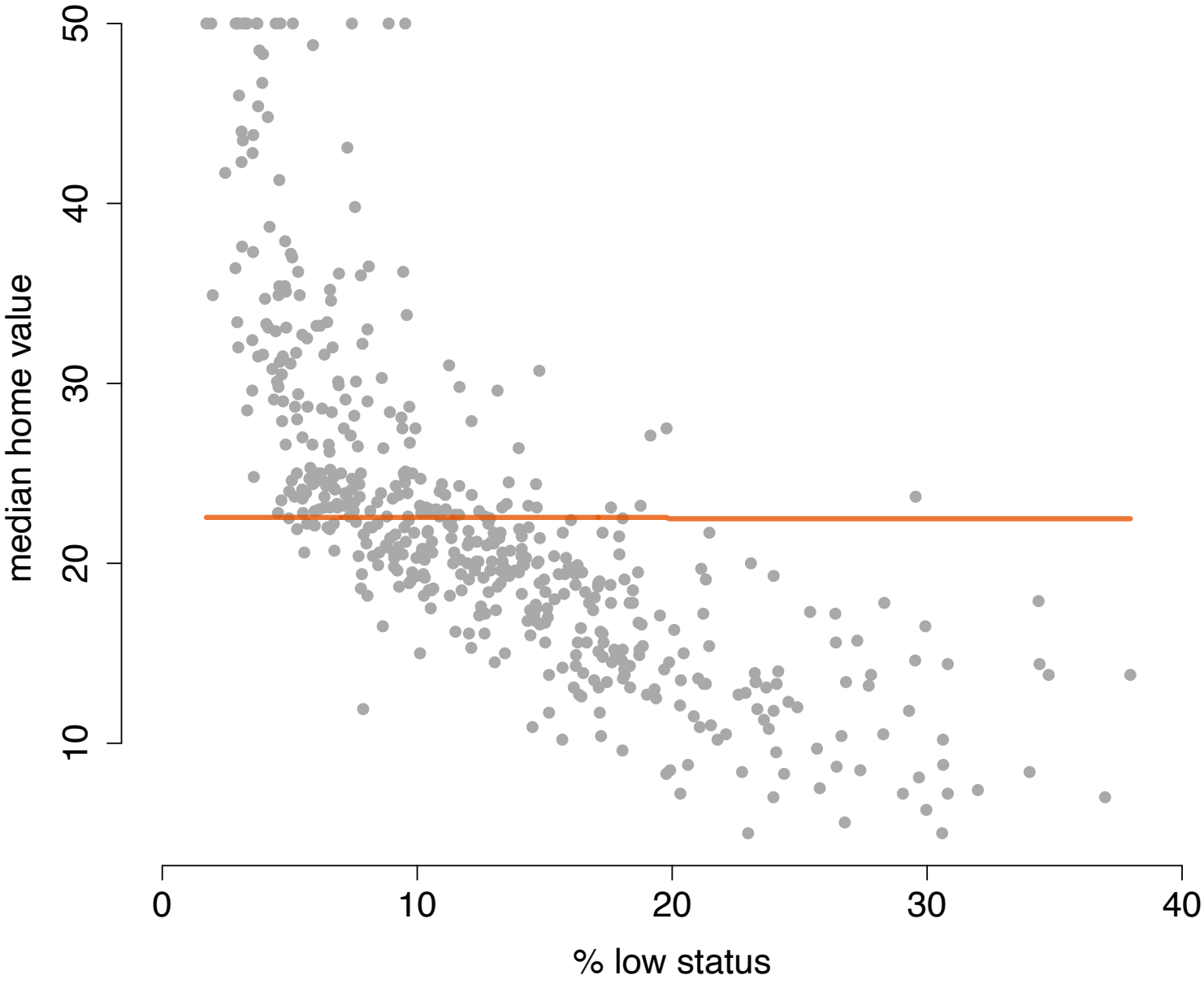
Or $k = 300$ ...



or $k = 400$ ...



or $k = 505$ ...



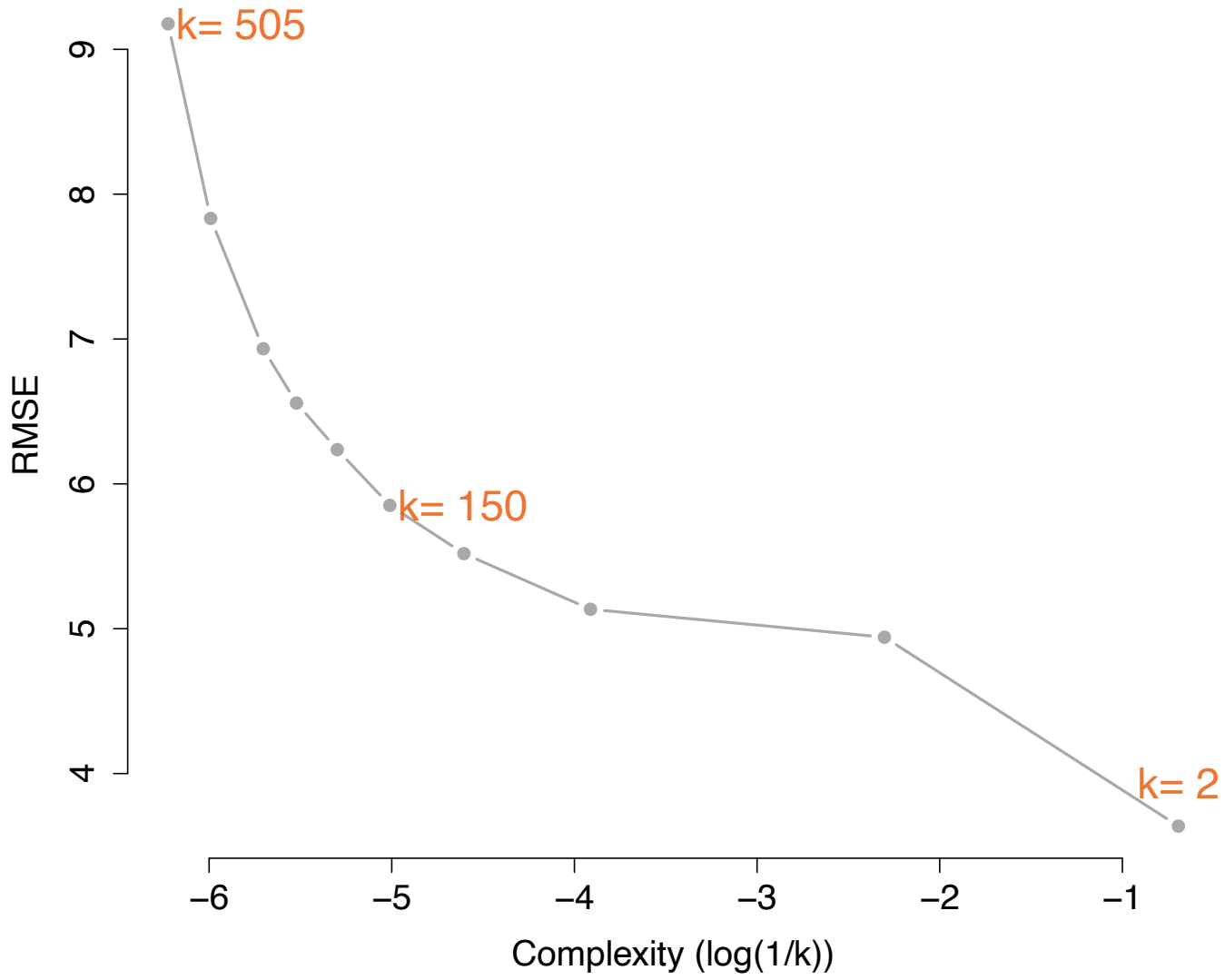
A rigorous way to select

– The root mean squared error measures how accurate my predictions are, on average.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left[Y_i - \widehat{f(X_i)}\right]^2}$$

In sample RMSE

It looks like $k = 2$ is the best. Should we choose this model?



