Lecture 1

Introduction to Predictive Models, The Bias Variance Tradeoff, Cross Validation

Mladen Kolar (mkolar@chicagobooth.edu)

Introduction to Predictive Models

Supervised learning

Training experience: a set of labeled examples (samples, observations) of the form $(x_1, y_1), (x_2, y_2), \ldots, (x_i, y_i), \ldots, (x_n, y_n)$ where $x_i = (1, x_{i1}, \ldots, x_{ip})$ are vectors of input variables (covariates, predictors) and y is the output

This implies the existence of a "teacher" who knows the right answers

What to learn: A function $f : \mathcal{X}_1 \times \mathcal{X}_2 \times \cdots \times \mathcal{X}_p \mapsto \mathcal{Y}$ which maps the input variables into the output domain

Goal: minimize the error (loss) function

- Ideally, we would like to minimize error on all possible instances
- But we only have access to a limited set of data ...

Supervised learning: Regression

Simply put, the goal is to predict a target variable Y with input variables X!

A useful way to think about it is that Y and X are related as

$$y_i = f(x_i) + \epsilon_i$$

and the goal is to learn f from n examples

- $f(x)$: the part of Y you learn from X, the signal.
- ϵ: the part of Y you do not learn from X, the noise.

Regression in business

Optimal portfolio choice:

- Predict the future joint distribution of asset returns
- Construct an optimal portfolio (choose weights)

Determining price and marketing strategy:

- Estimate the effect of price and advertisement on sales
- Decide what is optimal price and ad campaign

Credit scoring model:

- Predict the future probability of default using known characteristics of borrower
- Decide whether or not to lend (and if so, how much)

Regression in everything

Straight prediction questions:

- What price should I charge for my car?
- What will the interest rates be next month?
- Will this person like that movie?

Explanation and understanding:

- Does your income increase if you get an MBA?
- Will tax incentives change purchasing behavior?
- Is my advertising campaign working?

Even More Examples

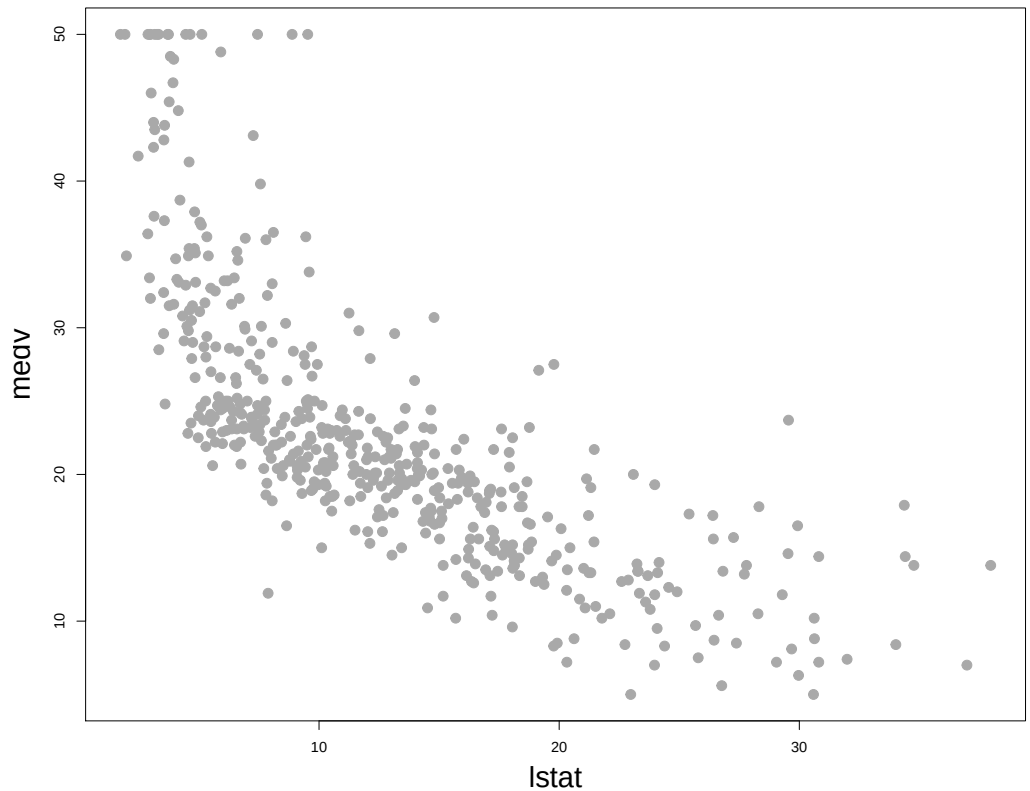
- Y: will a customer respond to a promotion (target marketing)
- Y: which customer is likely to cancel
- Y: the lifetime value of a customer (how much will they spend)
- Y: pregnancy (from shopping behaviour) so you can target
- Y: will a customer defect
- Y: predict which products a customer will like (Pandora, Amazon)
- Y: predict age of death (insurance companies)

See Tables 1-9 after page 142 of **Predictive Analytics** by Eric Siegel for many examples.

Example: Boston Housing

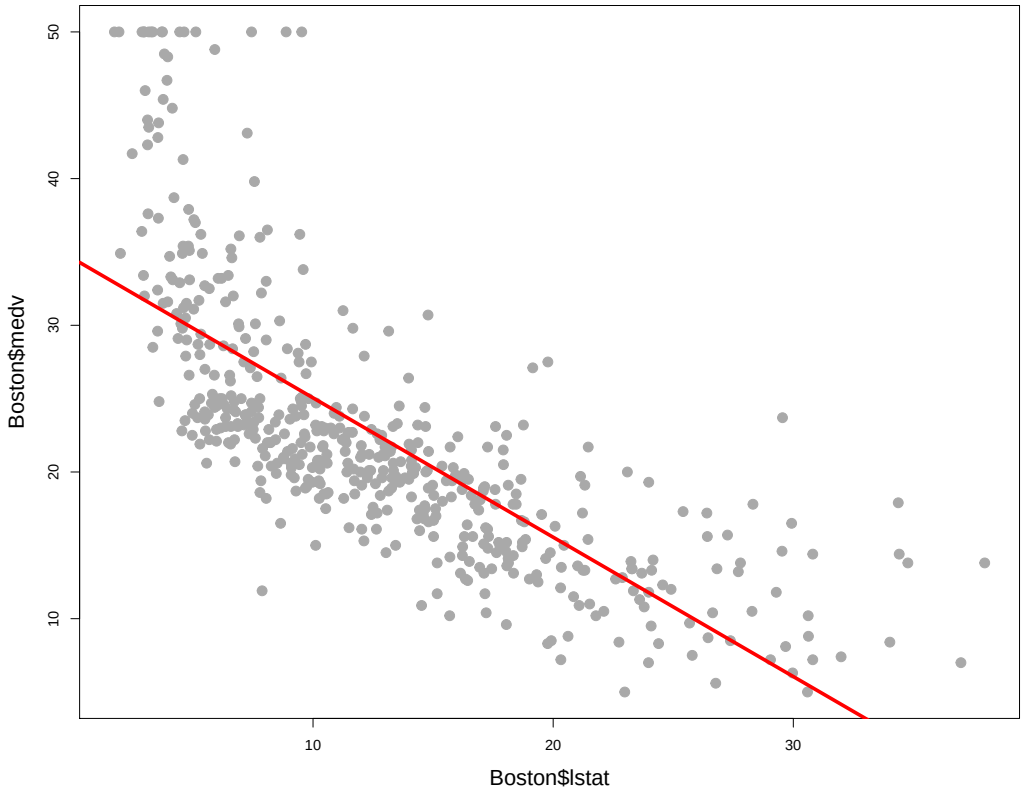
We might be interested in predicting the median house value as a function of some measure of social economic level. Each observation corresponds to a town in the Boston area.

- medv: median house value (data is old)
- lstat: % lower status.



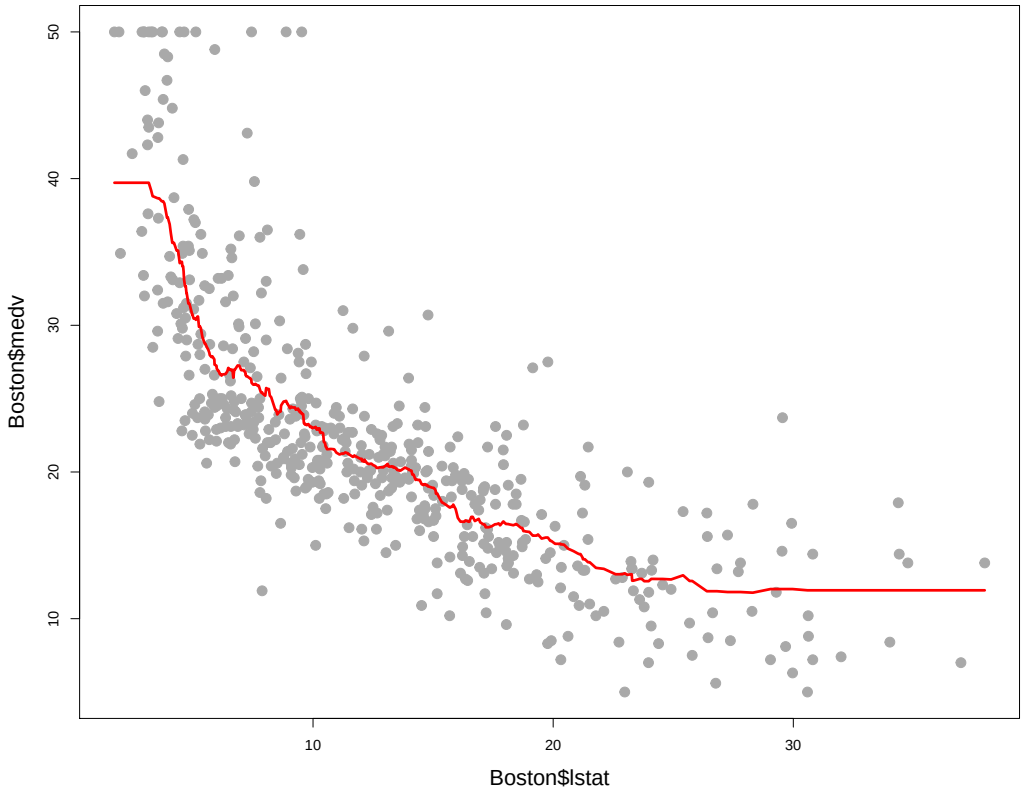
What should $f(\cdot)$ be?

How about this. . .



If *lstat* = 30 what is the prediction for *medv*?

or this?



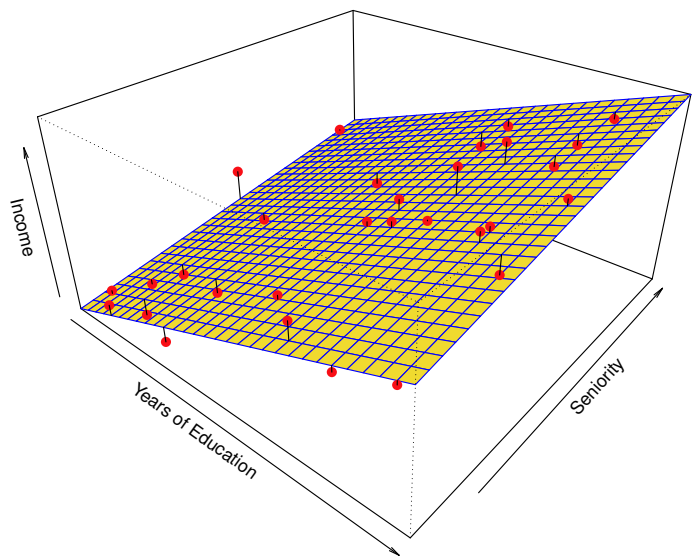
If *lstat* = 30 what is the prediction for *medv*?

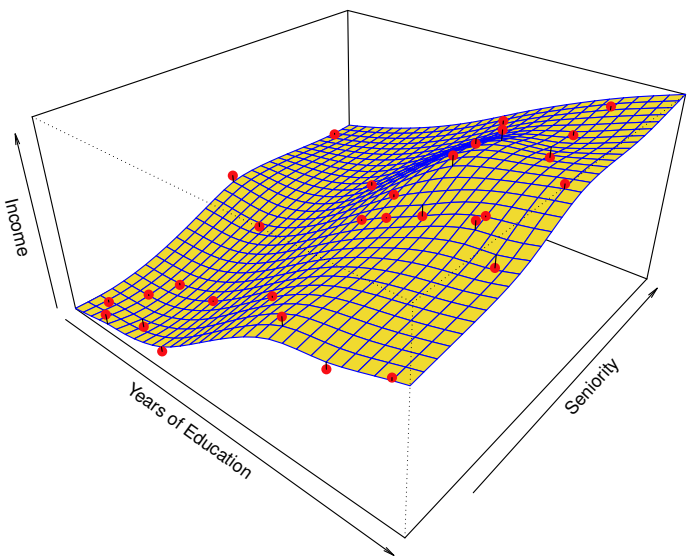
How do we estimate $f(\cdot)$?

- Using *training data*:

$$\{(X_1, Y_1), (X_2, Y_2), \ldots, (X_n, Y_n)\}$$

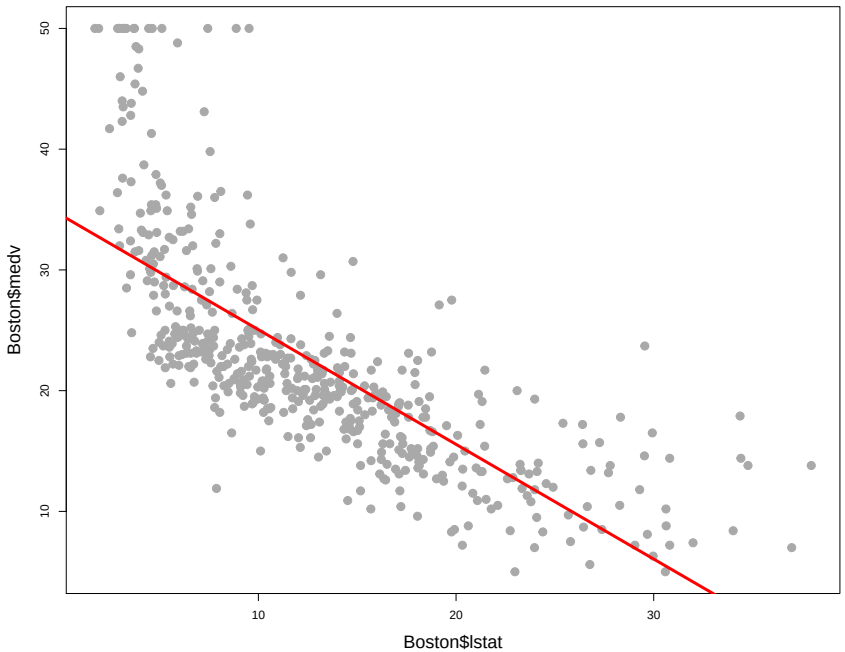
- We use a statistical method to *estimate* the function $f(\cdot)$
- Two general methodological strategies:
  1. simple parametric models (restricted assumptions about $f(\cdot)$)
  2. non-parametric models (flexibility in defining $f(\cdot)$)



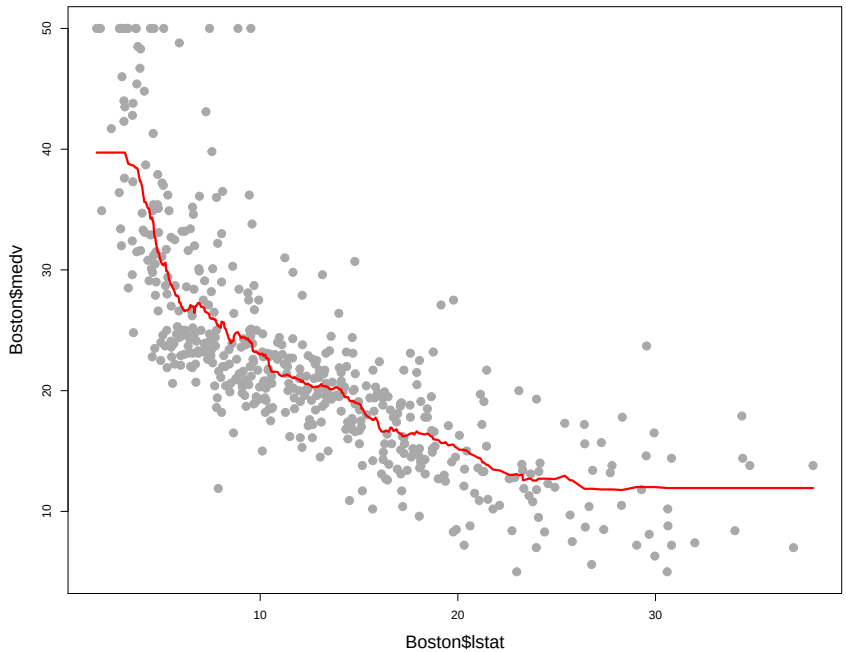


Back to Boston Housing

Parametric Model
($Y = \alpha + \beta x + \epsilon$)



