Introduction to Predictive Models
The Bias Variance Tradeoff
Cross Validation

Some of the figures in this presentation are taken from An Introduction to Statistical Learning, with applications in R (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani

Carlos Carvalho Mladen Kolar, and Rob McCulloch

1. Introduction to Predictive Models

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

In Data Mining terminology this is know as **supervised learning** (also called *Predictive Analytics*).

In general, a useful way to think about it is that Y and X are related in the following way:

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

The main purpose of this part of the course is to *learn or estimate* $f(\cdot)$ from data

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.

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

- $f(x)$: the part of Y you learn from X, *the signal*.
- ϵ: the part of Y you don't learn from X, *the noise*.

More generally,

we want the conditional distribution of Y given $X = x$.

Example: Boston Housing

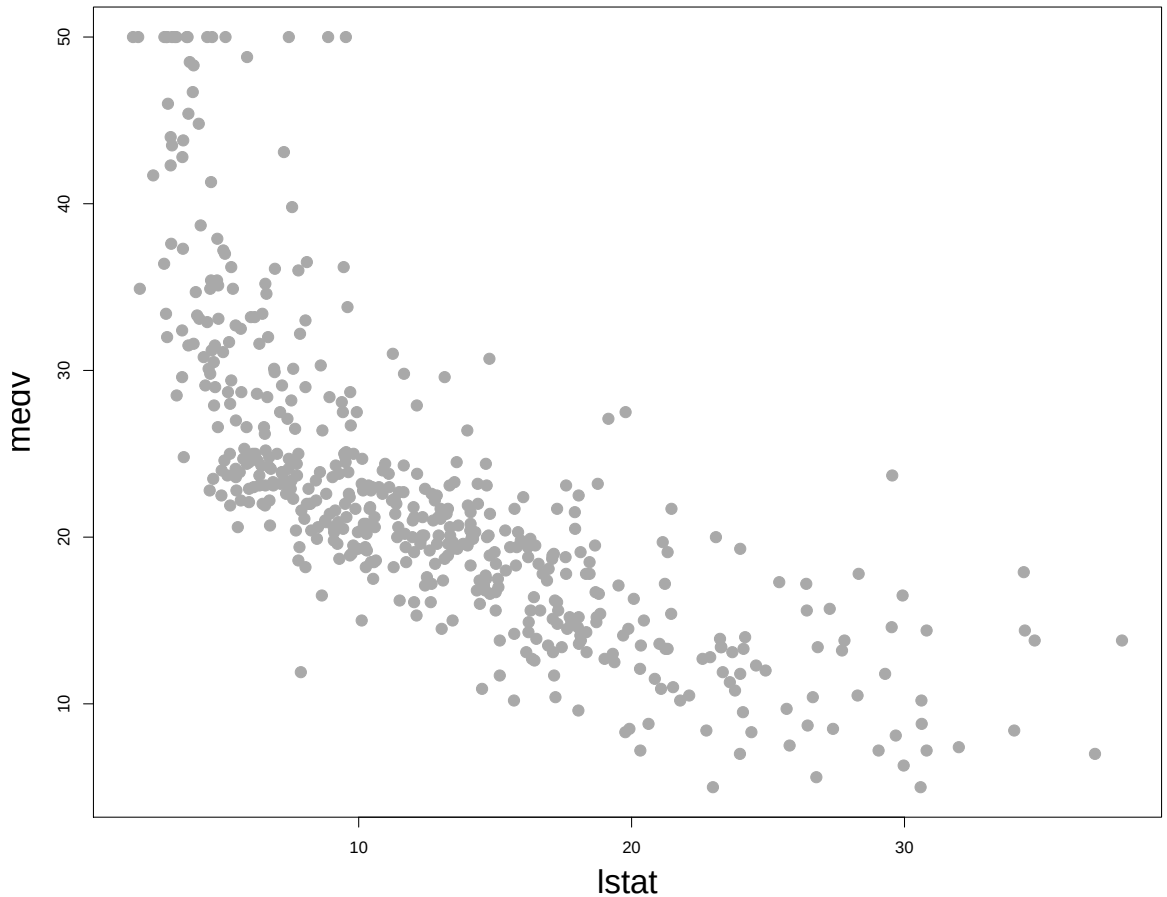
We might be interested in predicting the median house value as a function of some measure of social economic level... here's some data:

Each observation corresponds to a town in the Boston area.