We care about out of sample performance

- Suppose we have m additional observations (X_i^o, Y_i^o), for $i = 1, \ldots, m$, that we did not use to fit the model. Let's call this dataset the *validation set* (a.k.a *hold-out set* or *test set*)

We care about out of sample performance

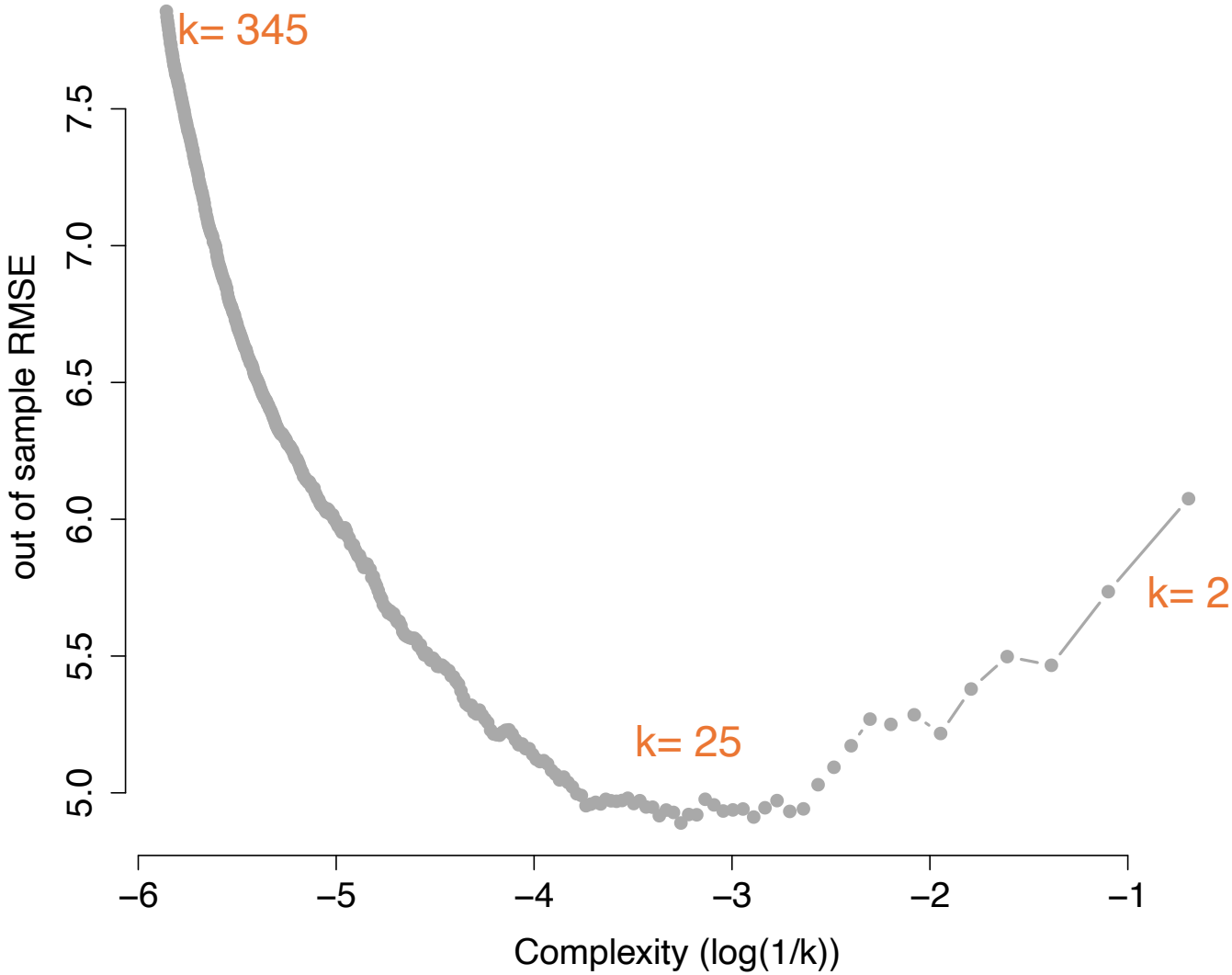
- Suppose we have m additional observations (X_i^o, Y_i^o), for $i = 1, \ldots, m$, that we did not use to fit the model. Let's call this dataset the *validation set* (a.k.a *hold-out set* or *test set*)

- We evaluate the fit with out of sample RMSE:

$$RMSE^o = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left[Y_i^o - \widehat{f(X_i^o)}\right]^2}$$

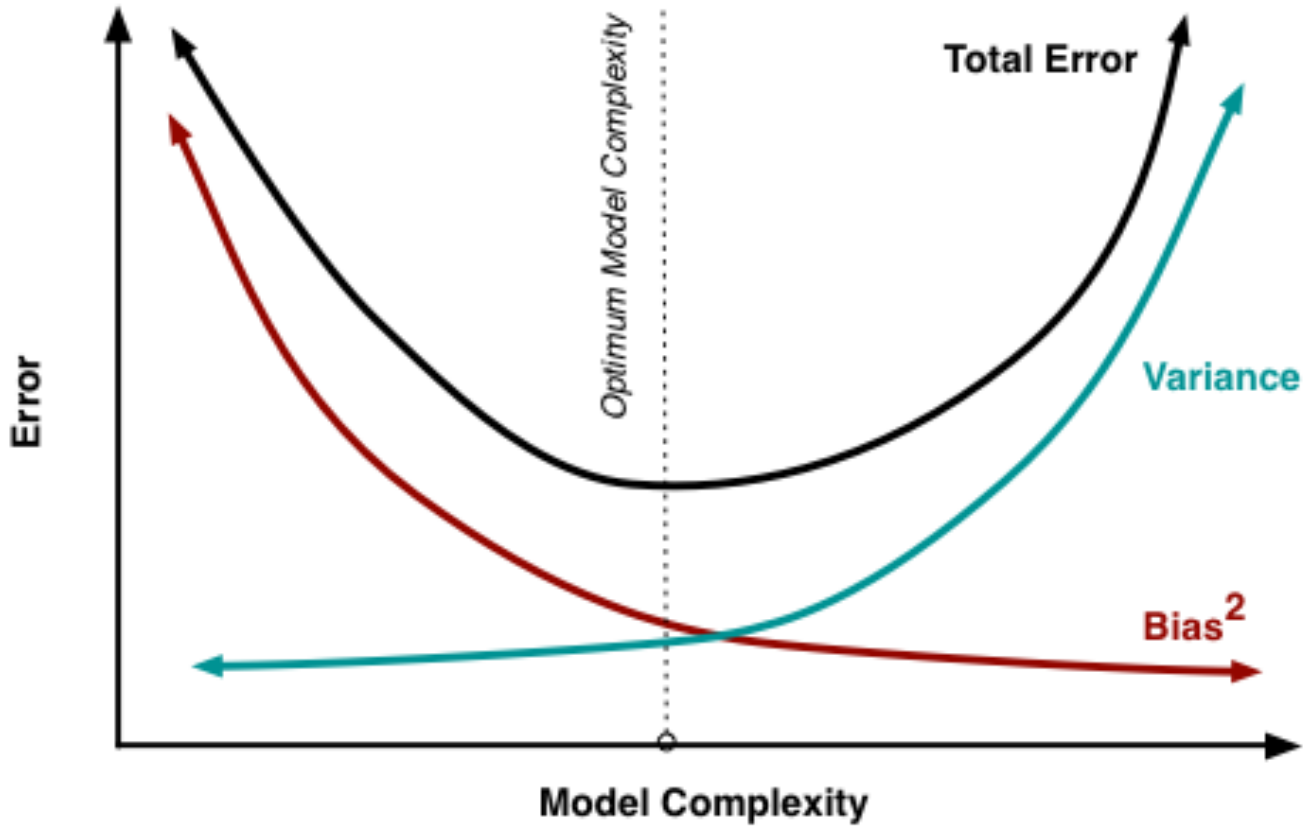
Out of sample RMSE

Fit each model on training set of size 400. Test each model (*out of sample*) on testing set of size 106. Here, we plot the out of sample performance.