Non-Parametric Model
(k-nearest neighbors)



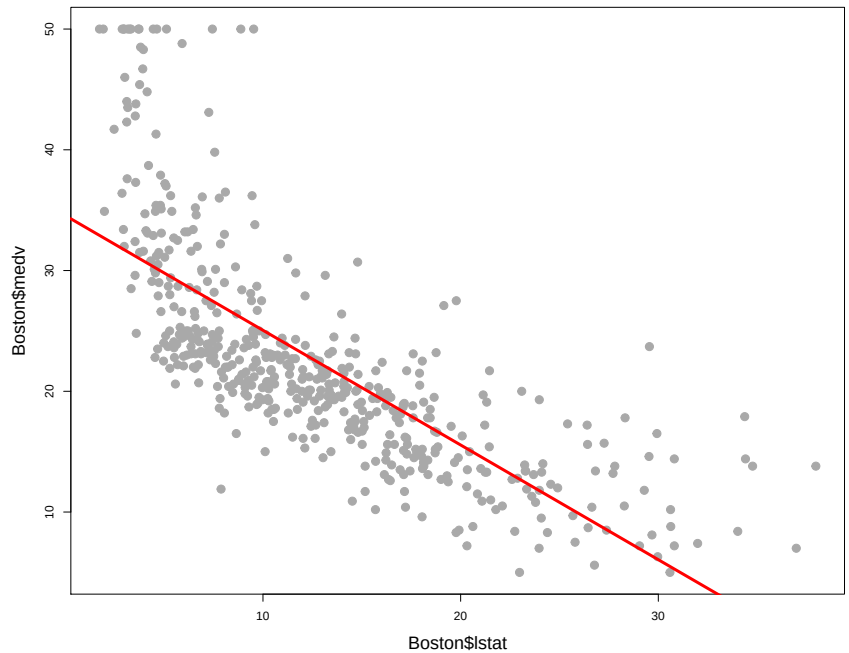
Simple parametric model:

$$Y_i = \alpha + \beta x_i + \epsilon_i$$

Using the training data,
we estimate $f(x)$ as

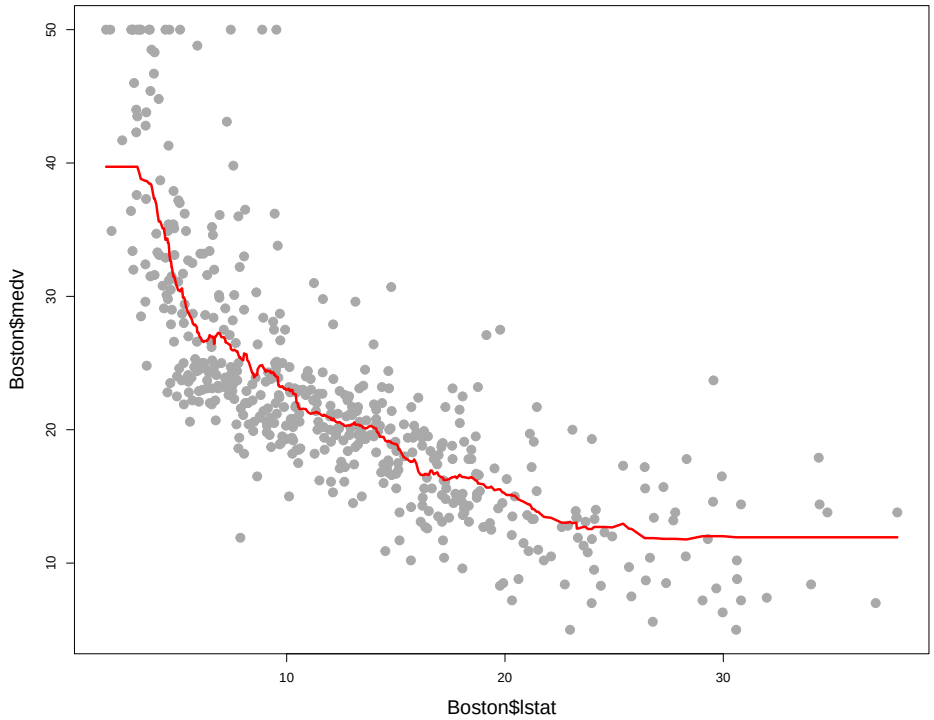
$$\hat{f}(x) = \hat{\alpha} + \hat{\beta} x$$

where $\hat{\alpha}$ and $\hat{\beta}$
are the linear
regression estimates.

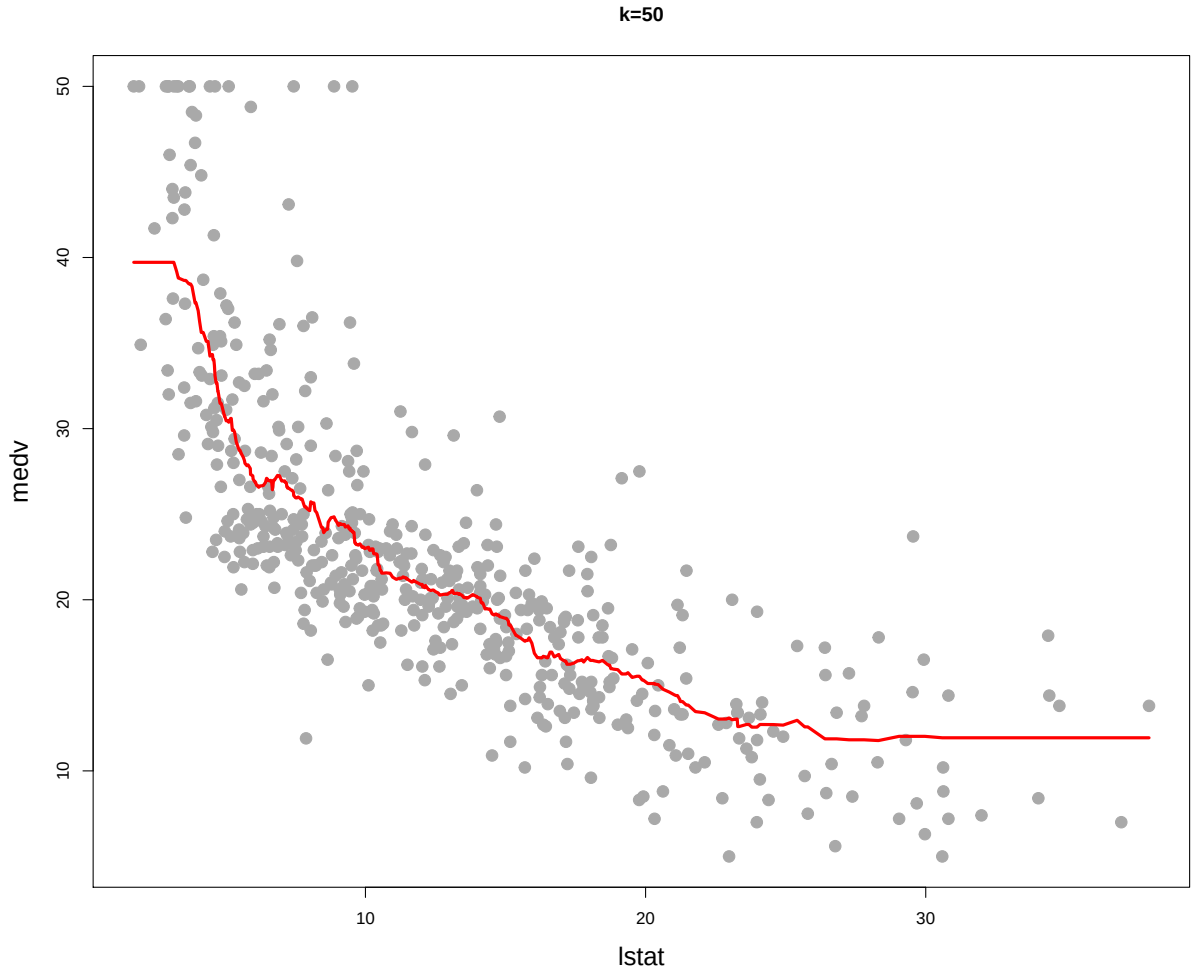


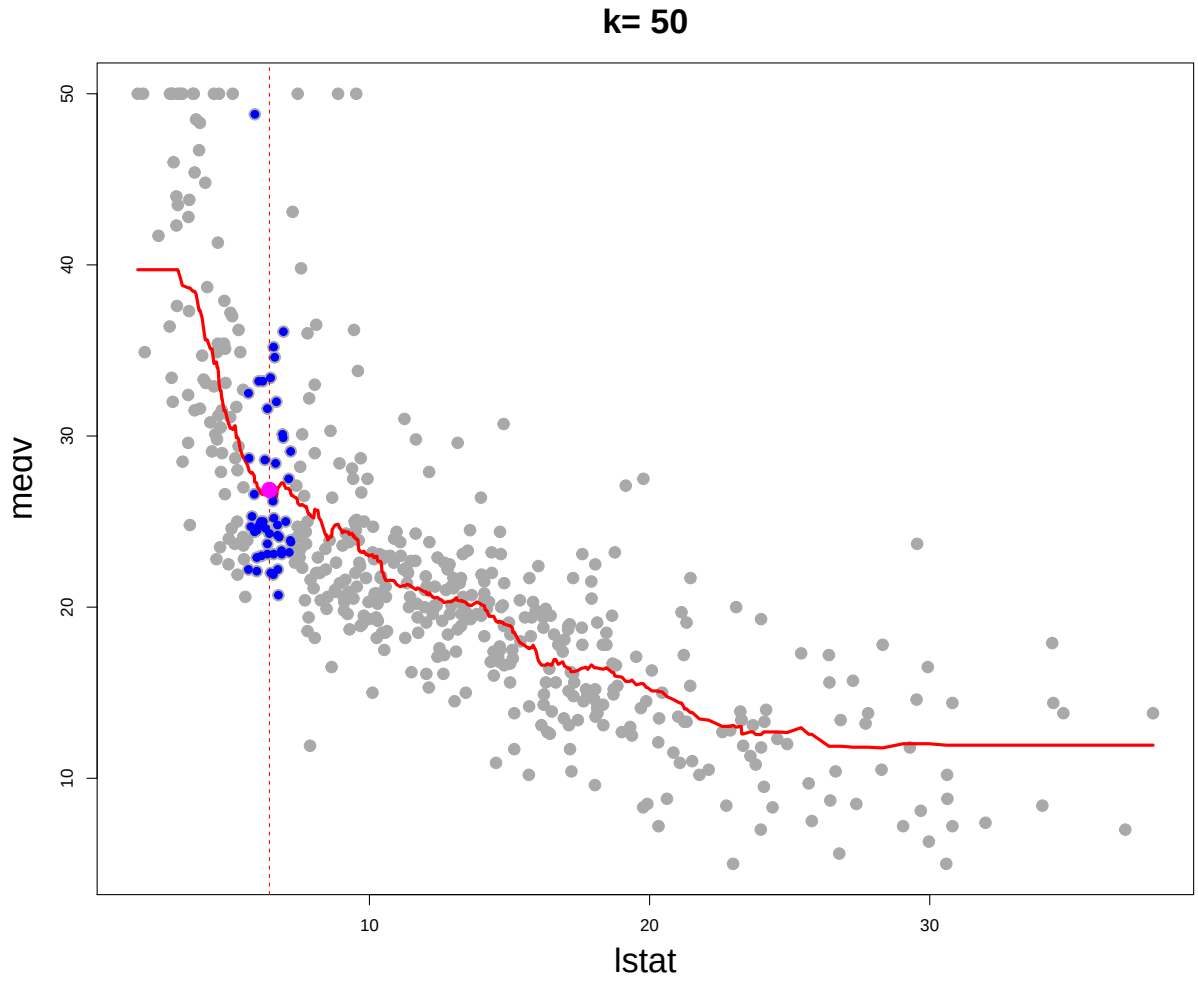
To get this estimate we used *kNN* - k-nearest neighbors.

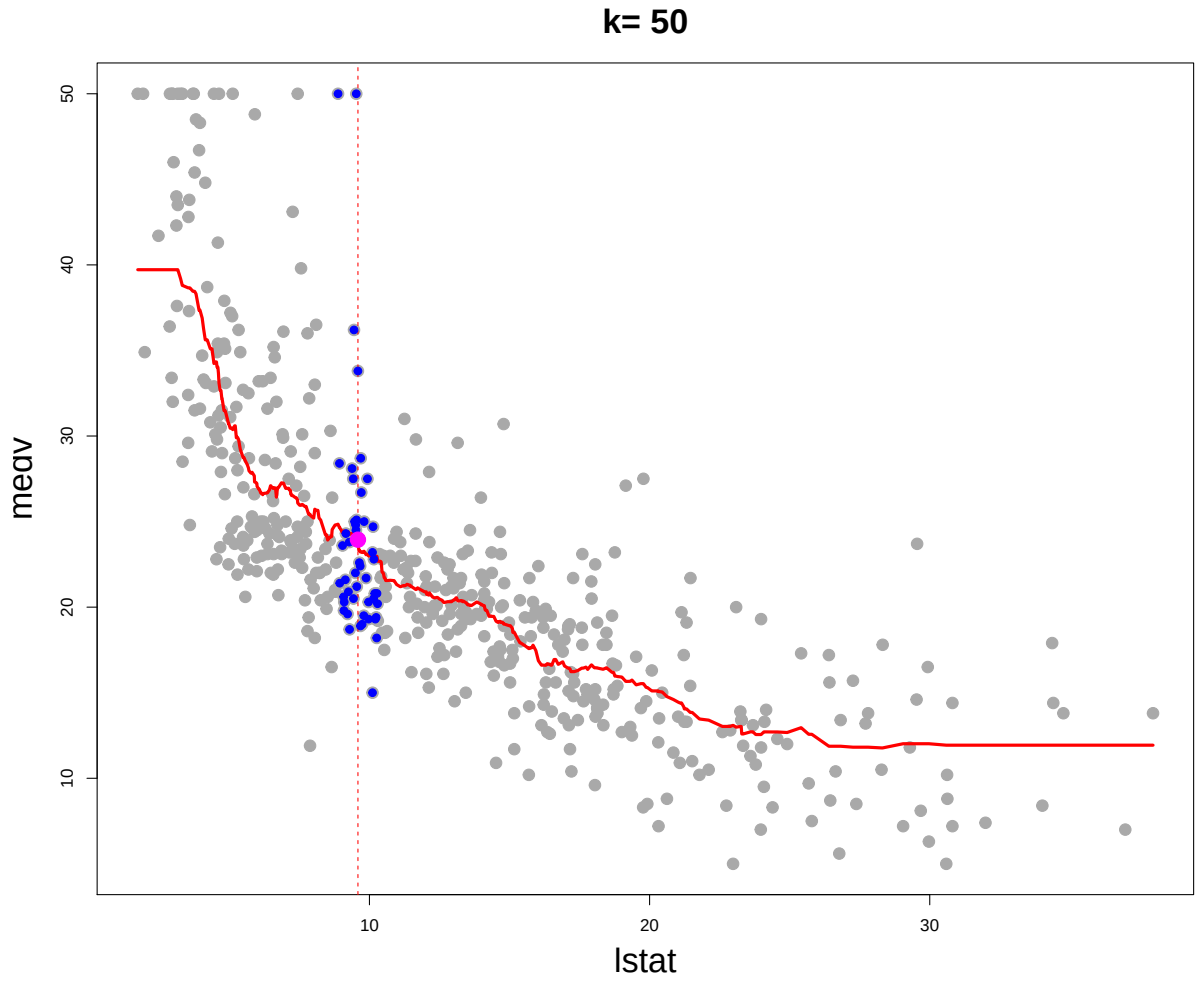
To estimate $f(x_f)$, average the y values for the k training observations with x *closest* to x_f.