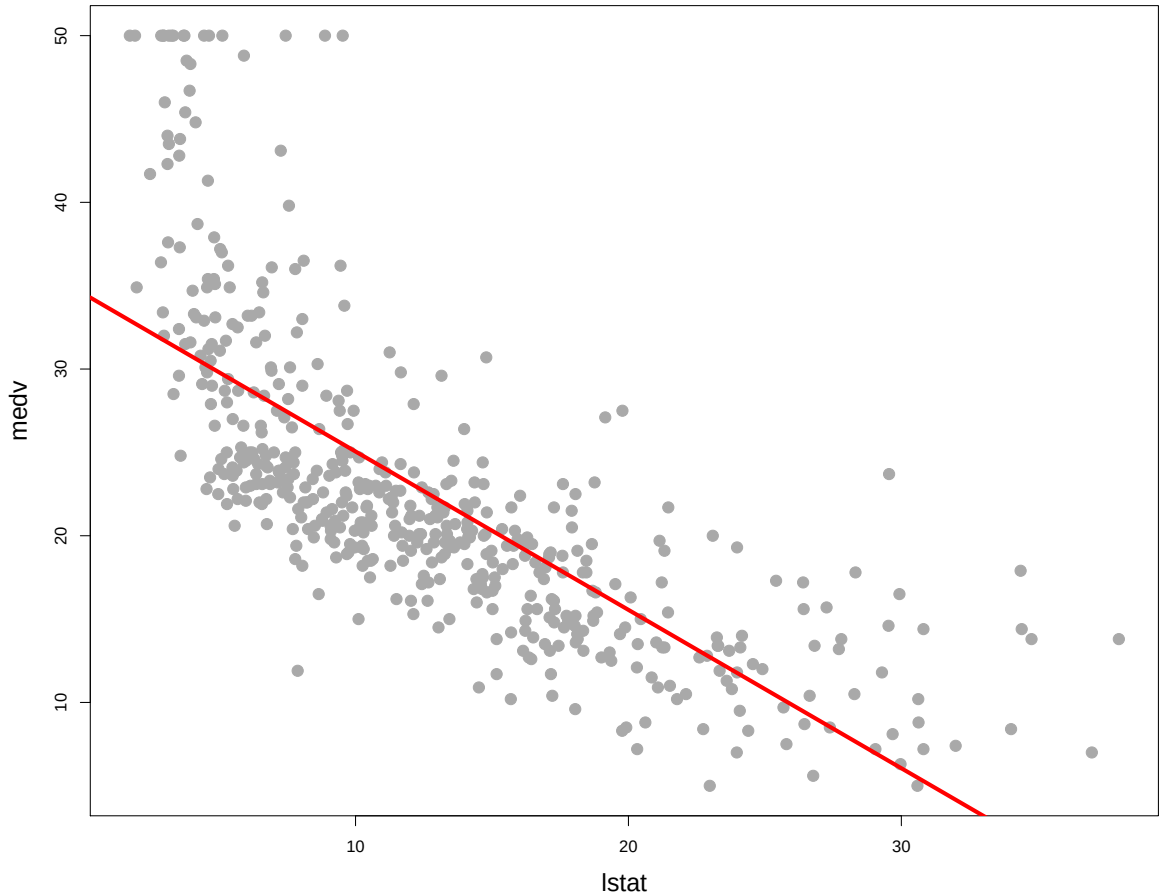
medv: median house value (data is old).

lstat: % lower status.

What should $f(\cdot)$ be?