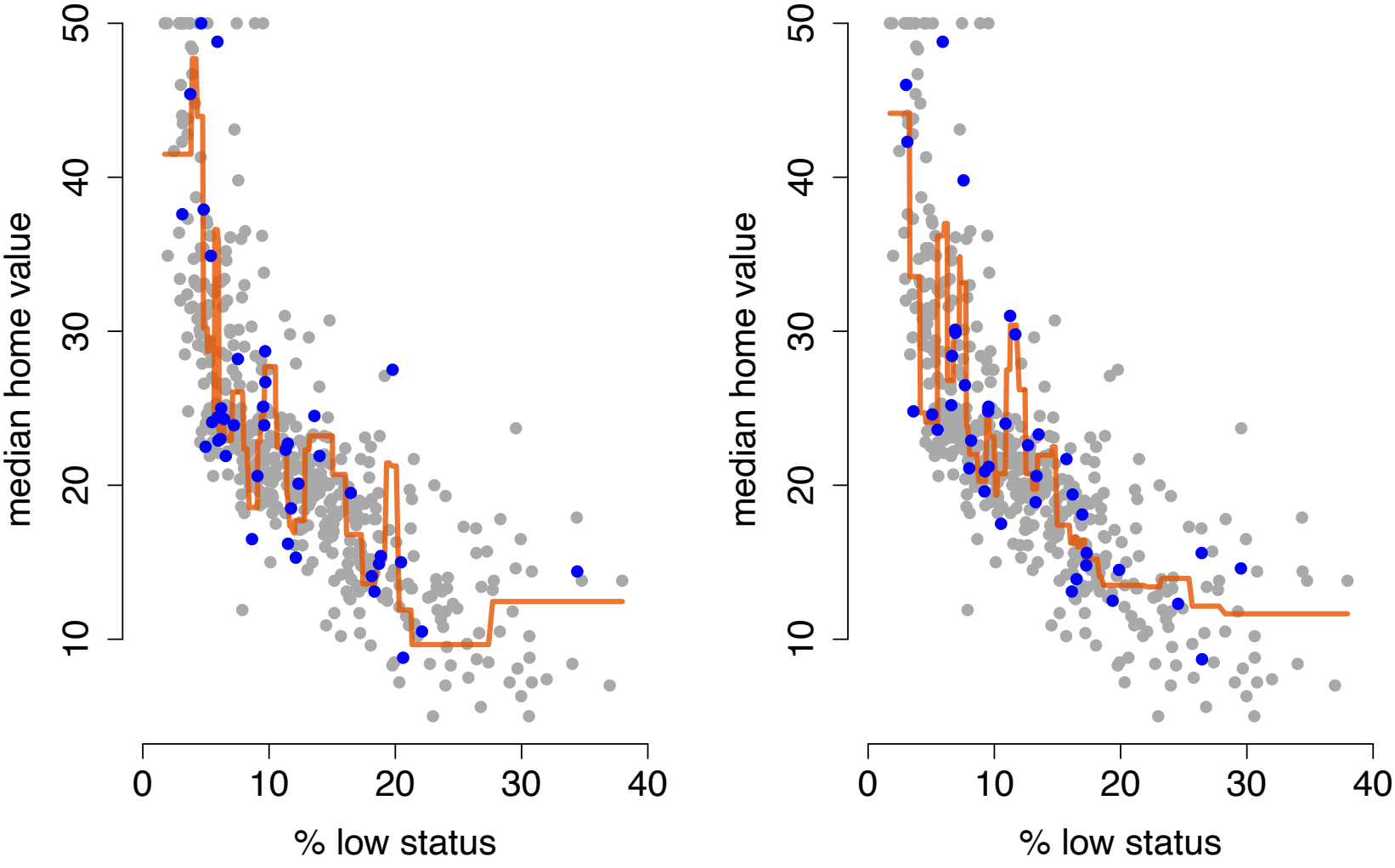
The Bias-variance tradeoff!

When fitting a predictive model, there is a tradeoff between bias and variance of predictions.



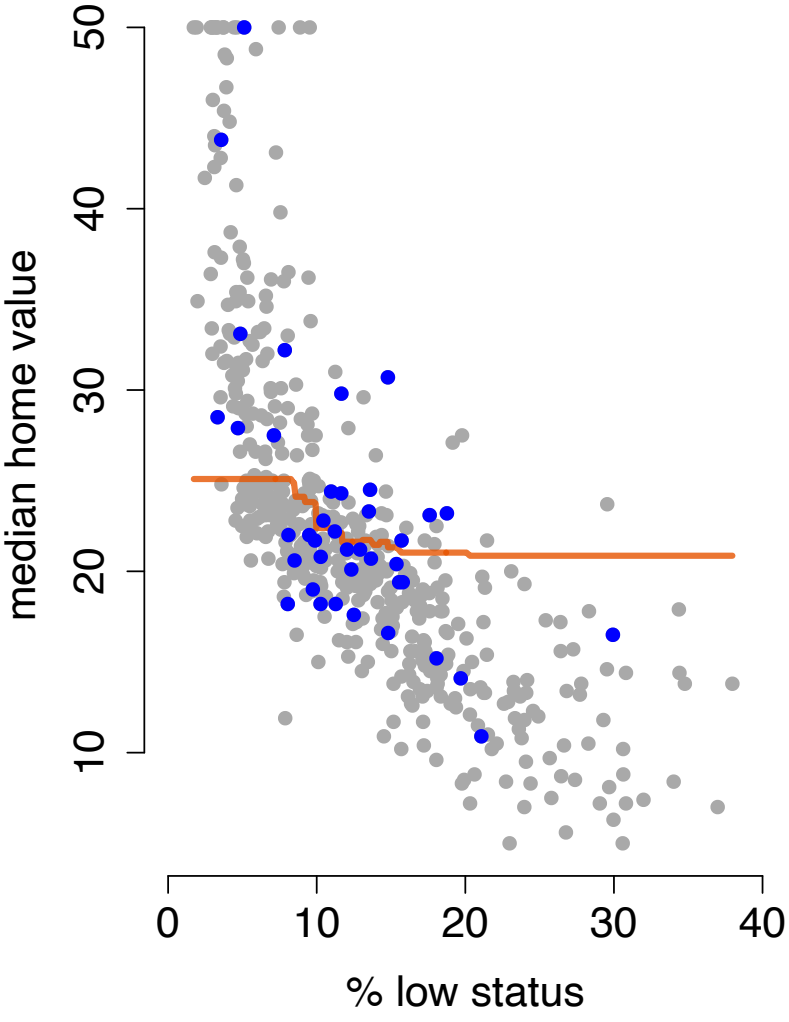
$k = 2$: low bias, high variance

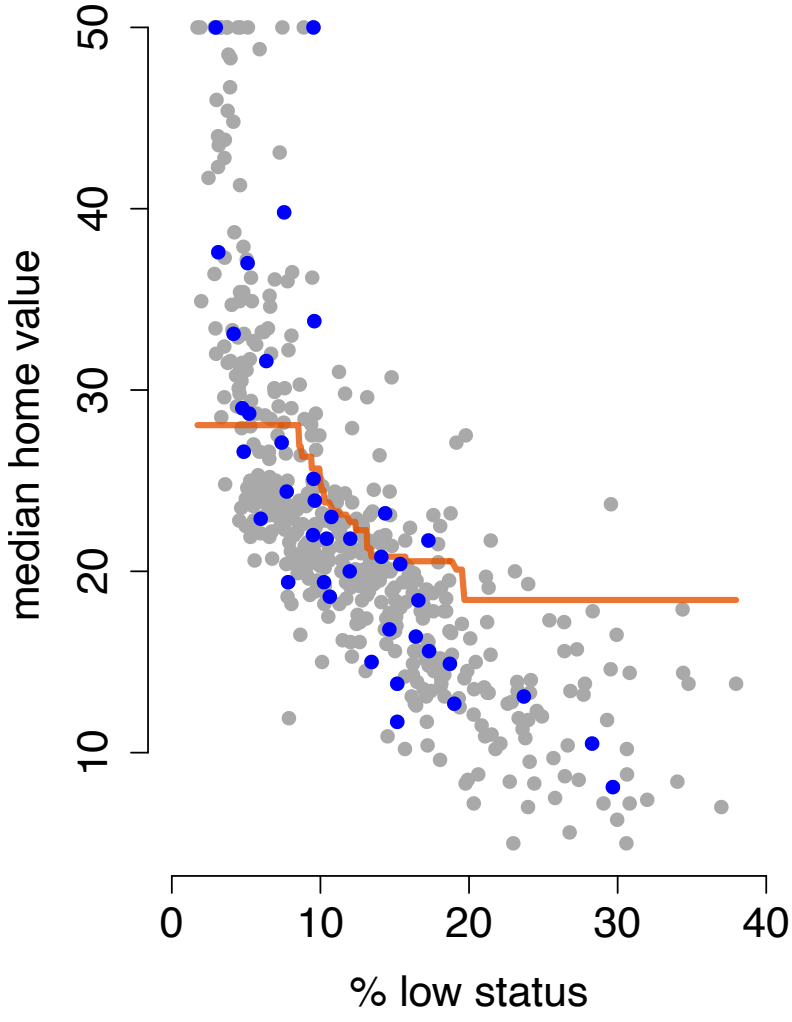
Training set of size 40.



$k = 25$: high bias, low variance

Training set of size 40.





Relationship to linear regression

Selecting k is Knn is the same as selecting which variables to include in your regression model!

In both cases, you are trying to build the best model for your outcome Y.

Questions that remain unanswered:

$\rightarrow$ How does model selection relate to causal inference?

$\rightarrow$ More directly, how can we use the best ideas from machine learning to help us automatically **control for** the variables we need in our model?

Where we're headed next

First, we will overview model selection methods for linear regression.

Second, we will discuss tree-based modeling (and the implicit selection that goes on), one of the most popular nonlinear modeling techniques available.

In both cases, the goal is to include all measured covariates in the model and let the statistical method select which ones are the most important.

Linear models and the B-V tradeoff

Variable selection and regularization

When working with linear regression models where the number of X variables is large, we need to think about strategies to select what variables to use...

We will focus on 2 ideas:

- Subset Selection
- Shrinkage

Subset selection

The idea here is very simple: fit as many models as you can and compare their performance based on some criteria!

Issues:

- How many possible models? Total number of models $= 2^p$
  Is this large?
- What criteria to use?
  Just as before, if prediction is what we have in mind, out-of-sample predictive ability should be the criteria

Information criteria

Another way to evaluate a model is to use Information Criteria metrics which attempt to quantify how well our model would have predicted the data (regardless of what you've estimated for the β_j's).

A good alternative is the BIC: Bayes Information Criterion, which is based on a "Bayesian" philosophy of statistics.

$$BIC = n \log(s^2) + p \log(n)$$

You want to choose the model that leads to minimum BIC.