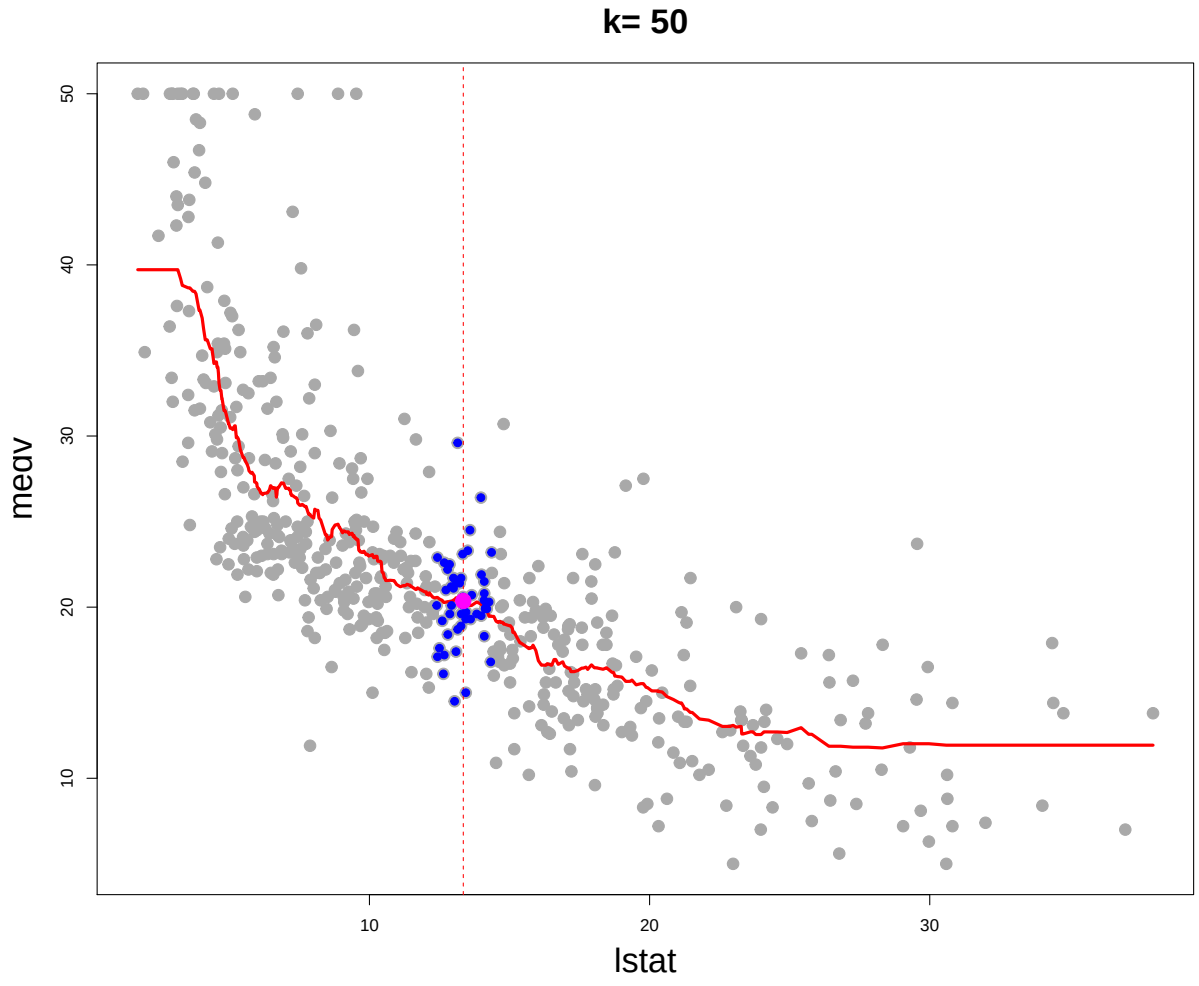


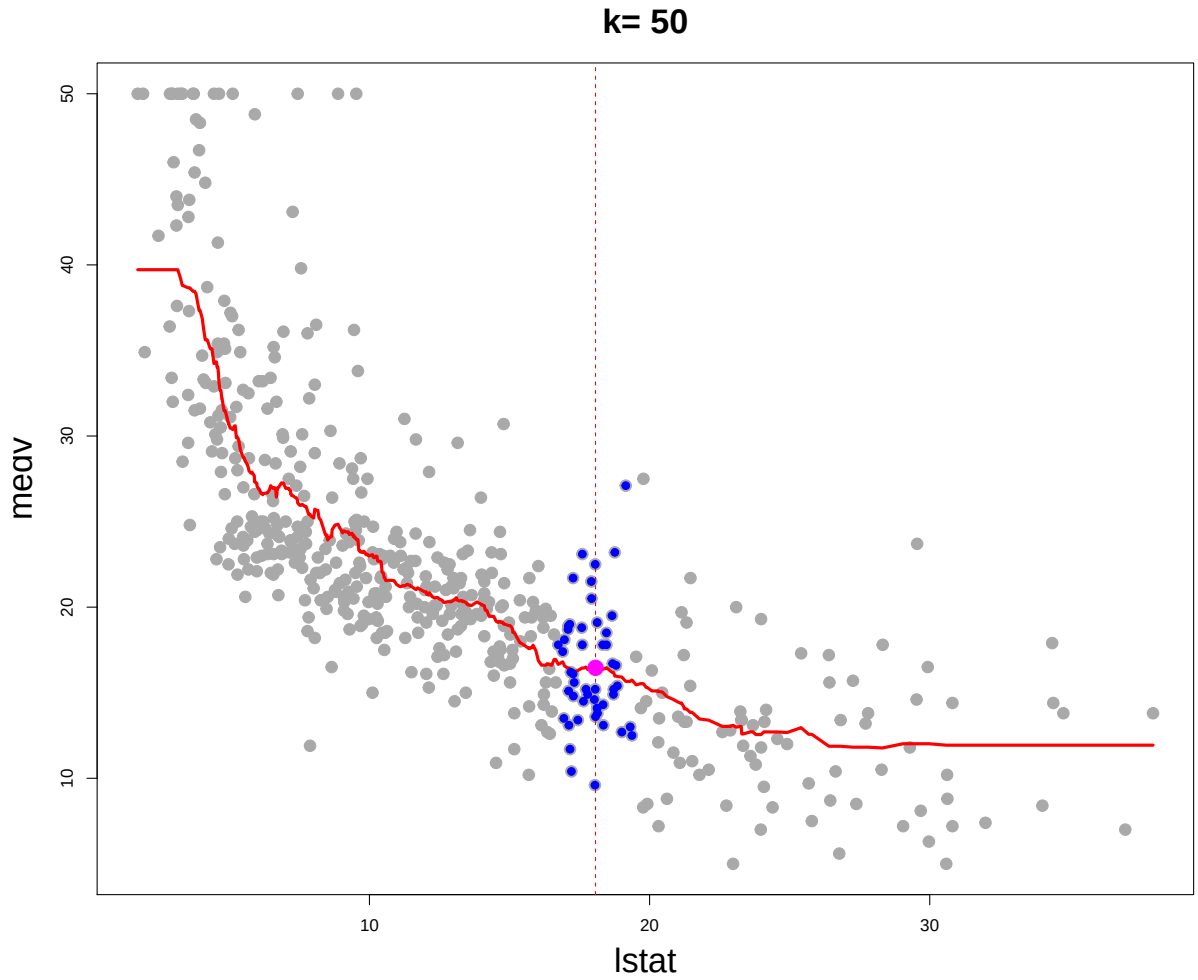
What do I mean by closest? We will choose the k=50 points that are closest to the X value at which we are trying to predict.


k=50
medv
lstat
10
20
30
40
50
10
20
30

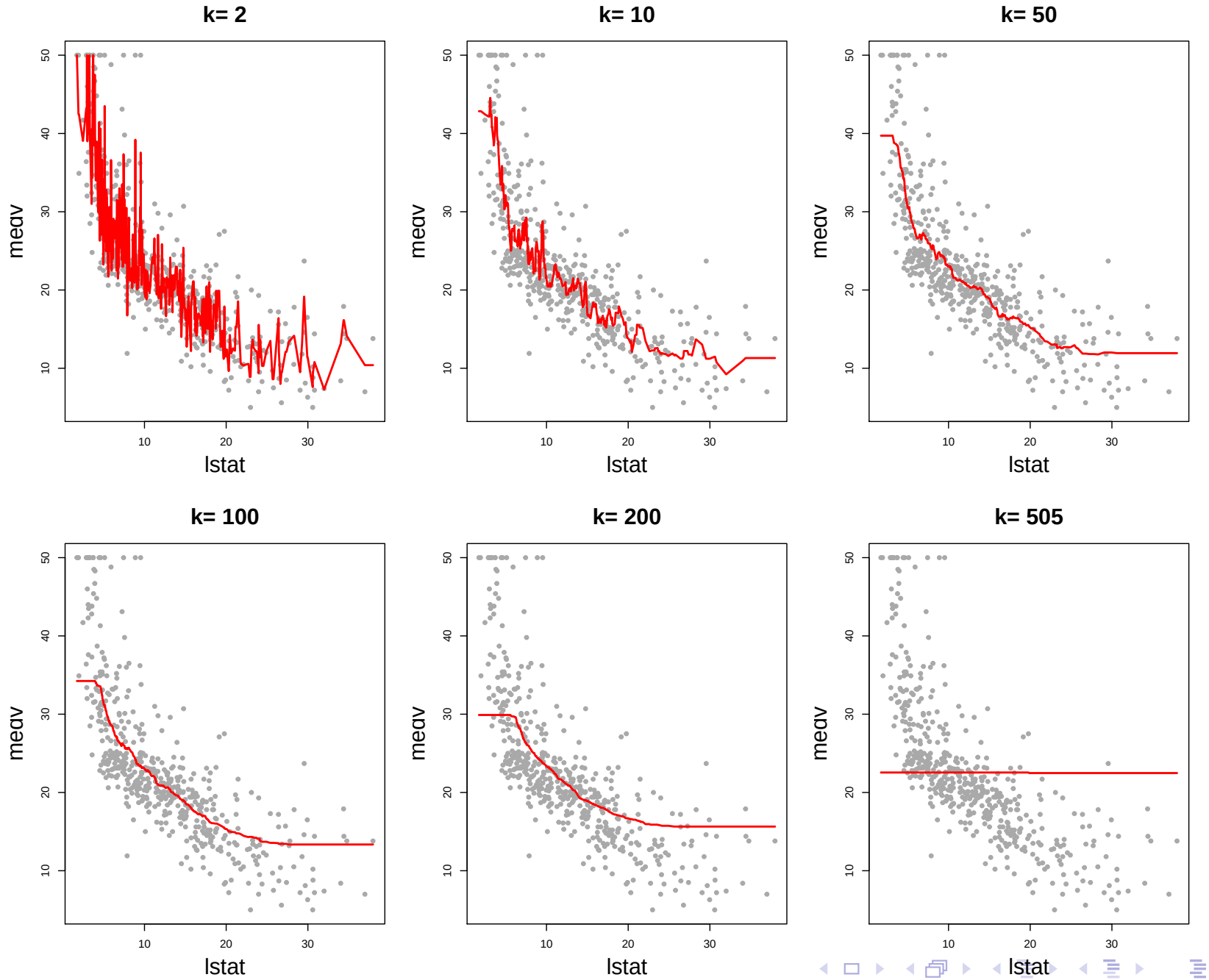

k= 50
medv
lstat
10
20
30
40
50
10
20
30


k= 50
medv
lstat
10
20
30
40
50
10
20
30


k= 50
medv
lstat
10
20
30
40
50
10
20
30


k= 50
medv
lstat
10
20
30
40
50
10
20
30

Seems sensible, what about other choices of k?



for k-NN:

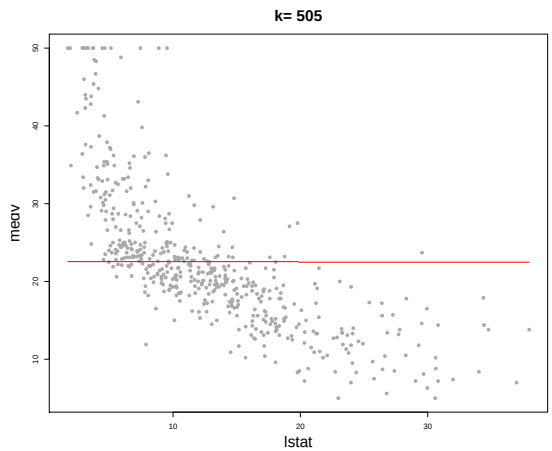
A big k gives us a simple looking function.

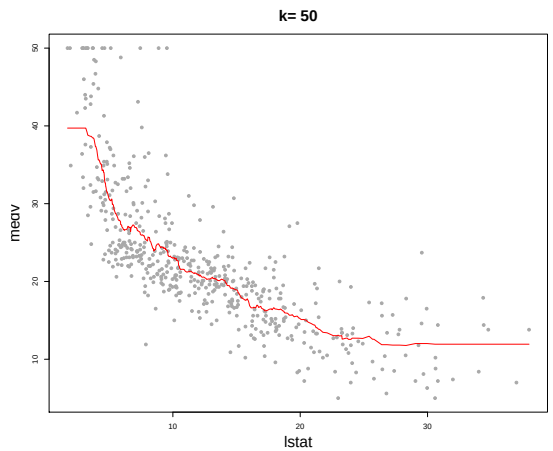
A small k can give us a more complex, flexible looking function.

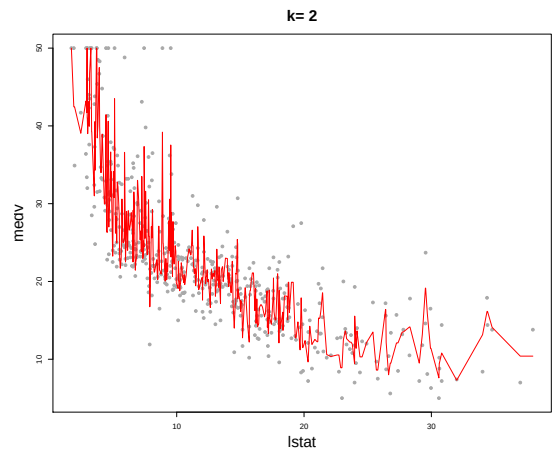
Complexity, Generalization and Interpretation

- As we have seen in the examples above, there are lots of options in estimating $f(X)$.
- Some methods are very flexible some are not... *why would we ever choose a less flexible model?*
  1. Simple, more restrictive methods are usually easier to interpret
  2. More importantly, it is often the case that simpler models are more accurate in making future predictions.

Not too simple, but not too complex!







Measuring Accuracy

Measuring Accuracy

How accurate are each of these models?

We can measure the fit of our model (our fuction estimate) using the mean squared error (MSE)

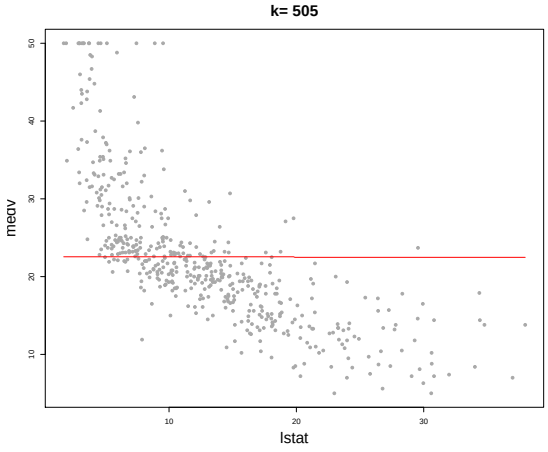
$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left[Y_i - \hat{f}(X_i)\right]^2$$

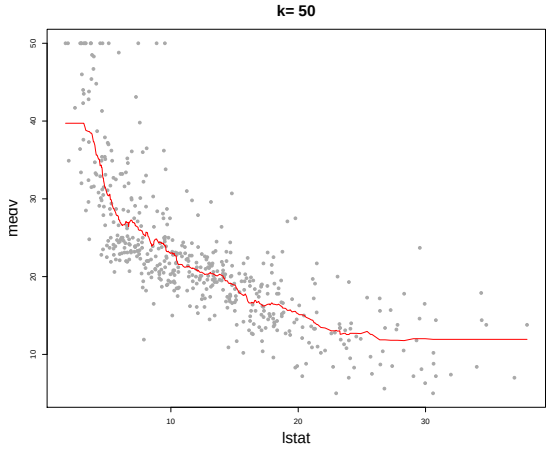
This measures, on average, how large the **mistakes** (errors) made by the model are. . .