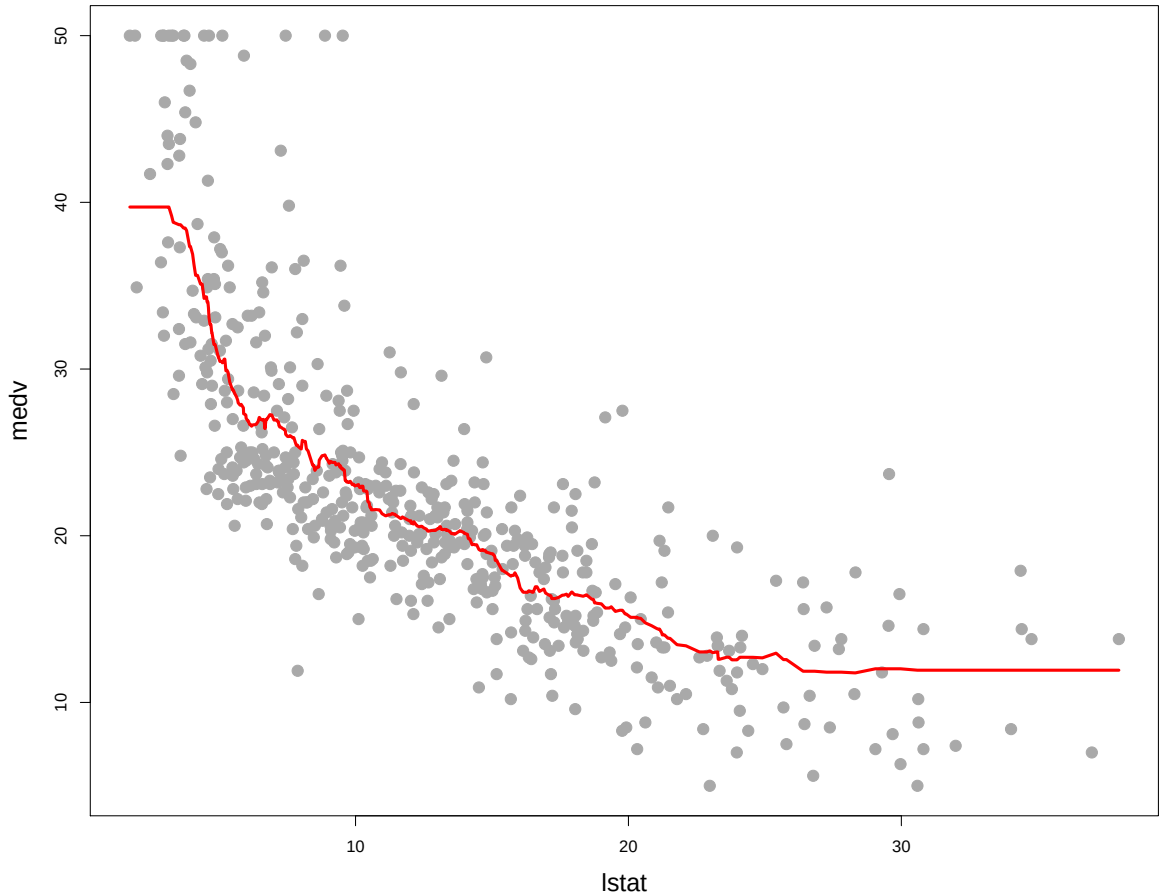


How about this...



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

or this?