Information criteria

One nice thing about the BIC is that you can interpret it in terms of model probabilities.

Given a list of possible models $\{M_1, M_2, \ldots, M_R\}$, the probability that model i is correct is

$$P(M_i) \approx \frac{e^{-\frac{1}{2}BIC(M_i)}}{\sum_{r=1}^{R} e^{-\frac{1}{2}BIC(M_r)}} = \frac{e^{-\frac{1}{2}[BIC(M_i)-BIC_{min}]}}{\sum_{r=1}^{R} e^{-\frac{1}{2}[BIC(M_r)-BIC_{min}]}}$$

(Subtract $BIC_{min} = \min\{BIC(M_1) \ldots BIC(M_R)\}$ for numerical stability.)

Similar, alternative criteria include AIC, C_p, adjusted R^2...

Bottom line: these are only useful if we lack the ability to compare models based on their out-of-sample predictive ability!

Search strategies: Stepwise regression

One computational approach to build a regression model step-by-step is "stepwise regression" There are 3 options:

- Forward: adds one variable at the time until no remaining variable makes a significant contribution (or meet a certain criteria... could be out of sample prediction)
- Backwards: starts will all possible variables and removes one at the time until further deletions would do more harm them good
- Stepwise: just like the forward procedure but allows for deletions at each step

Shrinkage methods

An alternative way to deal with selection is to **work with all p predictors at once** while placing a constraint on the size of the estimated coefficients

This idea is a regularization technique that reduces the variability of the estimates and tend to lead to better predictions.

The hope is that by having the constraint in place, the estimation procedure will be able to focus on "the important β's"

Ridge regression

Ridge regression is a modification of the least squares criteria that minimizes (as a function of β's)

$$\sum_{i=1}^{n}\left(Y_i - \beta_0 - \sum_{j=1}^{p}\beta_j X_{ij}\right)^2 + \lambda\sum_{j=1}^{p}\beta_j^2$$

for some value of $\lambda > 0$

- The "blue" part of the equation is the traditional objective function of LS
- The "red" part is the shrinkage penalty, ie, something that makes costly to have big values for β

Ridge regression

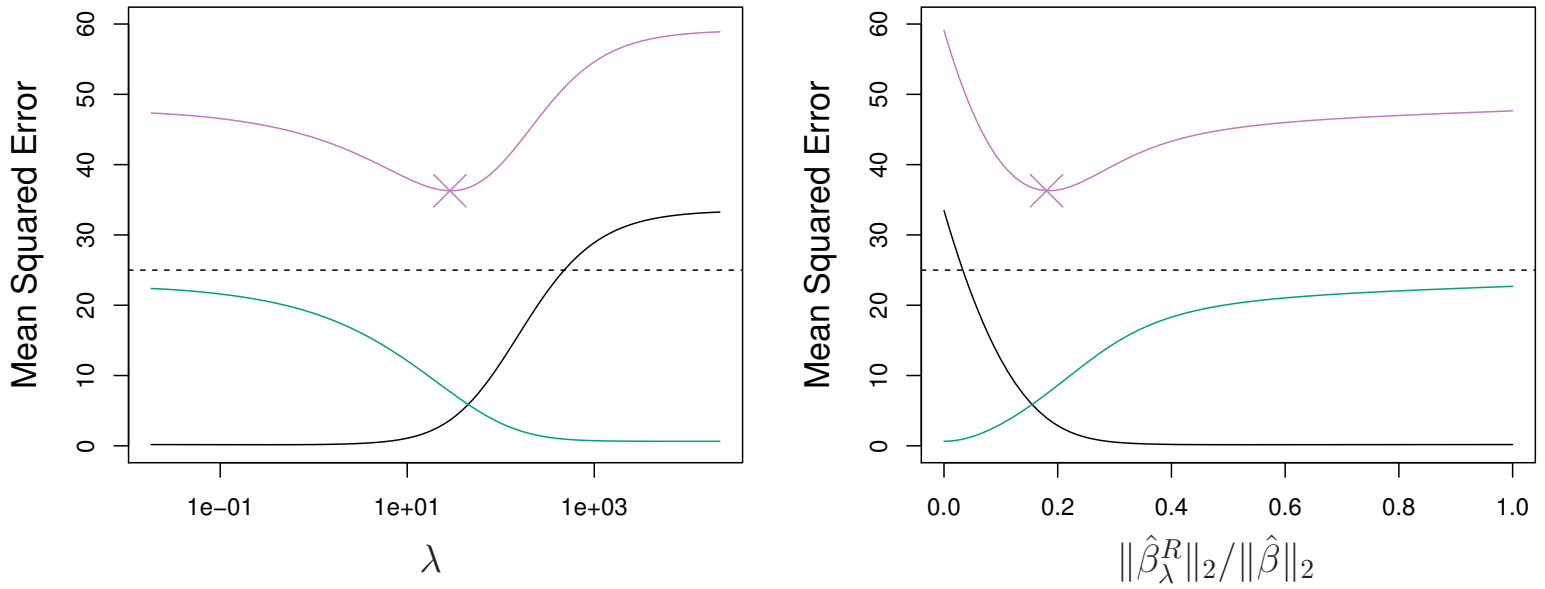
$$\sum_{i=1}^{n}\left(Y_i - \beta_0 - \sum_{j=1}^{p}\beta_j X_{ij}\right)^2 + \lambda\sum_{j=1}^{p}\beta_j^2$$

- if $\lambda = 0$ we are back to least squares
- when $\lambda \to \infty$, it is "too expensive" to allow for any β to be different than 0...
- So, for different values of λ we get a different solution to the problem

Ridge regression

- What ridge regression is doing is exploring the bias-variance trade-off! The larger the λ the more bias (towards zero) is being introduced in the solution, ie, the less flexible the model becomes... at the same time, the solution has less variance
- As always, the trick to find the "right" value of λ that makes the model not too simple but not too complex!
- Whenever possible, we will choose λ by comparing the out-of-sample performance (usually via cross-validation)

Ridge regression



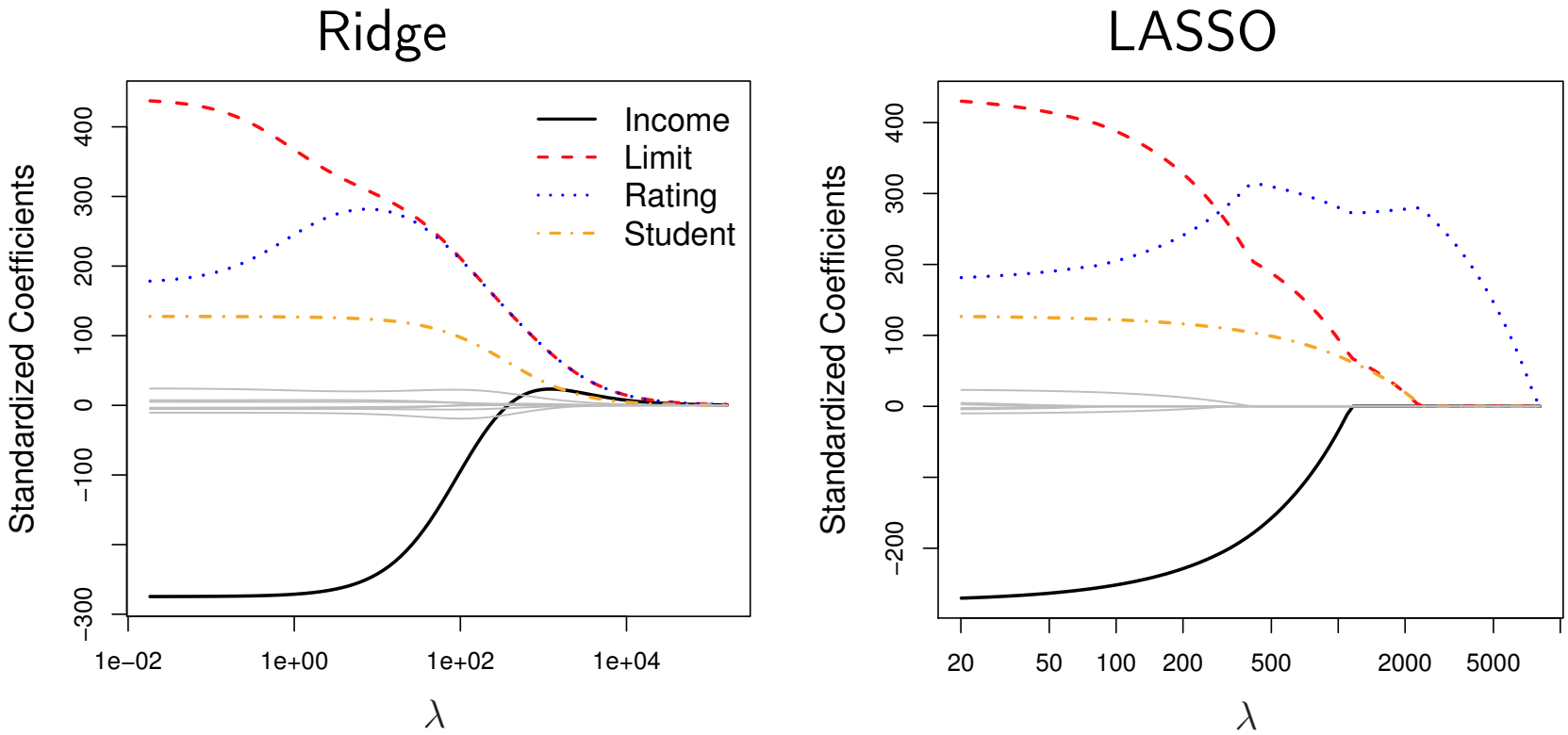
bias^2 (black), *variance* (green), test MSE (purple)

→ Ridge is computationally very attractive as the "computing cost" is almost the same of least squares (contrast that with subset selection!)