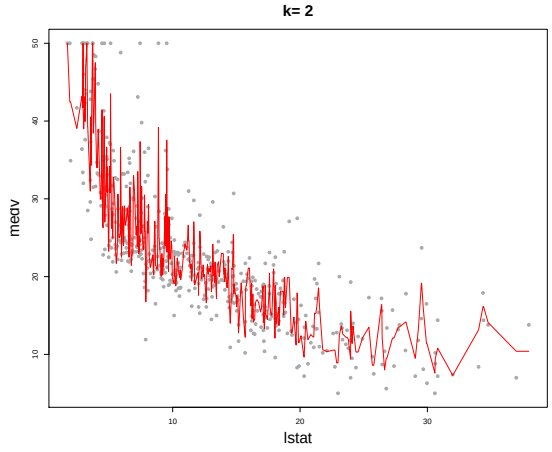
You will also see root mean squared error (RMSE) used

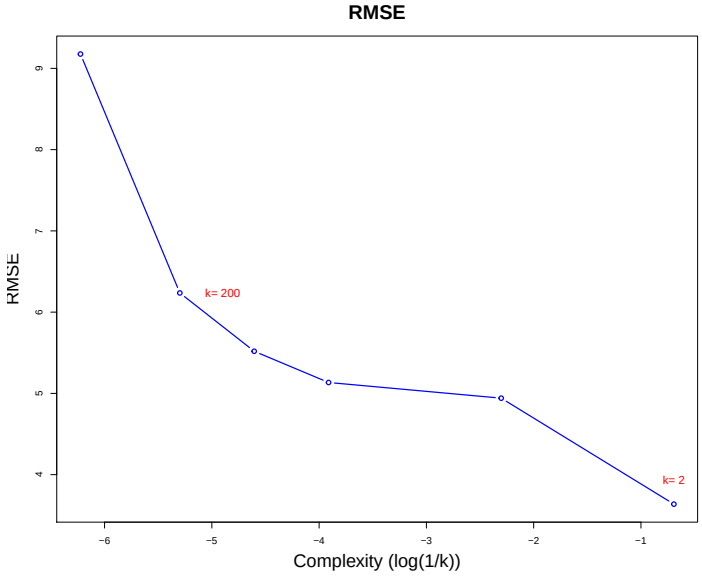
$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left[Y_i - \hat{f}(X_i)\right]^2}$$

Example: Boston housing (again)









So, I guess we should just go with the most complex model, i.e., $k = 2$, right?

Out-of-Sample Predictions

But, do we really care about explaining what we have already seen?

Key Idea: what really matters is our prediction accuracy out-of-sample!!!

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 test set (also known as **hold-out set**).

Let's look at the out-of-sample MSE:

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

(1)

Use the in-sample (training data) $(X_i, Y_i), i = 1, 2, \ldots, n\backslash$ to estimate f.

Now we have $\hat{f}$.

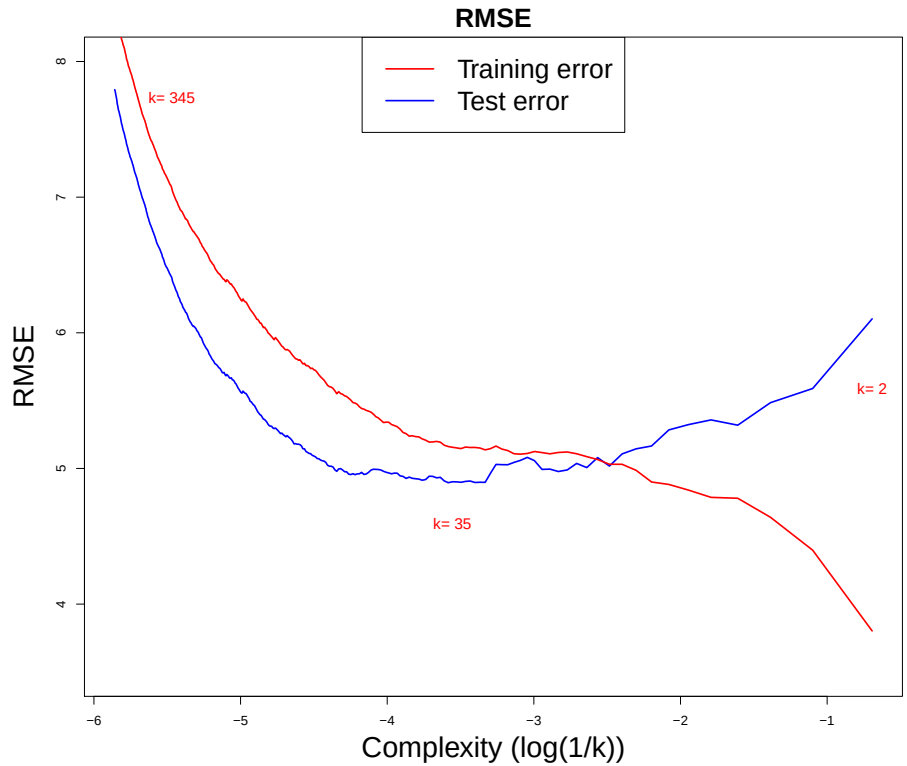
(2)

Evaluate predictive performance on the test data (X_i^o, Y_i^o), for $i = 1, \ldots, m$,

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

Out-of-Sample Predictions

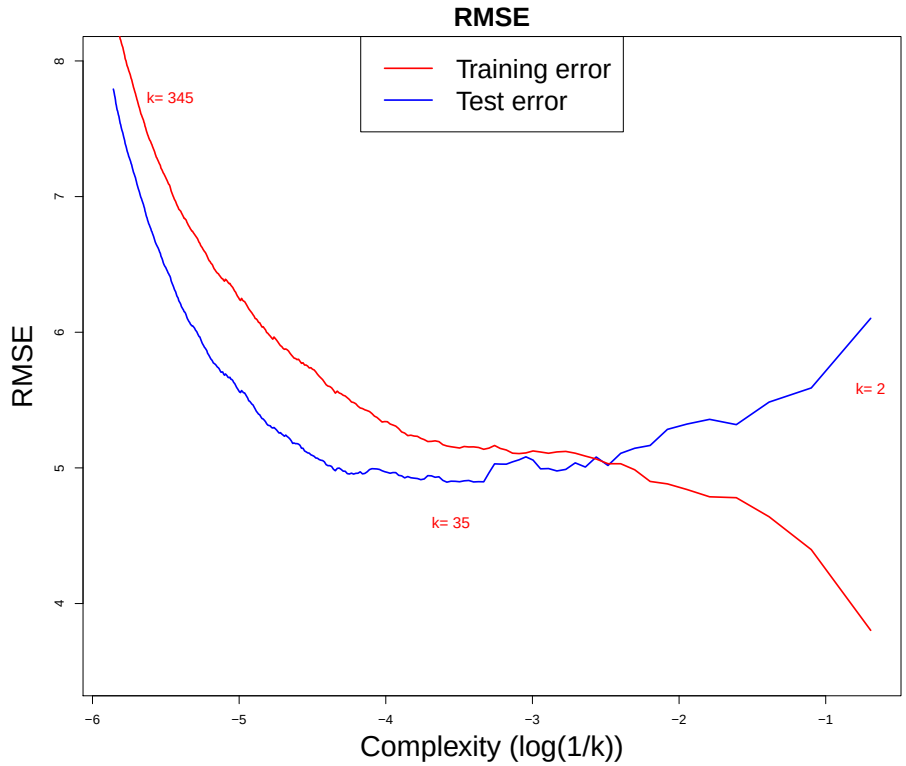
In our Boston housing example, I randomly chose a training set of size 400. I re-estimate the models using only this set and use the models to predict the remaining 106 observations (test set)...



Now, the model where $k = 35$ looks like the most accurate choice!!

Not too simple but not too complex!!!

Overfitting



The training error decreases as k gets smaller.

The testing error, measured on independent data, decreases at first, then starts increasing.

Overfitting

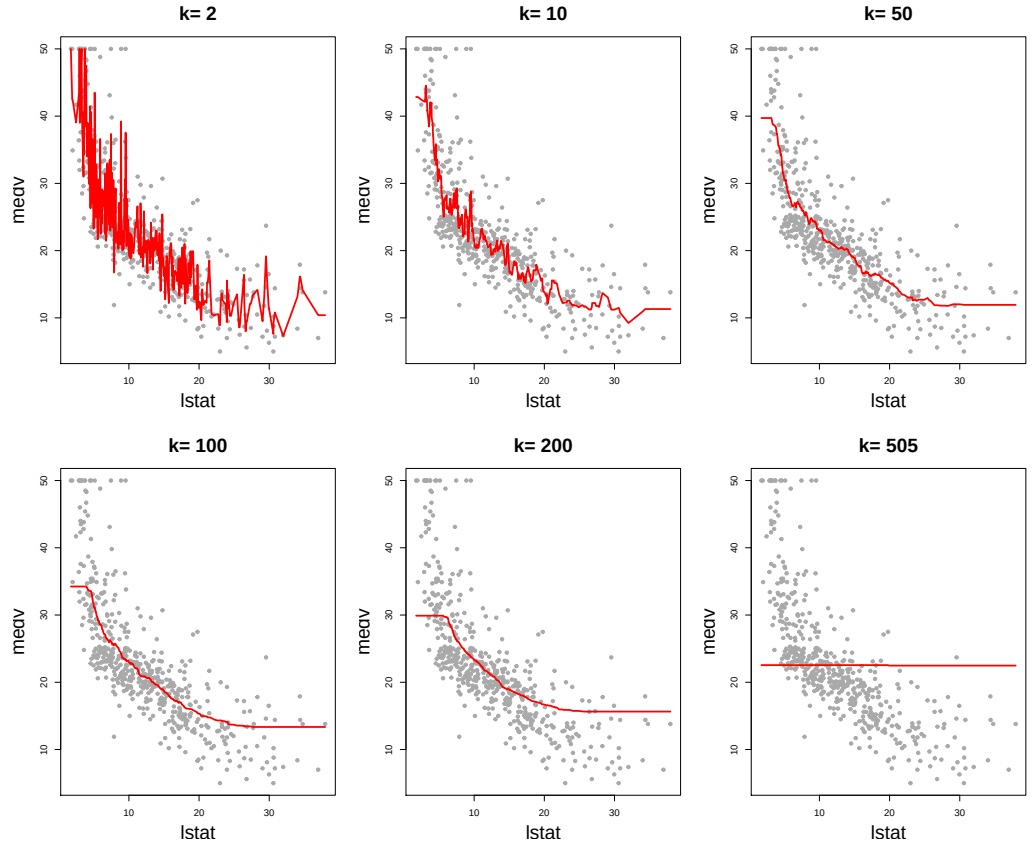
A general, hugely important problem for all machine learning algorithms.

We can find a function $\hat{f}$ that predicts perfectly the training data but does not generalize well to new data.

We are seeing an instance here: if we have a lot of parameters, the function $\hat{f}$ "memorizes" the data points, but is wild everywhere else.

What we really care is how well we can predict on new examples.

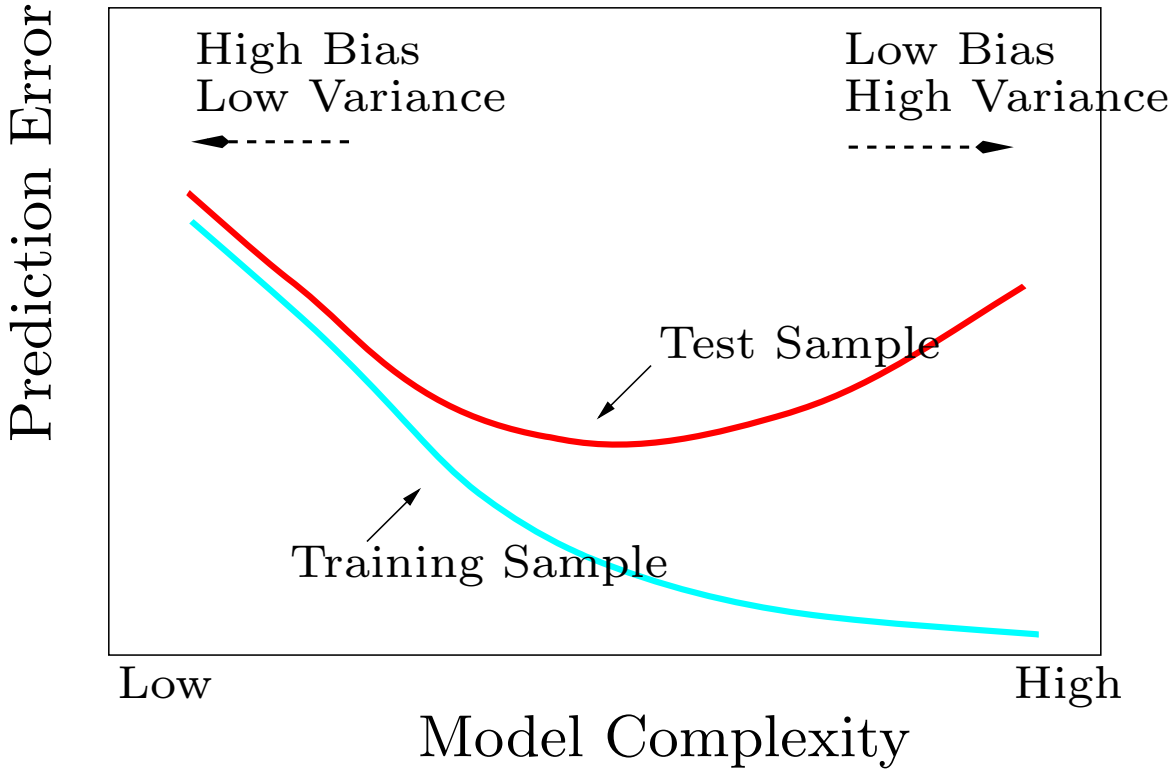
Overfitting and underfitting



The higher the degree of the polynomial, the more degrees of freedom. These estimators have high variance.

Typical overfitting means that error on the training data is very low, but error on new instances is high.

The Key Idea of the Course!!



Complex enough to find the signal, but not so complex that you chase the noise in the training data, only the signal will help you predict new Y given new X.