→ It's a good practice to always center and scale the X's

LASSO

The LASSO is a shrinkage method that performs automatic selection. It is similar to ridge but it will provide solutions that are sparse, ie, some β's exactly equal to 0! This facilitates interpretation of the results...



LASSO

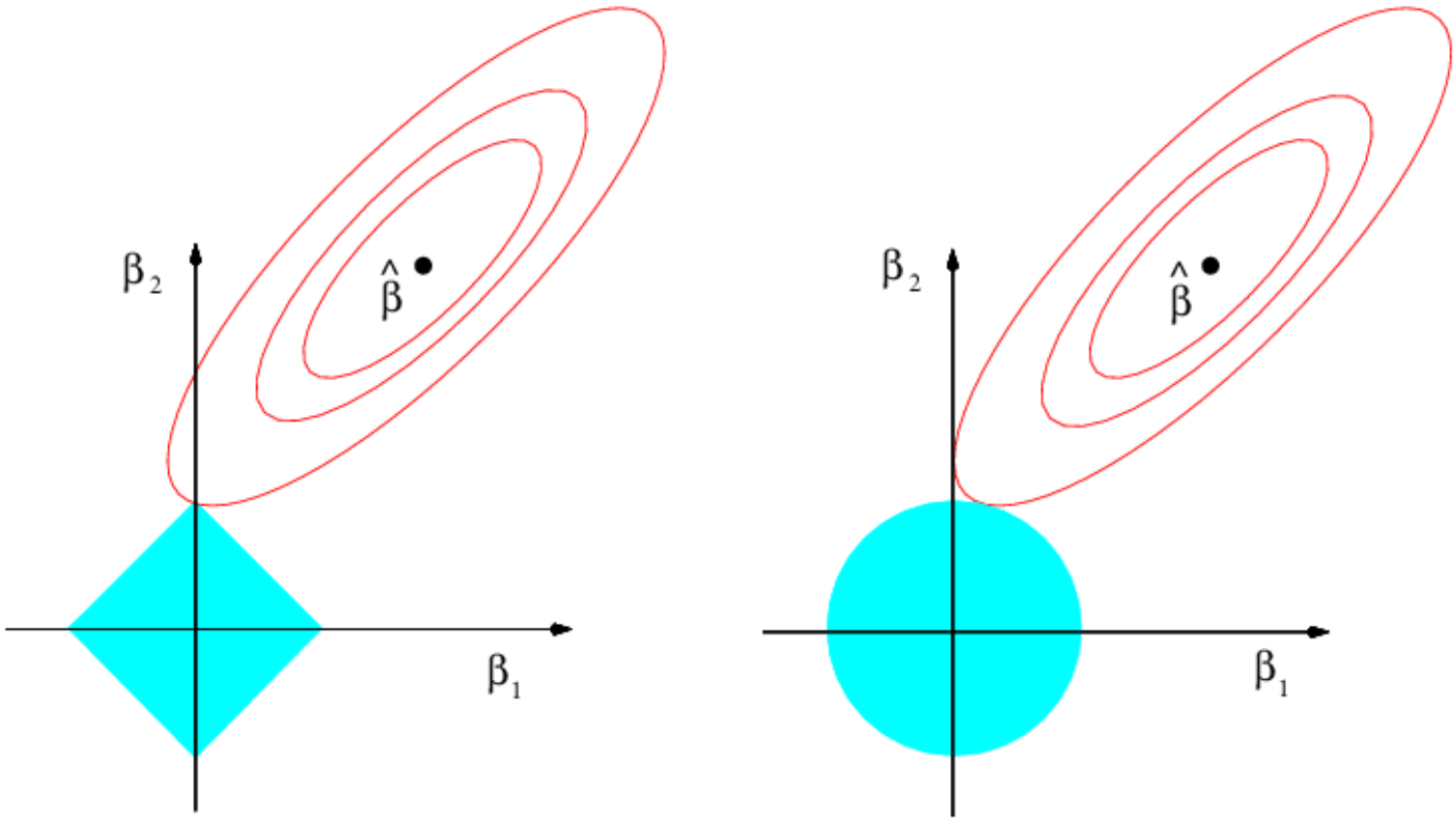
The LASSO solves the following problem:

$$\text{arg min}_{\beta} \left\{ \sum_{i=1}^{n} \left(Y_i - \beta_0 - \sum_{j=1}^{p} \beta_j X_{ij} \right)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\}$$

- Once again, λ controls how flexible the model gets to be
- Still a very efficient computational strategy
- Whenever possible, we will choose λ by comparing the out-of-sample performance (usually via cross-validation)

Ridge vs. LASSO

Why does the LASSO outputs zeros?



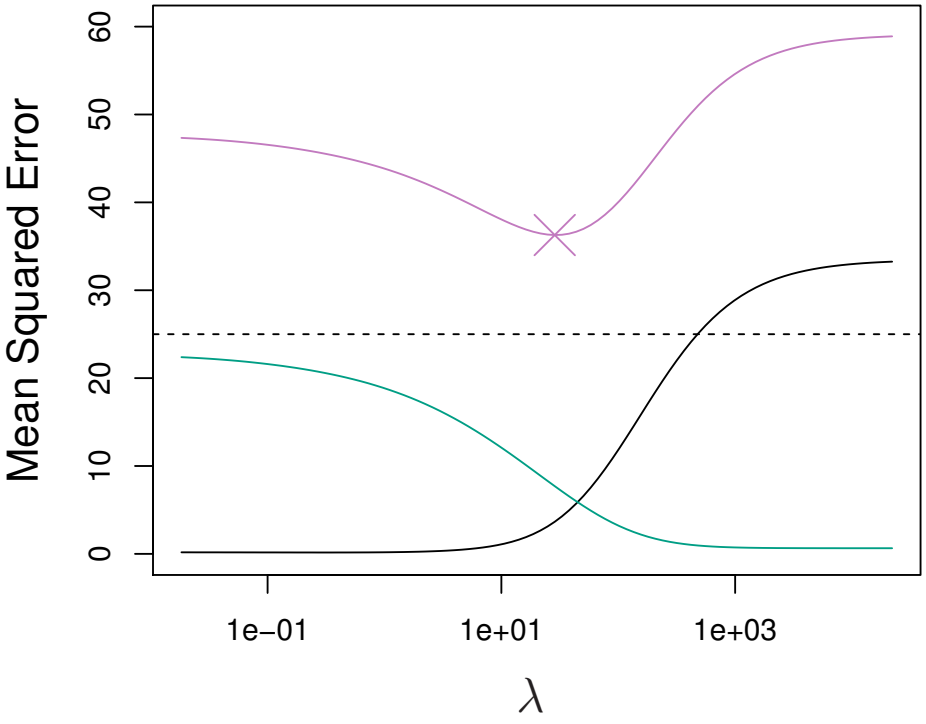
Ridge vs. LASSO

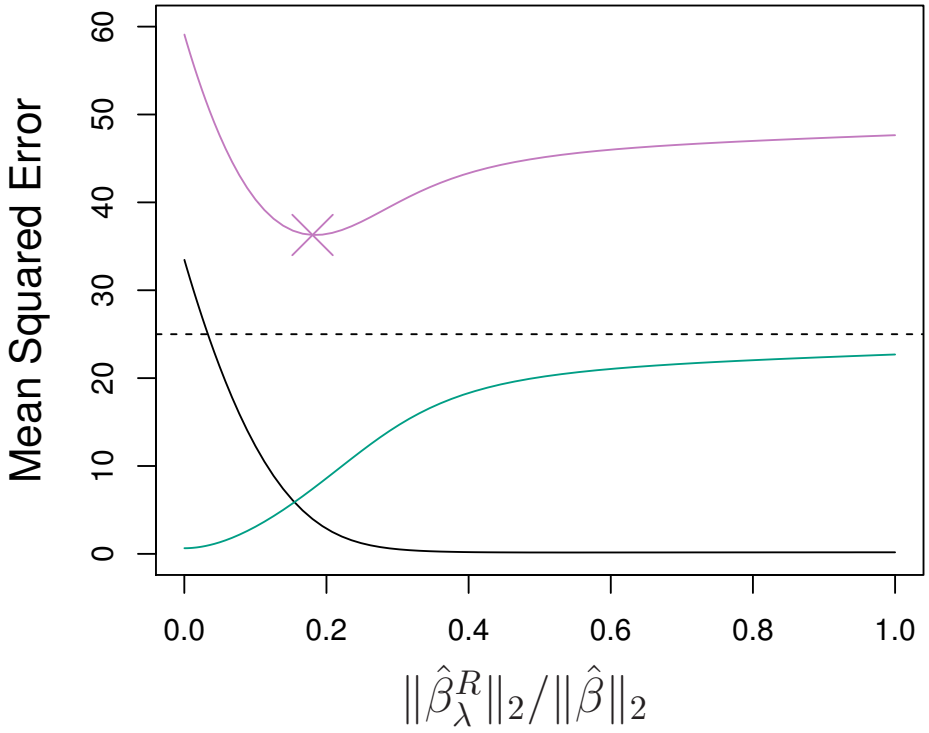
Which one is better?