Bias-Variance Trade-Off

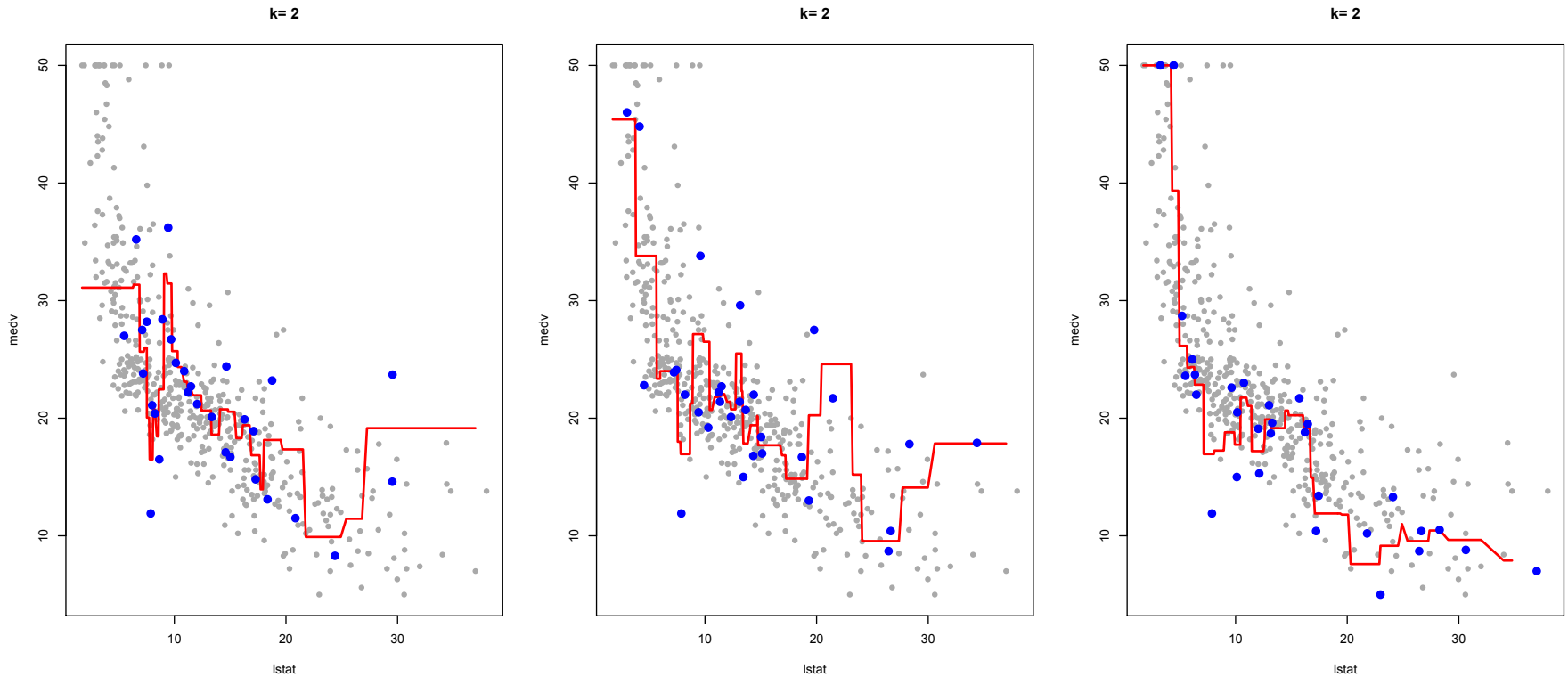
Bias-Variance Trade-Off

Why do complex models behave poorly in making predictions?

Let's start with an example...

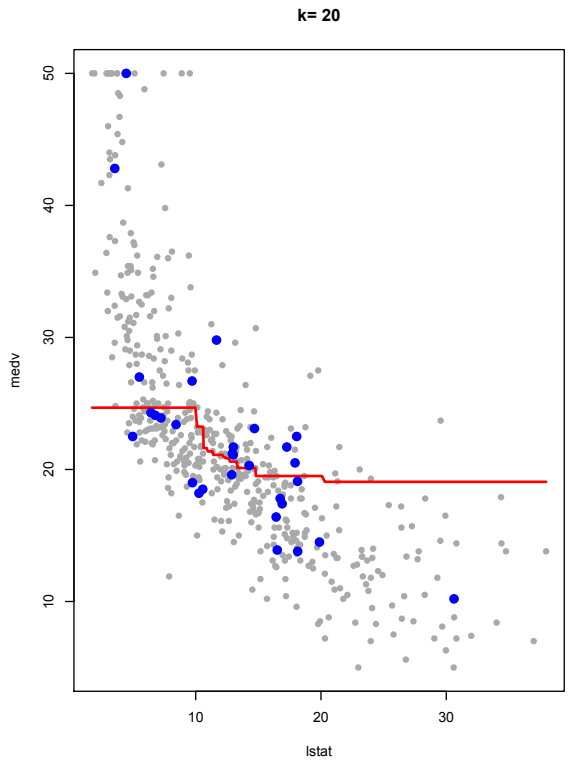
- In the Boston housing example, I will randomly choose 30 observations to be in the training set 3 different times...
- for each training set I will estimate $f(\cdot)$ using the k-nearest neighbors idea... first with $k = 2$ and them with $k = 20$

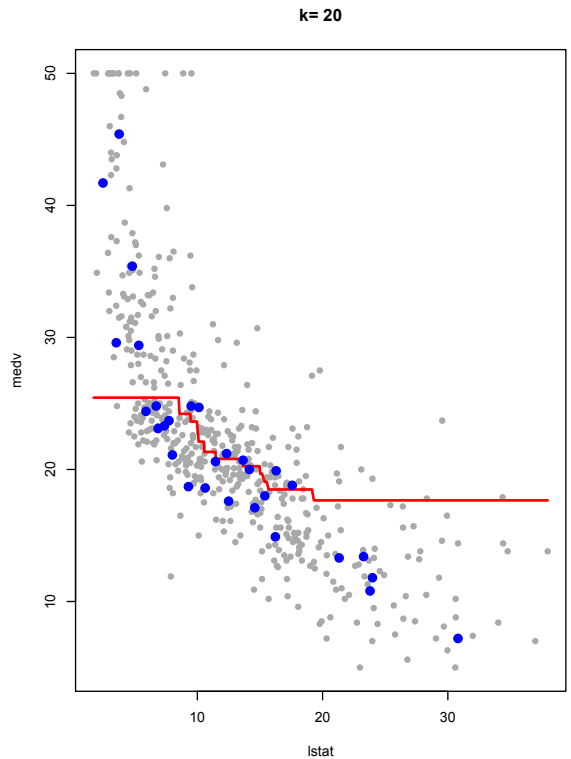
$k = 2$ High variability...

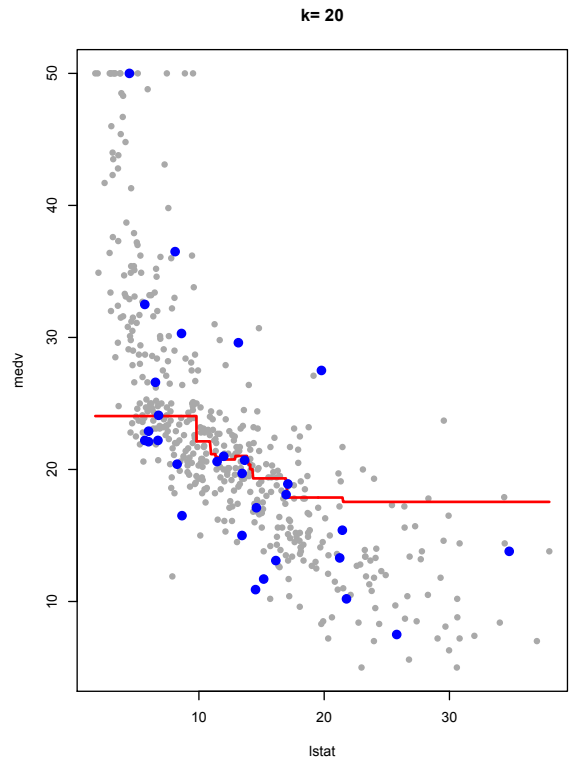


(blue points are the training data used)

$k = 20$ Low variability . . . but BIAS!!







(blue points are the training data used)

What did we see here?

- When $k = 2$, it seems that the estimate of $f(\cdot)$ varies a lot between training sets...
- When $k = 20$ the estimates look a lot more stable...

Now, imagine that you are trying to predict *medv* when *lstat* $= 20\ldots$

compare the changes in the predictions made by the different training sets under $k = 2$ and $k = 20\ldots$

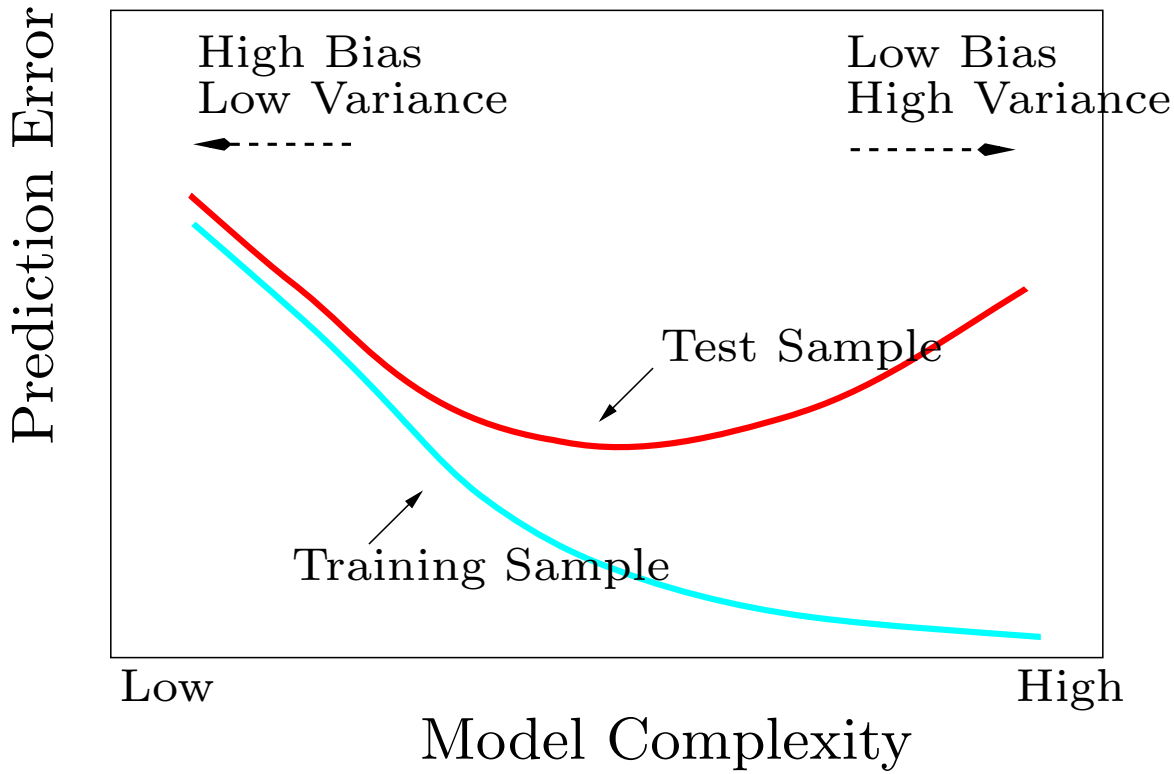
What do you see?

Bias-Variance Trade-Off

- This is an illustration of what is called the *bias-variance trade-off*.
- In general, simple models are trying to explain a complex, real problem with not a lot of flexibility so it introduces bias... on the other hand, by being simple the estimates tend to have low variance
- On the other hand, complex models are able to quickly adapt to the real situation and hence lead to small bias... however, if *too adaptable*, it tends to vary a lot, i.e., high variance.

Bias-Variance Trade-Off

Once again, this is the key idea of the course!!



Bias-Variance Trade-Off

Let's get back to our original representation of the problem... it helps us understand what is going on...

$$Y = f(X) + \epsilon$$

- We need flexible enough models to find $f(\cdot)$ without imposing bias...
- ... but, too flexible models will "chase" non-existing patterns in ϵ leading to unwanted variability

Data splitting

If we have a lot of data, we can split it into three parts:

- train set (50%) — used for fitting models
- validation set (25%) — used for model selection
- test set (25%) — assessment of the selected model

How many points to use?

- Too few training points, learned model is bad
- Too few test points, you never know if you reached a good solution

How does error as a function of number of training examples for a fixed model complexity behave?

Error estimators

Be careful!!!

Test set only unbiased if you never never ever ever do any any any any learning on the test data.

For example, if you use the test set to select
the degree of the polynomial... no longer unbiased!!!

Cross-validation

Using a validation-set to evaluate the performance of competing models has two potential drawbacks:

1. the results can be highly dependent on the choice of the validation set... what samples? how many?
2. by leaving aside a subset of data for validation we end up estimating the models with less information. It is harder to learn with fewer samples and this might lead to an overestimation of errors.

Cross-Validation is a refinement of the validation strategy that help address both of these issues.

Leave-One-Out (LOO) Cross-Validation

Assume we have n observations in our dataset $\mathcal{D}$.
Define the validation set by choosing only one observation j,
call $\mathcal{D}_{-j}$ the data set with j-th observation removed.
Learn $\hat{f}_{-j}$ on the dataset $\mathcal{D}_{-j}$.

Estimate true error on predicting y_j.

- $(y_i - \hat{f}_{-j}(x_j))^2$ is an unbiased estimator of the true error

LOO cross validations averages over all the data points

$$\text{err}_{LOO} = \frac{1}{n}\sum_j (y_j - \hat{f}_{-j}(x_j))^2$$

- For each data point you leave out, learn a new function $\hat{f}_{-j}$

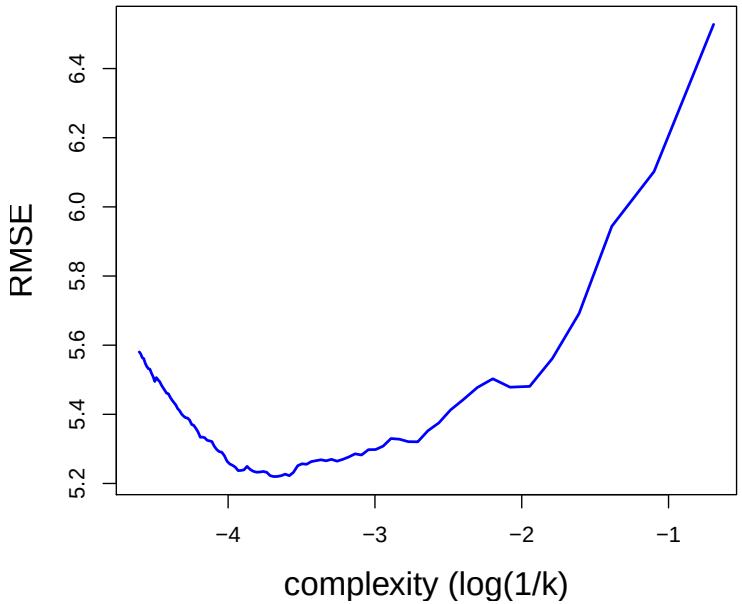
LOO cross validation is (almost) unbiased estimate of
the true error of $\hat{f}$!

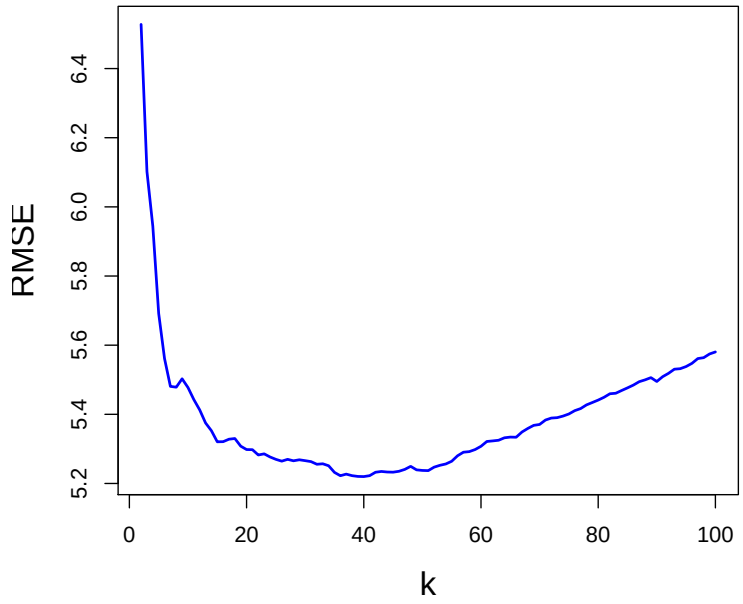
- Great news! Use LOO error for model selection!

When computing LOOCV error, we only use n-1 data points

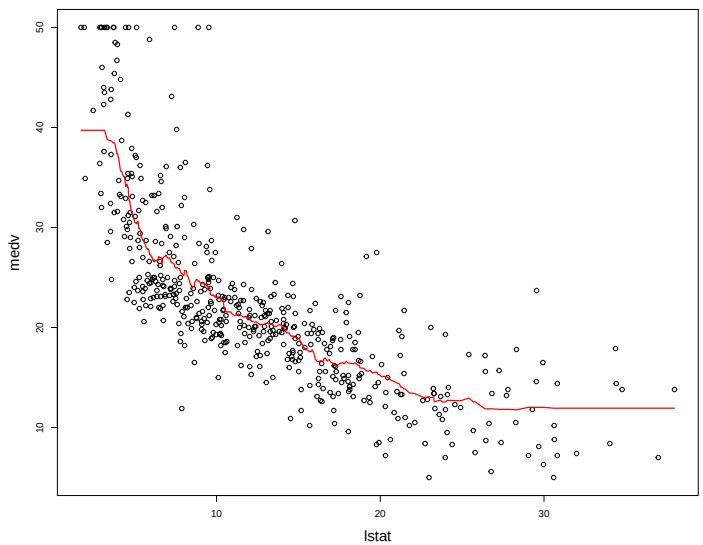
- So it is not estimate of true error of learning with n data points!
- Usually pessimistic, though - learning with less data typically gives worse answer.

Example: Boston Data





Min is at about $k = 40$.



Computational cost of LOO

Suppose you have 100,000 data points.
You implemented a great version of your learning algorithm

- Learns in only 1 second

Computing LOO will take about 1 day!!!

- If you have to do for each choice of degree, it will take looooooong

Use k-fold cross validation

A solution to complexity of computing LOO.
Randomly divide training data into k equal parts.

- $\mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2 \cup \ldots \cup \mathcal{D}_k$

For each j

- Learn function $\hat{f}_{-\mathcal{D}_j}$ using data points not in $\mathcal{D}_j$
- Estimate error of $\hat{f}_{-\mathcal{D}_j}$ on validation set $\mathcal{D}_j$

$$\text{err}_{\mathcal{D}_j} = \frac{k}{n} \sum_{i \in \mathcal{D}_j} (y_i - \hat{f}_{-\mathcal{D}_j}(i))^2$$

k-fold cross validation error is an average over data splits:

$$\text{err}_{\text{k-fold}} = \frac{1}{k} \sum_j \text{err}_{\mathcal{D}_j}$$

Usually, k is 5 or 10

k-fold Cross Validation

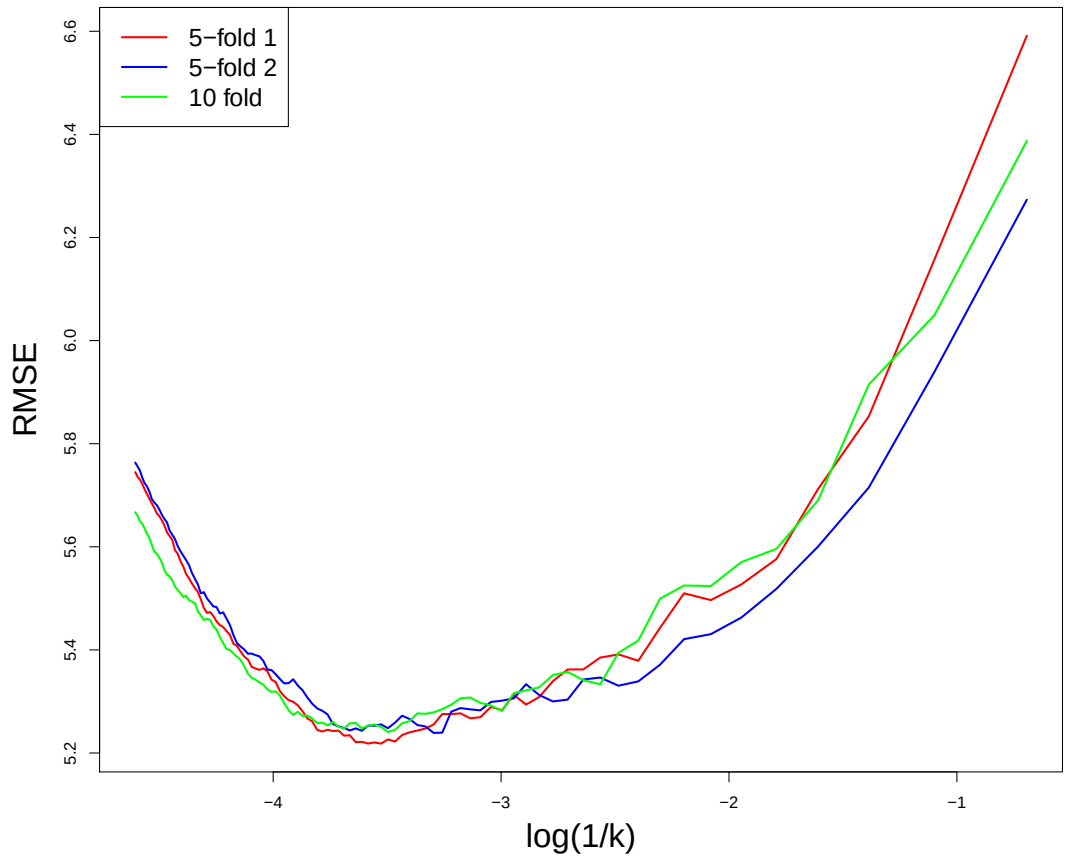
The usual choices are $k = 5$ and $k = 10\ldots$

We ran 5-fold twice.

Sometimes the results can be sensitive to the random choice of folds.

We ran 10-fold once.

We don't always get this much agreement!



LOOCV vs k-fold:

You might think that LOOCV is best if you have the time to run it.

However, with LOOCV the training data is almost the same everytime so that there is not as much variation on the fitted model as you would get if you really drew another sample. Hence the risk in prediction is underestimated.

5 or 10-fold CV is the industry standard !!!

More on k-Nearest Neighbors, $p > 1$

We have looked at simple examples of kNN (with one x!!).

In this section we look at kNN more carefully, in particular, how do you use kNN when x has p variables??!!

An important advantage of kNN is that it is feasible for *BIG DATA*, big n **and** big p.

The *k*-nearest neighbors algorithm will try to **predict** based on similar (close) records on the **training dataset**.

Remember, the problem is to guess a future value Y_f given new values of the covariates $X_f = (x_{1f}, x_{2f}, x_{3f}, \ldots, x_{pf})$.

kNN:

What do the Y's look like in the region around X_f?

We need to find the k observations in the training dataset that are close to X_f. How? "Nearness" to the i^{th} neighbor can be defined by (euclidean distance):

$$d_i = \sqrt{\sum_{j=1}^{p}(x_{jf} - x_{ji})^2}, \quad x_i \text{ in training data}$$

Prediction:

Take the average of the $Y's$ in the k-nearest neighborhood.
Average y_i corresponding to k smallest d_i.

Note:

- The distance metric used above is only valid for numerical values of X. When $X's$ are categorical we need to think about a different distance metric or perform some manipulation of the information.

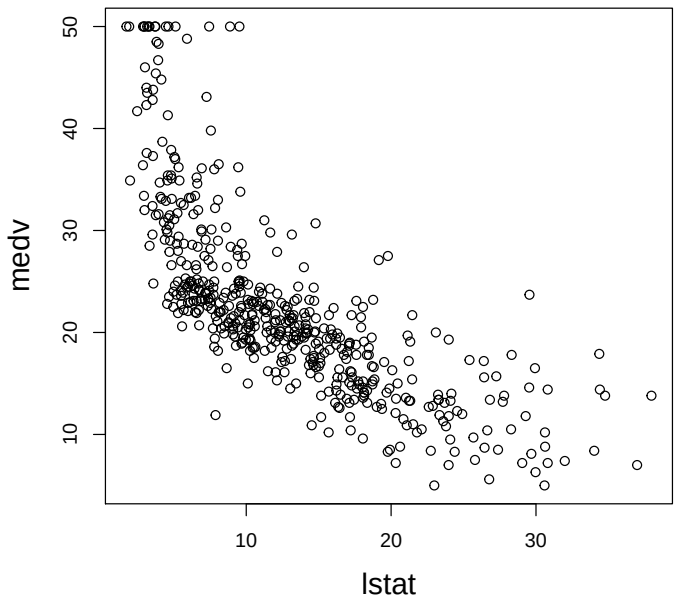
- The scale of X also will have an impact. In general it is a good idea put the $X's$ in the same scale before running kNN.

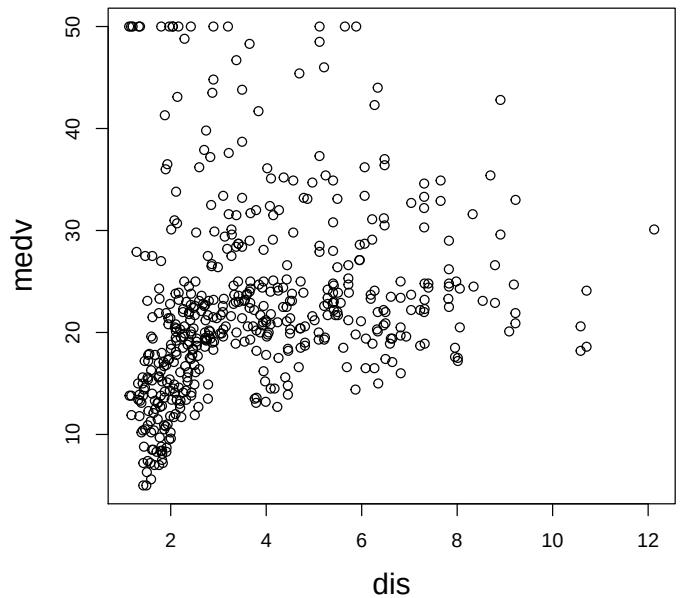
What do we mean by scale?

If weight is in pounds you get one distance, if weight is in kilograms you get a different number!!

To see how this works, let's do the Boston example with $p = 2$:

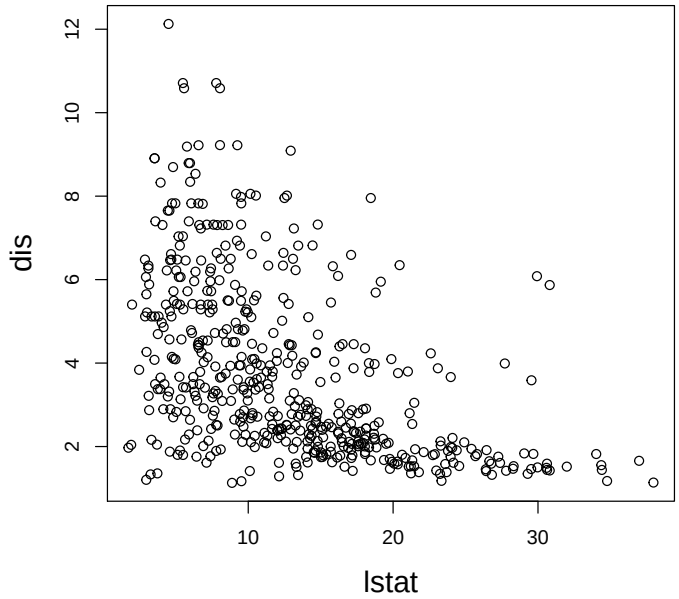
x_1=lstat\ x_2=dis: weighted mean of distances to five Boston employment centres

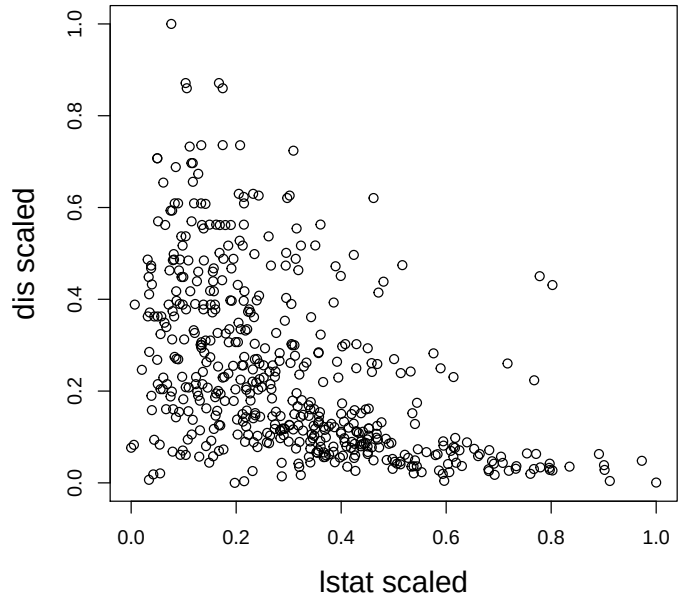




Hmm. how is y=medval related to x_2=dis ?

Left: x_1 vs. x_2.





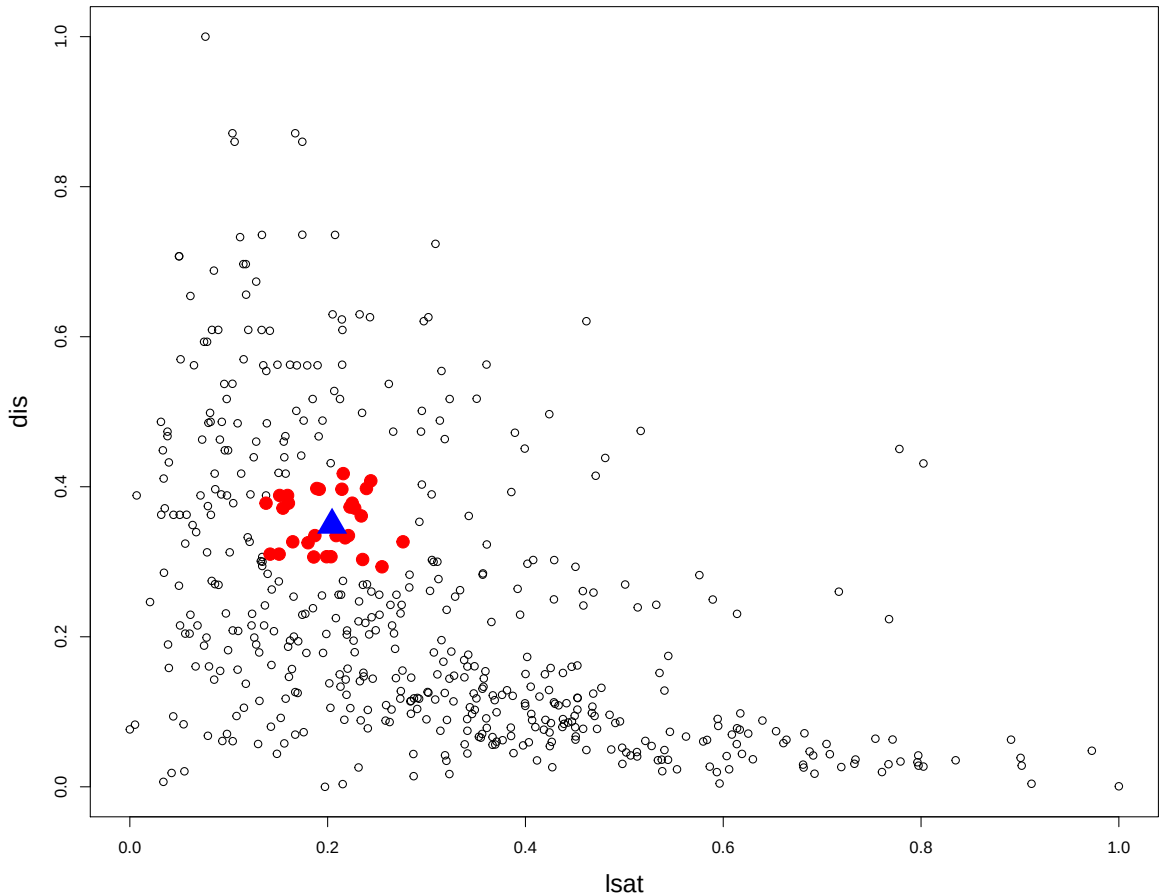
Right:

We rescale each $x \Rightarrow \dfrac{x - \min(x)}{(\max(x) - \min(x))}$

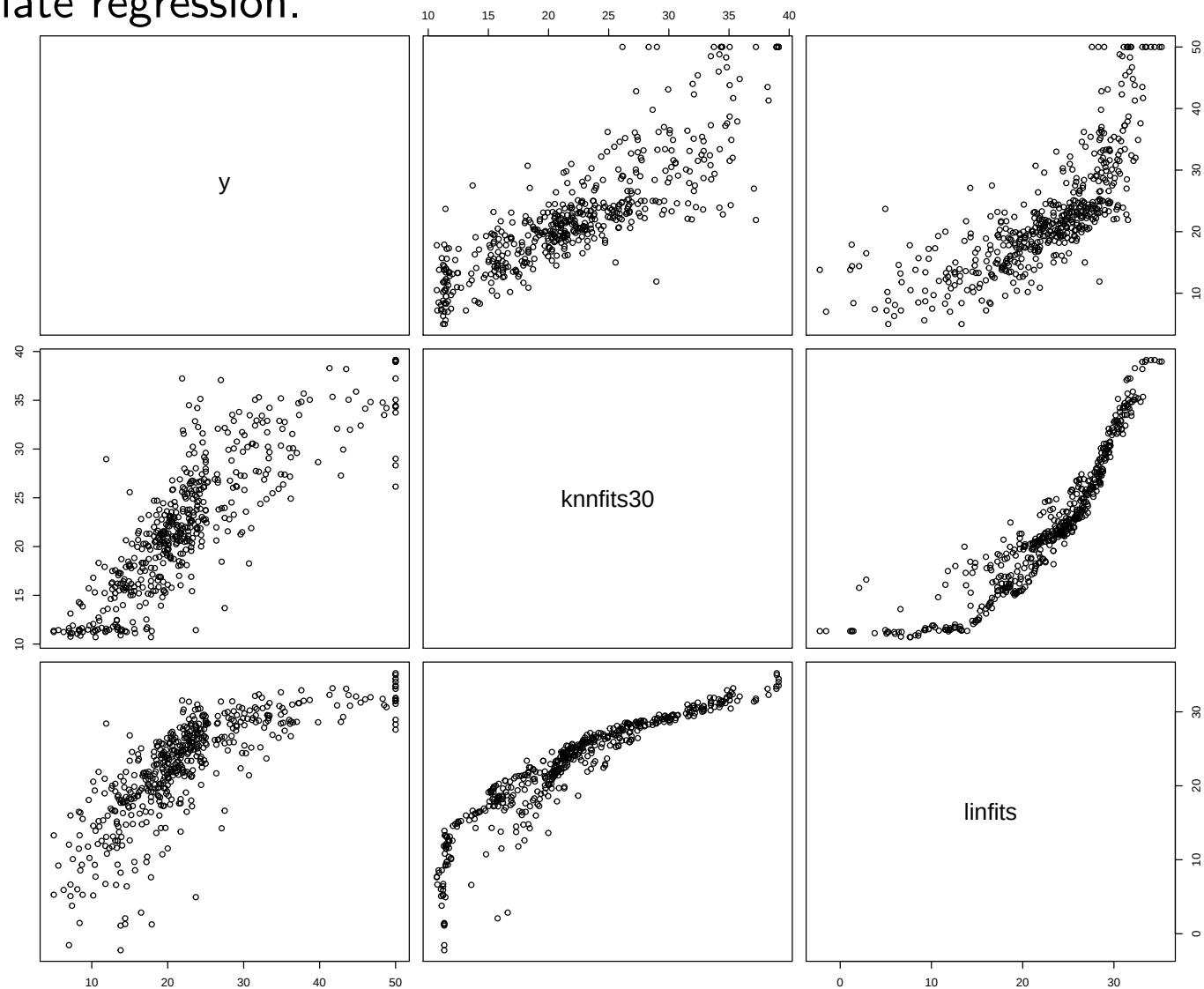
Another common rescaling is $x \Rightarrow \dfrac{(x - \bar{x})}{\mathrm{sd}(x)}$

This kind of scaling is very common in practice. However, you can have all kinds of problems. Suppose an x is heavily skewed ???

To predict y at the blue triangle, we average the y values corresponding to the red points.



Here are the in-sample fits using k=30, compared with y and the fits from a bivariate regression.

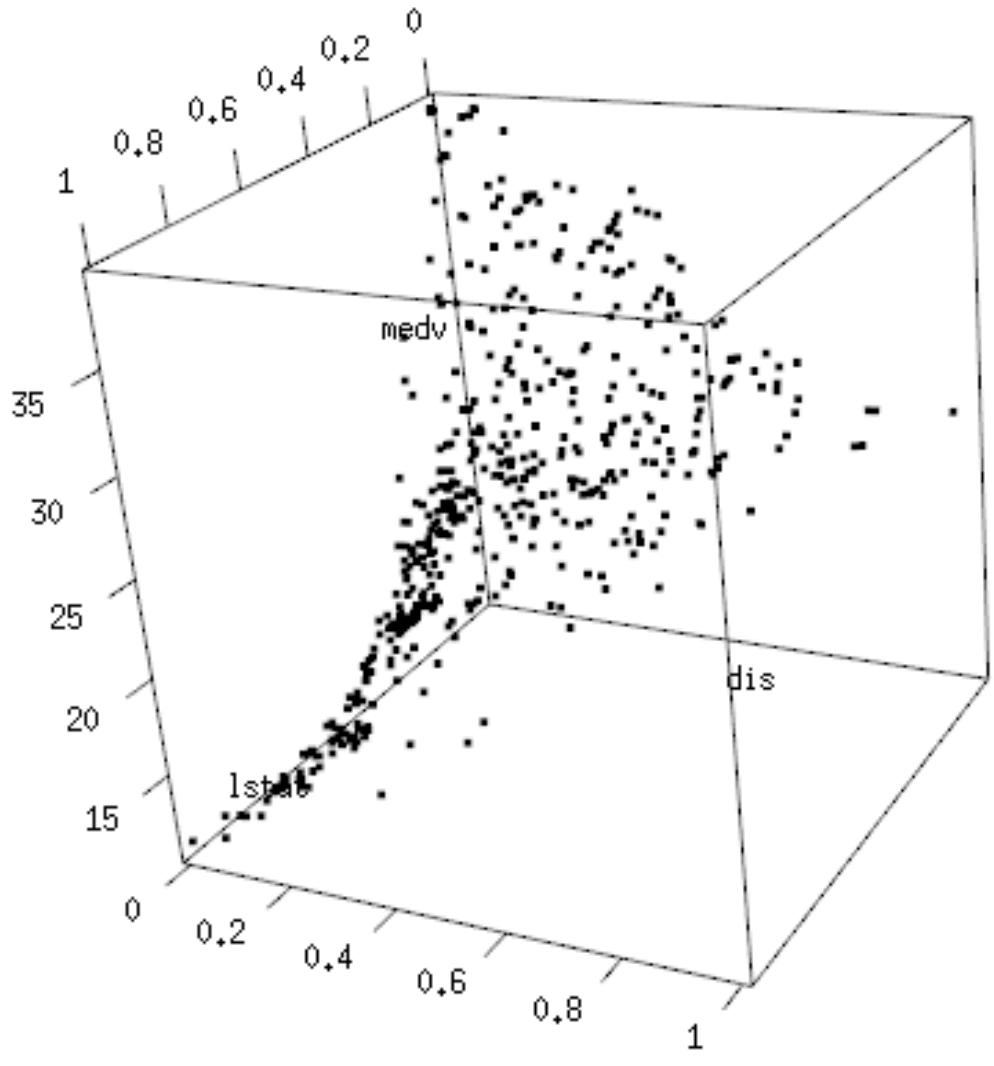


Here are the correlations corresponding to the plots.

	y	knnfits30	linfits
y	1.00	0.84	0.75
knnfits30	0.84	1.00	0.91
linfits	0.75	0.91	1.00

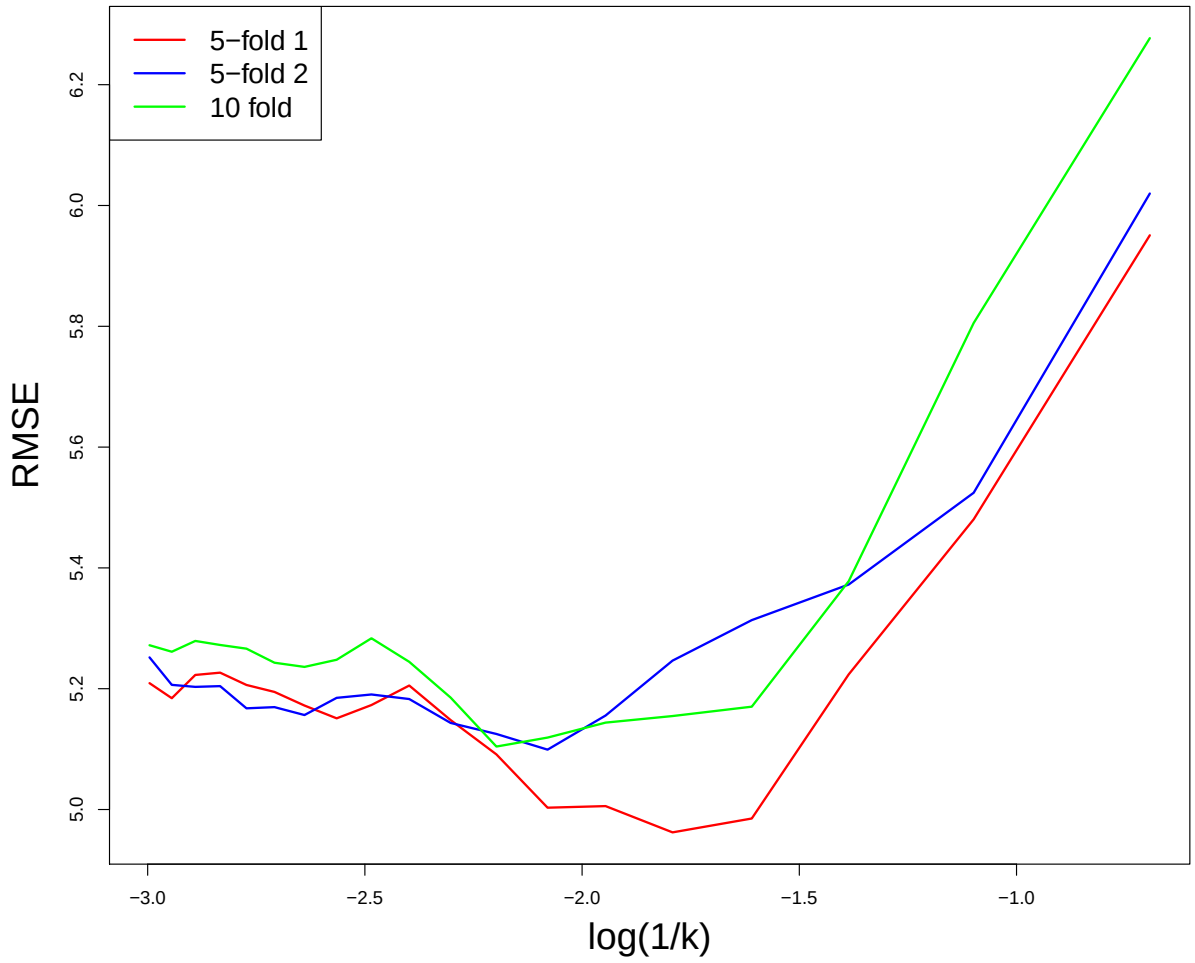
In sample, knn30 looks better than linear regression.

A big R-squared can just mean you overfit !!



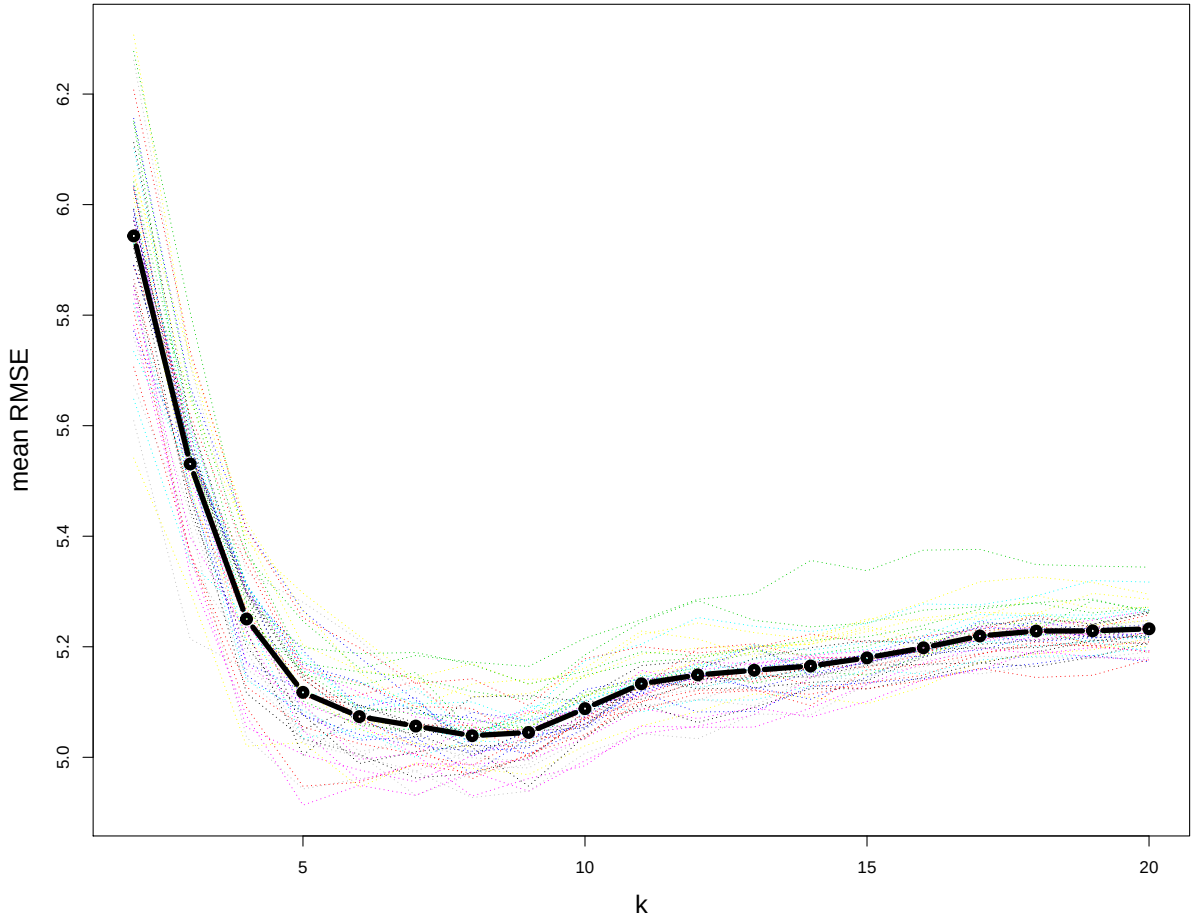
3-D visualization is not easy!!!

Let's do the CV and see what works out of sample.



Seems to indicate a much smaller k (than when we just used `lstat`), but it is very noisy.

Let's average many CV's.