- **It depends**...
- In general LASSO will perform better than Ridge when a relative small number of predictors have a strong effect in Y while Ridge will do better when Y is a function of many of the X's and the coefficients are of moderate size
- LASSO can be easier to interpret (the zeros help!)
- But, if prediction is what we care about the only way to decide which method is better is comparing their out-of-sample performance

Choosing λ

The idea is to solve the ridge or LASSO objective function over a grid of possible values for λ...





How to do this in R

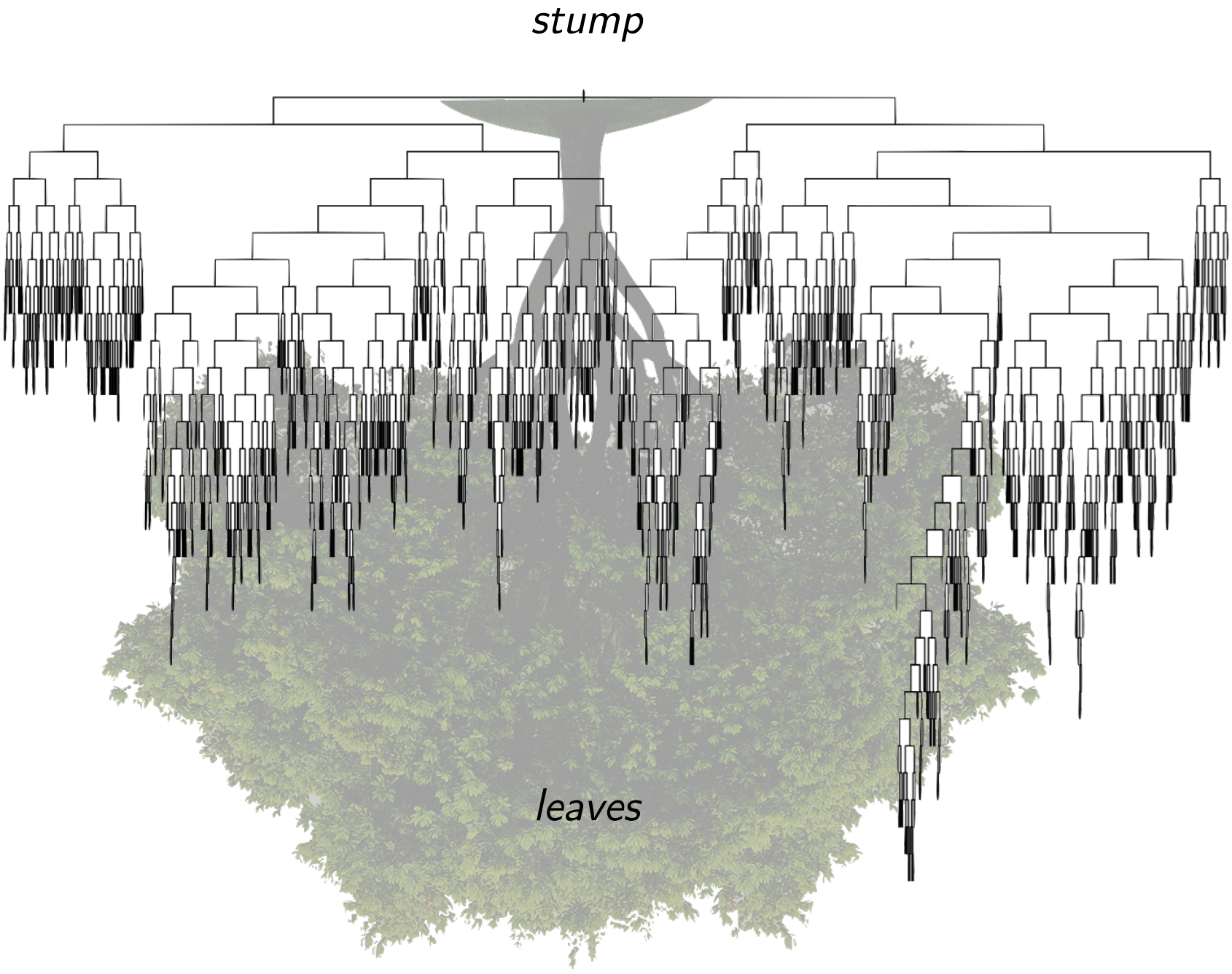
Check out the package `glmnet`. You can easily do Ridge, LASSO, and much more!

Nonlinear models (Trees) and the B-V tradeoff

Trees

Trees

Trees



Trees

Tree-based methods are a major player in data-mining.

Good:

- flexible fitters, capture nonlinearity and interactions.
- do not have to think about scale of variables.
- handles categorical and numeric y and x very nicely.
- fast.
- interpretable (when small).

Bad:

Not the best in out-of-sample predictive performance
(*but not bad!!*).

However,

If we **bag** or **boost** trees, we can get the best off-the-shelf prediction available.

Bagging and Boosting are **ensemble methods** that combine the fit from many (hundreds, thousands) of tree models to get an overall predictor.

The overall goal ...

$$Y_i = f(X_i) + \epsilon_i$$

with $f =$ 

Back to the Boston housing data

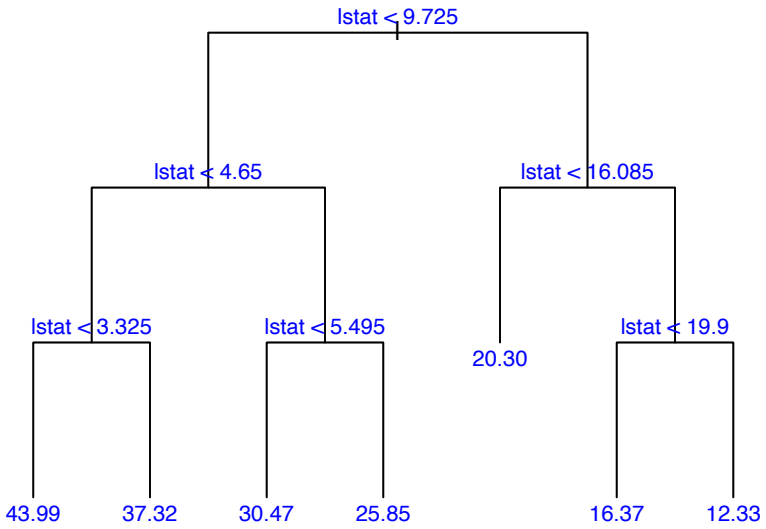
Let's look at a simple 1-dimensional example so that we can see what is going on.

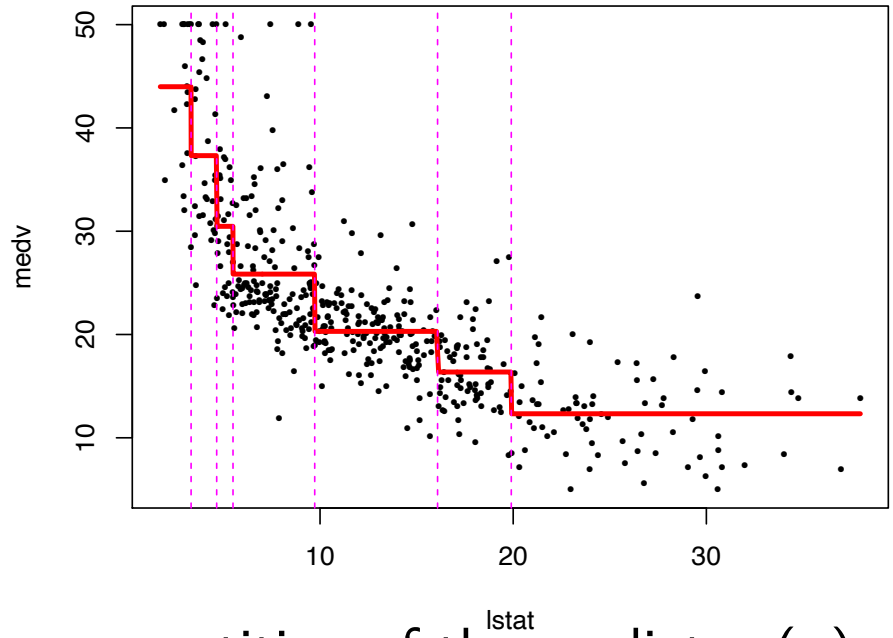
We'll use the Boston housing data and relate x=lstat to y=medval.

At left is the **tree** fit to the data.

At each **interior node** there is a decision rule of the form $\{x < c\}$. If $x < c$ you go left, otherwise you go right.

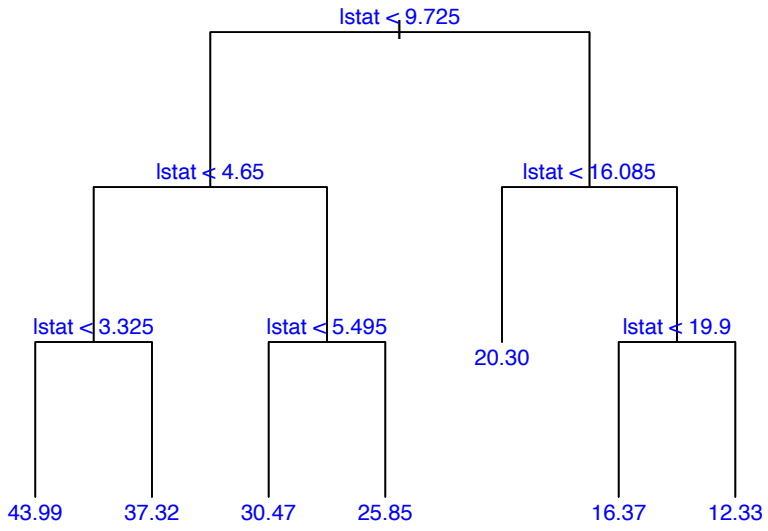
Each observation is sent down the tree until it hits a bottom node or **leaf** of the tree.

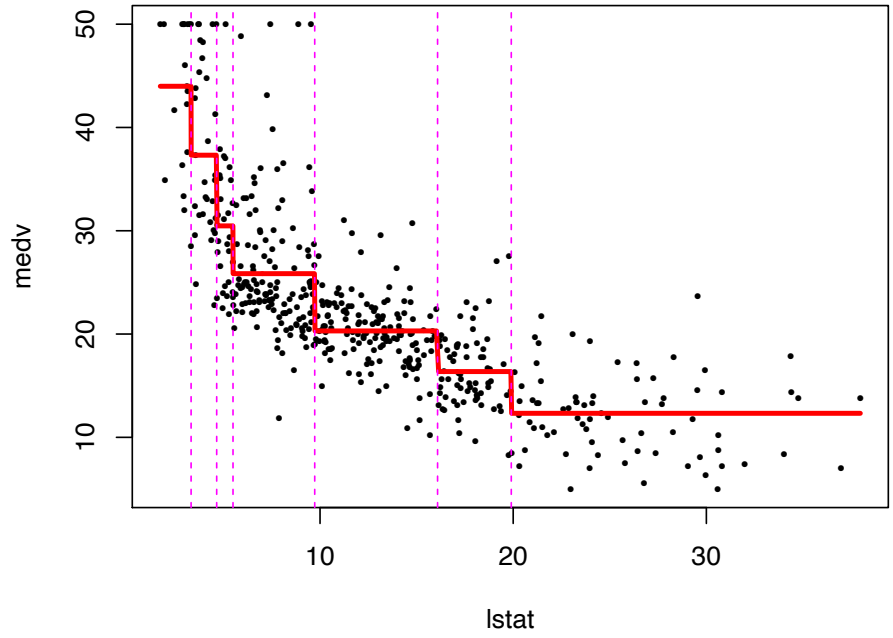




The set of bottom nodes gives us a partition of the predictor (x) space into disjoint regions. At right, the vertical lines display the partition. With just one x, this is just a set of intervals.

Within each region (interval) we compute the average of the y values for the subset of training data in the region. This gives us the step function which is our $\hat{f}$. The $\bar{y}$ values are also printed at left at the bottom nodes.



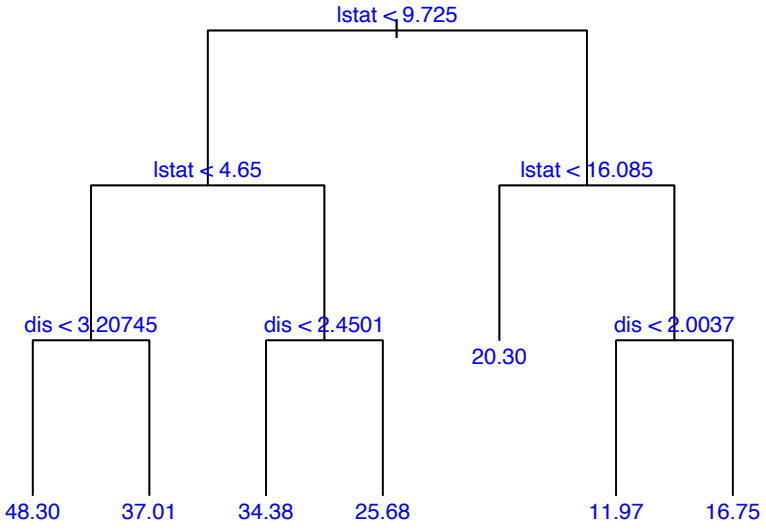


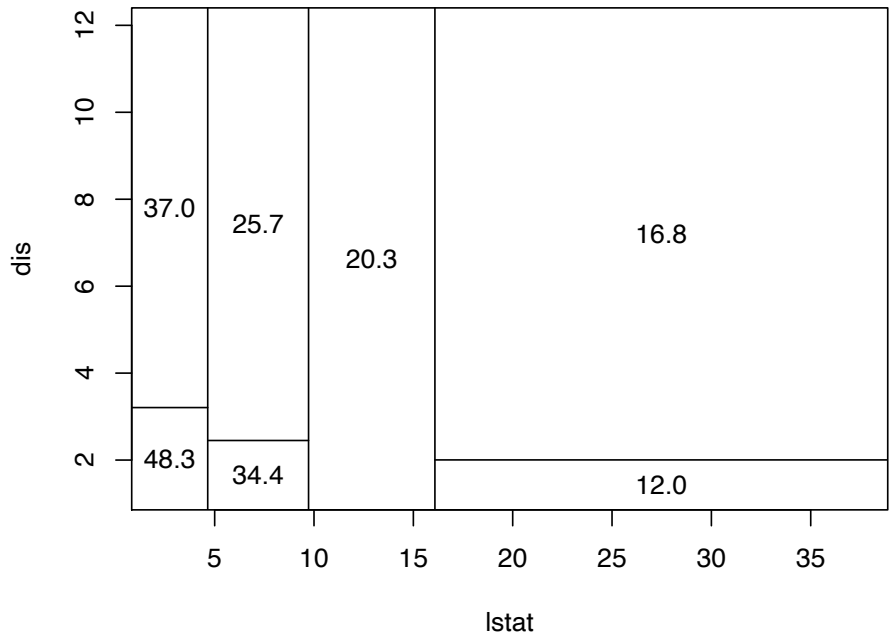
To predict, we just use our step function estimate of $f(x)$.

Equivalently, we drop x down the tree until it lands in a leaf and then predict the average of the y values for the training observations in the same leaf.

Two explanatory variables

Here is a tree with $x = (x_1, x_2) =$ (lstat,dis) and y=medv.