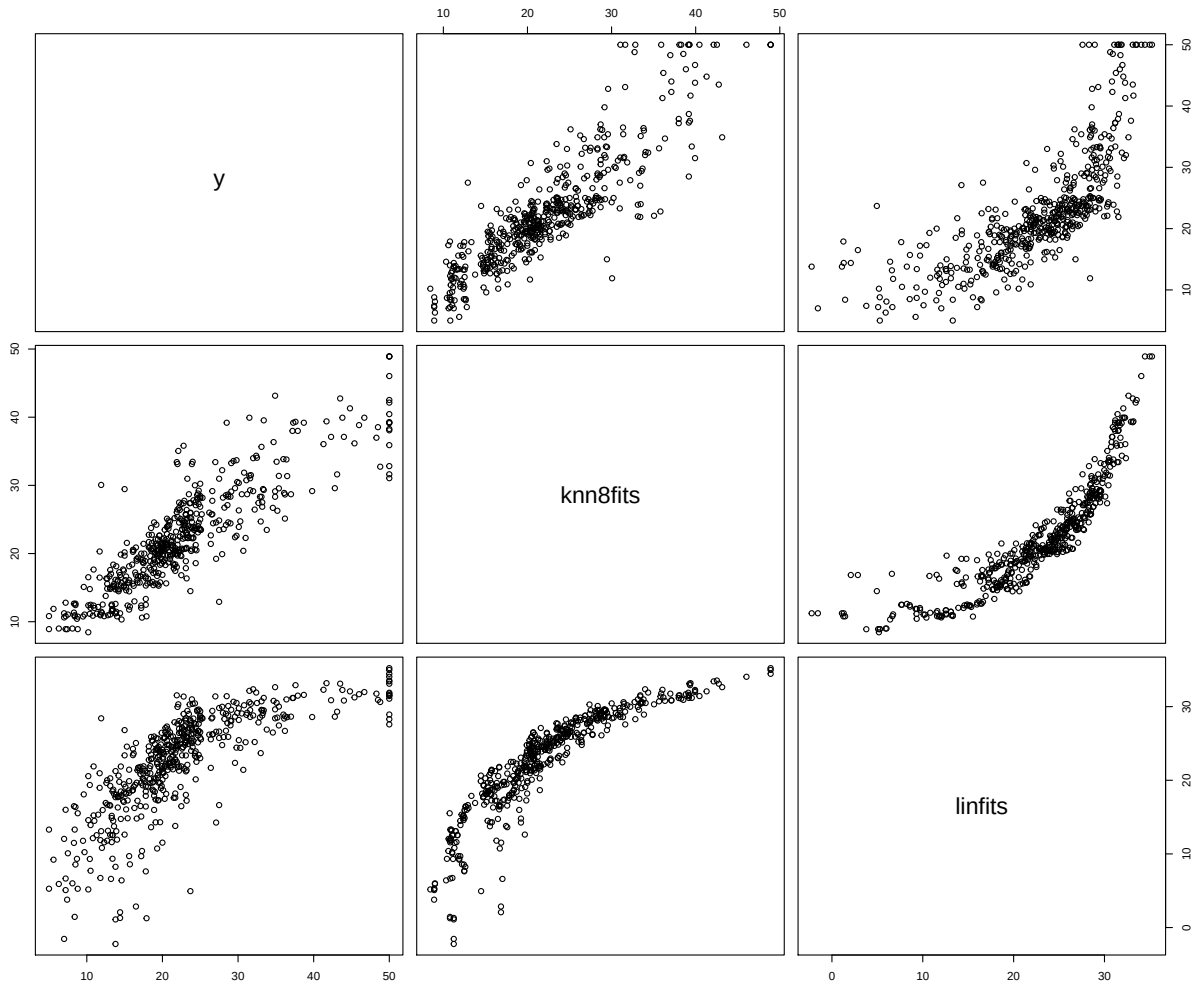


Indicates a k of about 8, which is m uch smaller than when we just had `lstat`. Minimum MSE is smaller than when we just had `lstat` but not my much, 5, versus 5.2.

What do we do next, after having used cross-validation to choose a model (or tuning parameter)?

It may be an obvious point, but worth being clear: we now fit our estimator to the entire training set (x_i, y_i), $i = 1, \ldots, n$, using the selected model.

Refit using all the data and $k = 8$.



Here are the correlations corresponding to the plots.

```
              y knn8fits linfits
y          1.00     0.88    0.75
knn8fits 0.88     1.00    0.87
linfits  0.75     0.87    1.00
```

In sample, knn8 looks better than linear regression.

And now we care!!

Matching

A lot of data-mining methods work by matching.

- Which ones are most like you ??
- Which training x are most like x_f ??
- Shoppers **like you** have bought ...

"Like"" means a choice of distance, and getting the distance right in high dimensions can be very hard.

Big p, big n

In principle, we can use kNN for large n and p.

kNN **is** a very widely used technique.

However, you should always be scared!!!

What does distance mean for big p?

Remember the **curse of dimensionality**!!!

Summary of kNN

kNN is a powerfull, widely used, intutitive technique.

We have used it to illustrate the Bias-Variance tradeoff which is **a fundamental concept**.

Note:

- Choosing the distance can be tricky.
- Using distance in high dimensions can be tricky.
- Rescaling all the (numeric) x's is common.

Doing CV with a Bigger n

The Boston housing data we have been using as an example only has $n = 506$ observations.

While the big ideas are the same, some things will work out differently with larger n.

Let's do $n = 20,000$ to illustrate.

The key differences will be:

- We won't have to rerun the CV many times and average.
  With the larger sample size, you will get much less variation in the CV results.

- We will start by leaving out a **test** data set.

  We will use CV on the remaining data to make modeling decisions, and then apply our data to the test data to see you well we predict out of sample.

When we refit using all the Boston data, we did not have any true out-of-sample data !!

kNN: California Housing

Data: Median home values in census tract plus the following information

- Location (latitude, longitude)
- Demographic information: population, income, etc...
- Average room/bedroom number, home age
- Let's start using just location as our $X's$... euclidean distance is quite natural here, right?

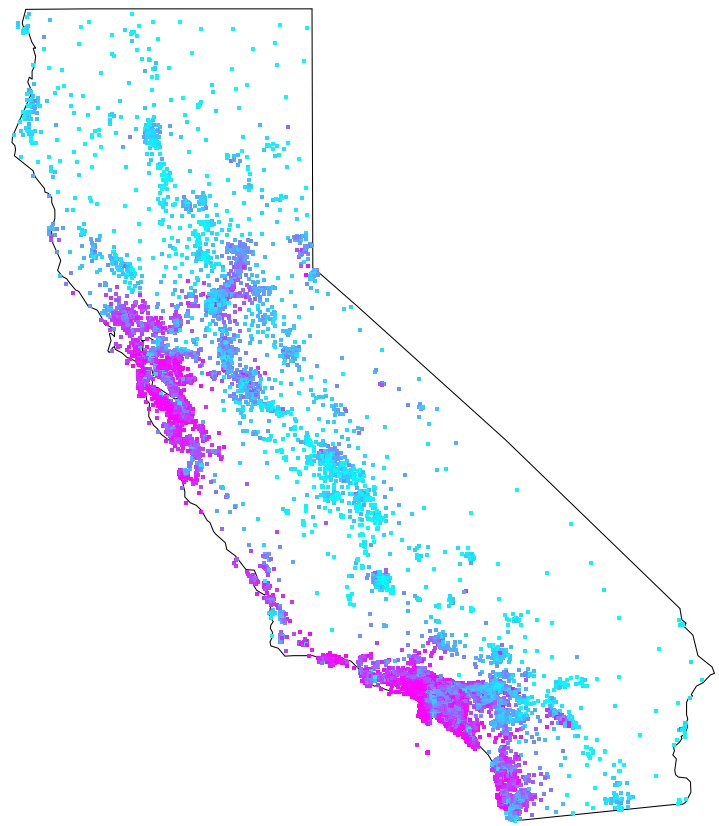
Goal: Predict $log(MedianValue)$

There are 20,640 observations and 8 x's.

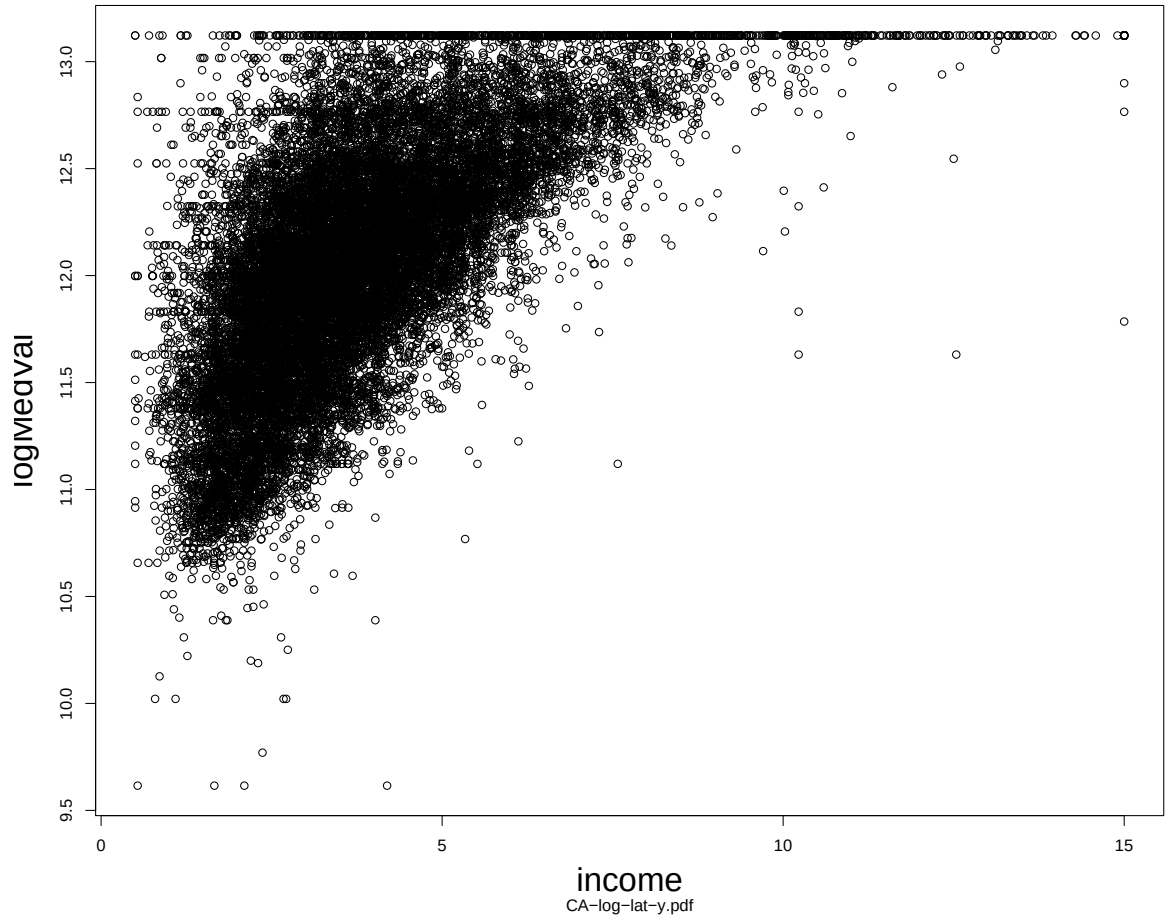
We should spend a long time plotting the data, but let's suppose we are in a hurry.

A couple of plots:

y=logMedVal
vs longitude and latitude.



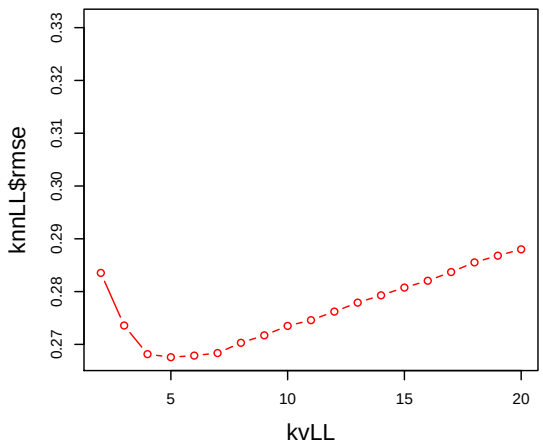
y=logMedVal
vs median income.

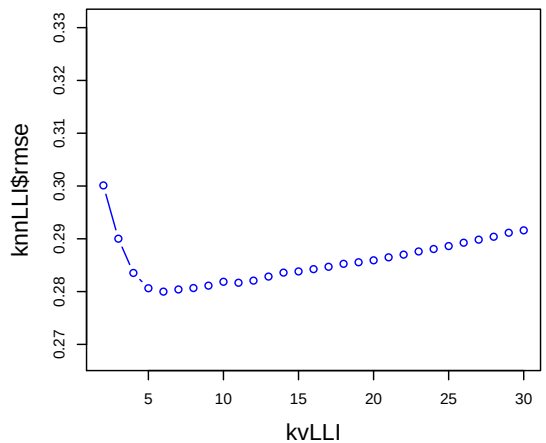


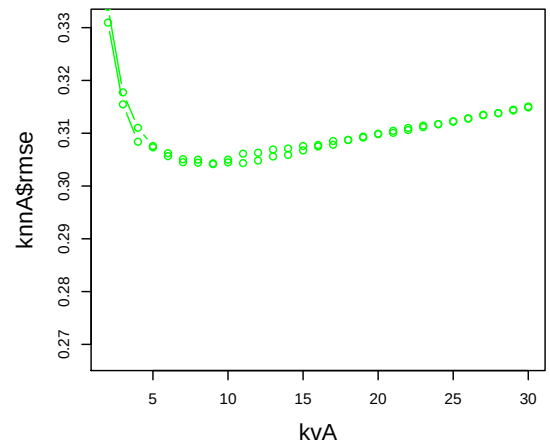
10,000 in train. Rest in test. Standardize each x: (x-m)/s.

Do 5-fold cross-validation on train.

Red: longitude and latitude
Blue: longitude and latitude and Income
Green: all 8 x's (2 runs).

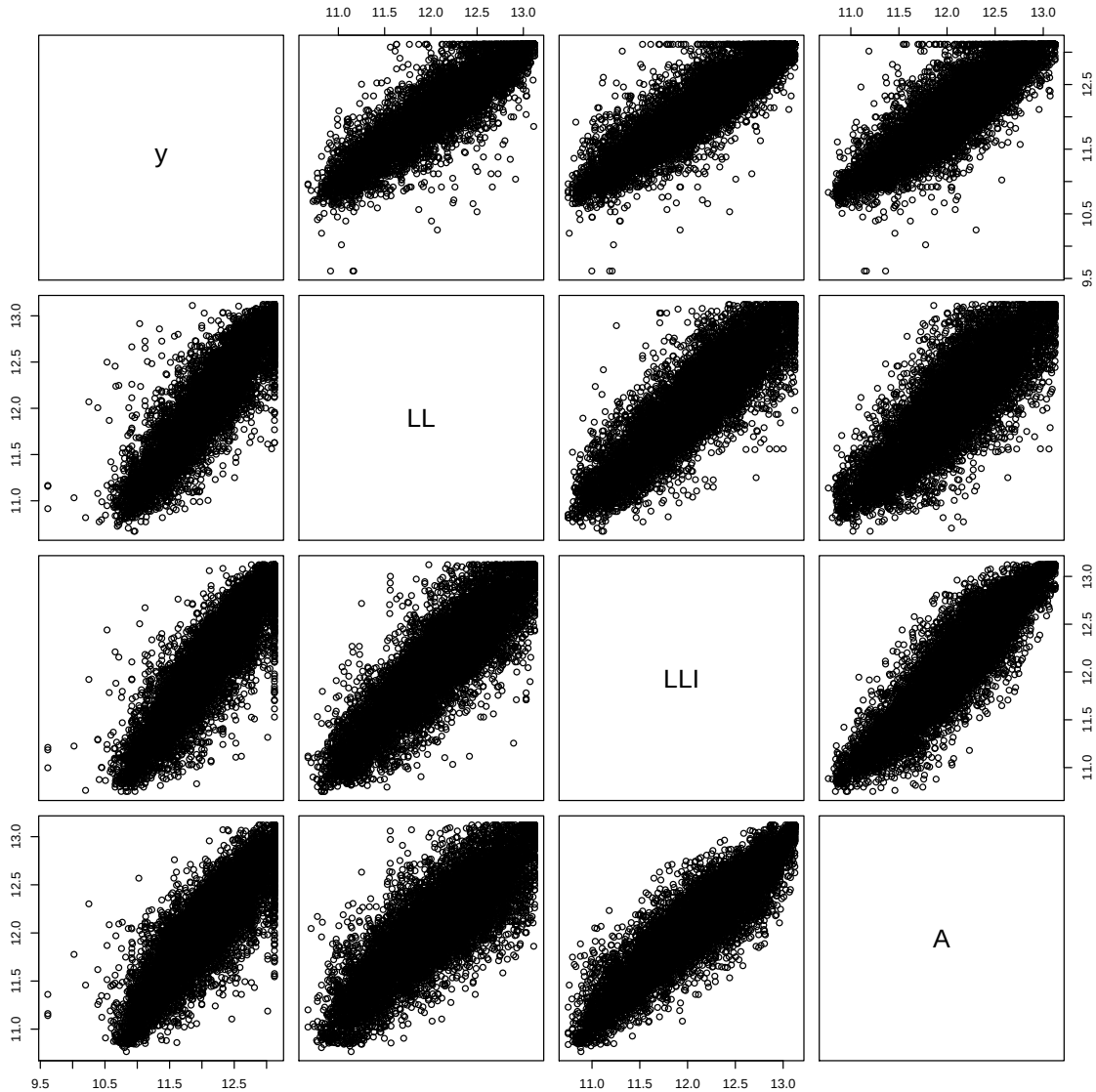






Pretty small k values.
All x is worst.

Using the best k, fit using all the train data, predict on the test.



And, here are the rmse's on the test data.

```
rmse test, long,lat:  0.2533981
rmse test, long,lat,income:  0.2628331
rmse test, all:  0.2897788
```

location, location, location.....