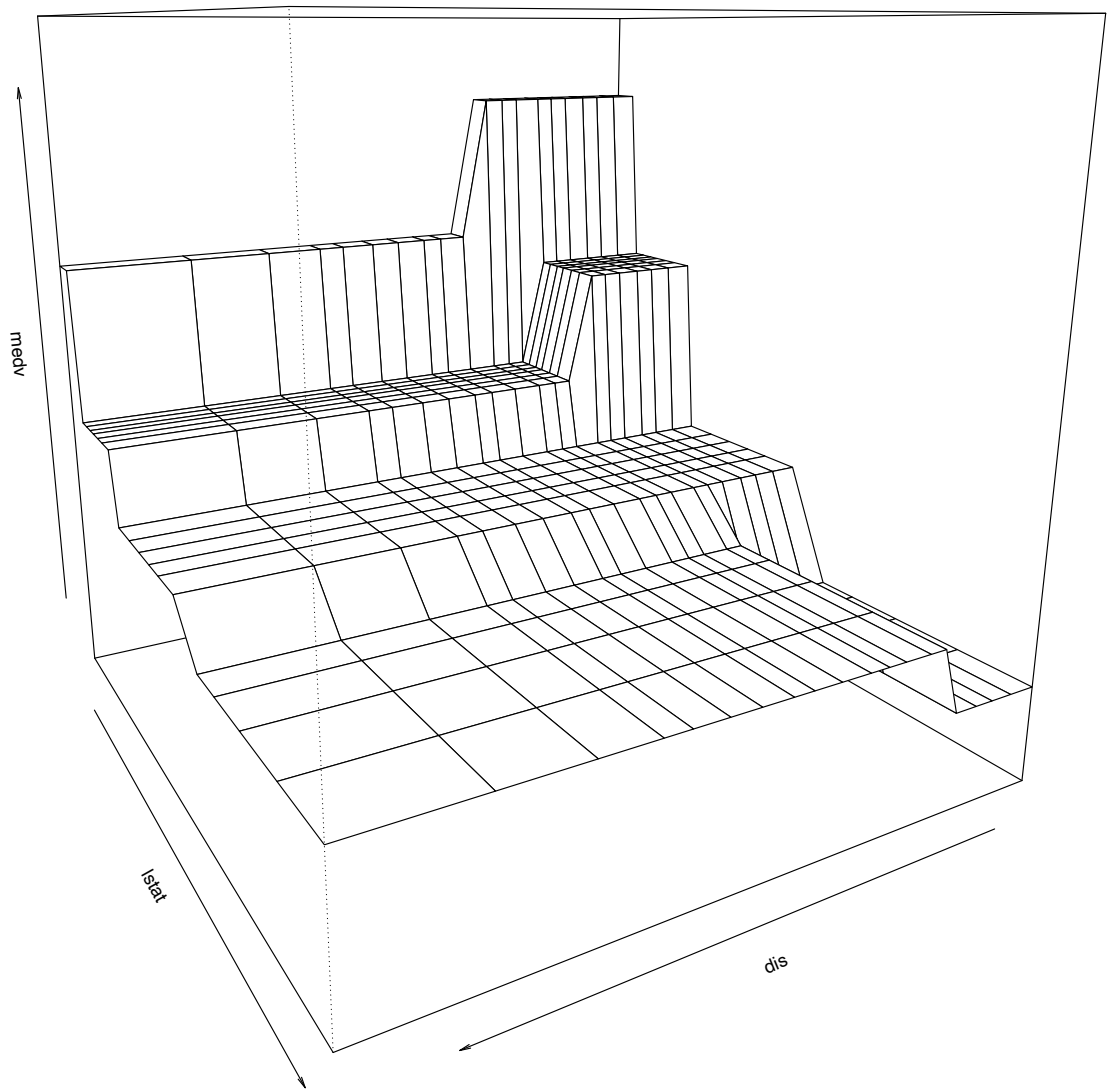




At right is the *partition* of the x space corresponding to the set of bottom nodes (leaves).

The average y for training observations assigned to a region is printed in each region and at the bottom nodes.

The regression function in 2-D!



This is the regression function given by the tree.

It is a step function which can seem stupid, but it delivers nonlinearity *and* interactions in a simple way and works with a lot of variables.

Notice the interaction.

The effect of `dis` depends on `lstat`!!

Tree models and the bias variance tradeoff

How do we fit trees to data??

The key idea is that a complex tree is a big tree.

We usually measure the complexity of the tree by the number of bottom nodes.

Minimizing a penalized loss

To fit a tree, we try to minimize:

$$C(T, y) = L(T, y) + \alpha |T|$$

where,

- $L(T, y)$ is our loss in fitting data y with tree T.
- $|T|$ is the number of bottom nodes in tree T.

For numeric y our loss is usually sum of squared errors, for categorical y we can use the deviance or the miss-classification rate.

Note: α is analogous to the lasso λ !!!

How do we do the minimization?

Now we have a problem.

While trees are simple in some sense, once we view them as variables in an optimization they are large and complex.

A key to tree modeling is the success of the following heuristic algorithm for fitting trees to training data.

(I. Grow Big)

Use a greedy, recursive forward search to build a big tree.

(i)

Start with the tree that is a single node.

(ii)

At each bottom node, search over all possible decision rules to find the one that gives the biggest decrease in loss (increase in fit).

(iii)

Grow a big tree, stopping (for example) when each bottom node has 5 observations in it.

(II. Prune Back)

(i)

Recursively, prune back the big tree from step (I).

(ii)

Give a current pruned tree, examine every pair of bottom nodes (having the same parent node) and consider eliminating the pair.

Prune the pair the gives the biggest decrease in our criterion C.

This is give us a sequence of subtrees of our initial big tree.

(iii)

For a given α, choose the subtree of the big tree that has the smallest C.

So,

Give training data and α we get a tree.

How do we choose α?

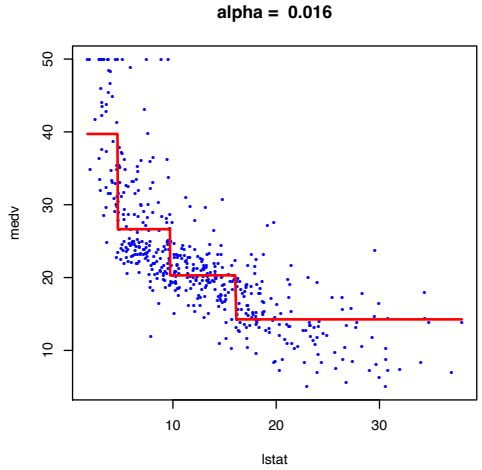
As usual, we can leave out a validation data set and choose the α the performs best on the validation data, or use k-fold cross validation.

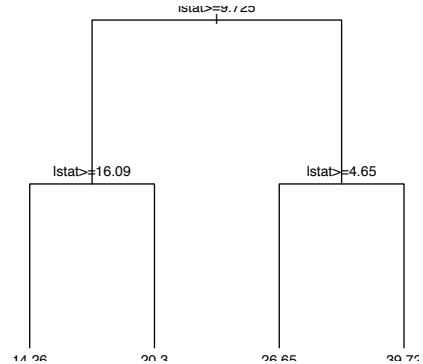
On the right are three different tree fits we get from three different α values (using all the data).

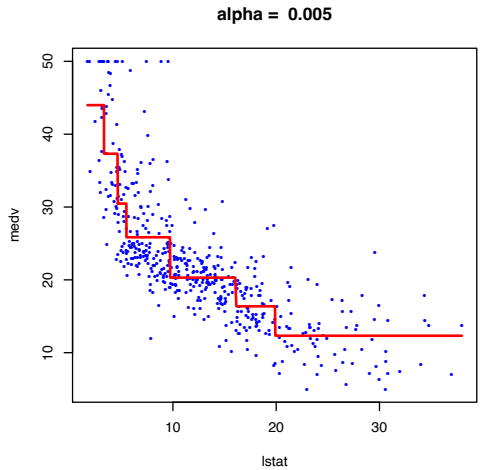
The smaller α is, the lower the penalty for complexity is, the bigger tree you get.

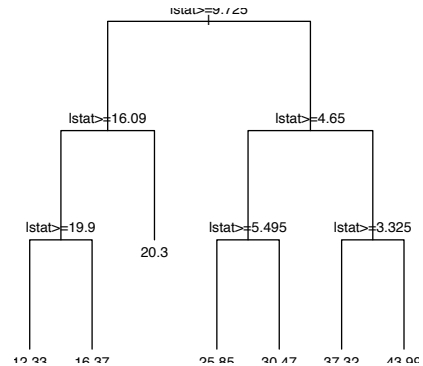
The top tree is a sub-tree of the middle tree, and the middle tree is a sub-tree of the bottom tree.

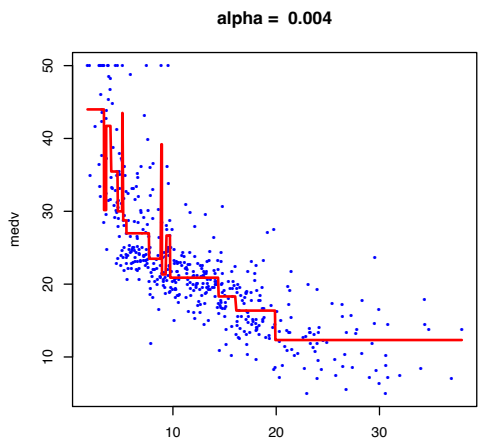
The middle α is the one suggested by CV.

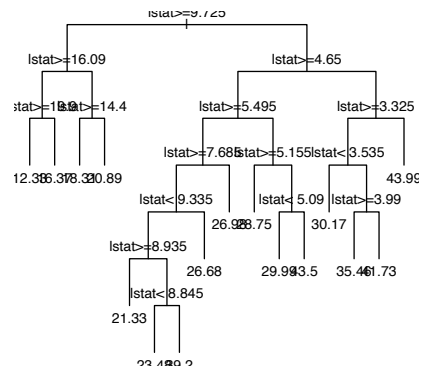












What are the models data scientists use today?

- Random forests
- Boosting and Bagging trees
- **Bayesian additive regression trees (BART)**
  - $\rightarrow$ tree growth is probabilistic
  - $\rightarrow$ regularization is "baked into" the priors