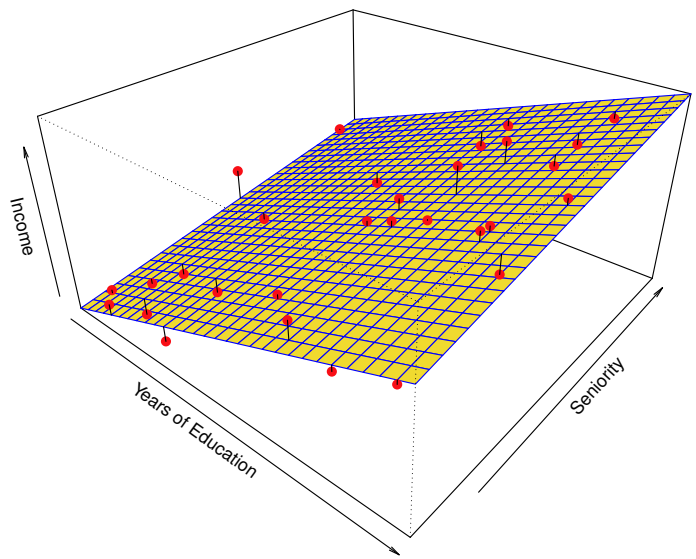
If *lstat* = 30 what is the prediction for *medv*?

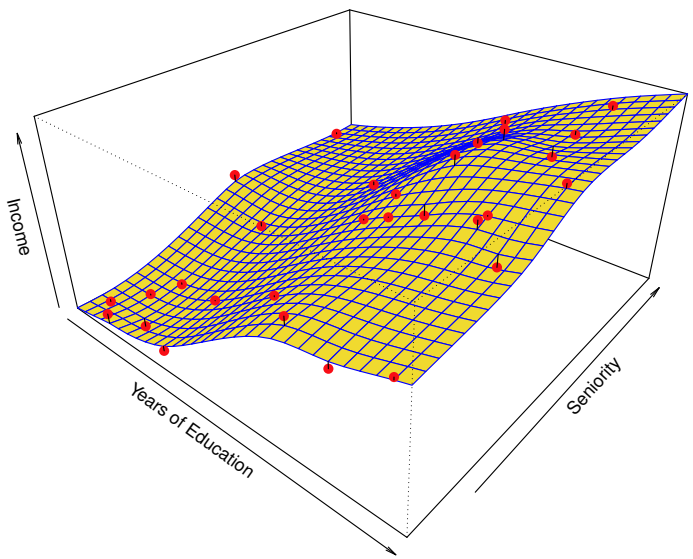
How do we estimate $f(\cdot)$?

- Using *training data*:

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

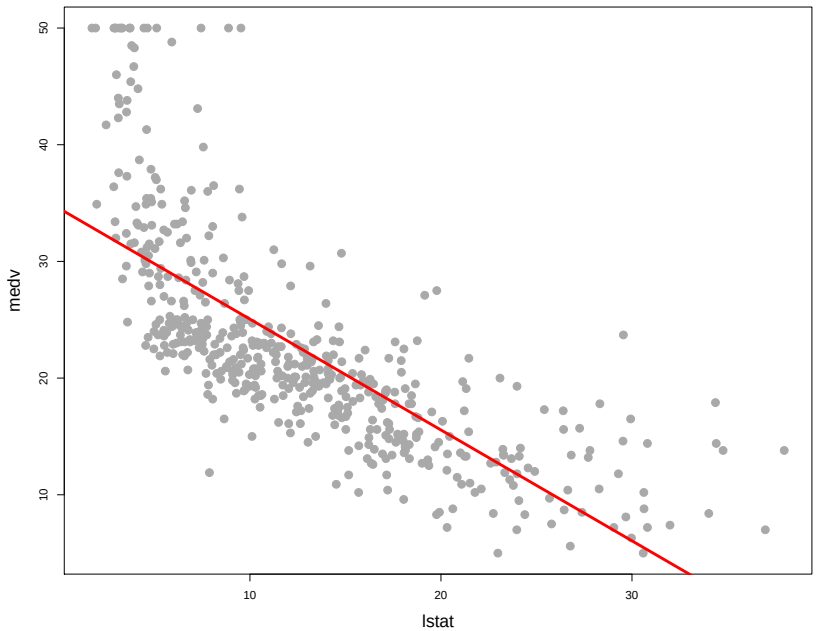
- We use a statistical method to *estimate* the function $f(\cdot)$
- Two general methodological strategies:
  1. simple parametric models (restricted assumptions about $f(\cdot)$)
  2. non-parametric models (flexibility in defining $f(\cdot)$)



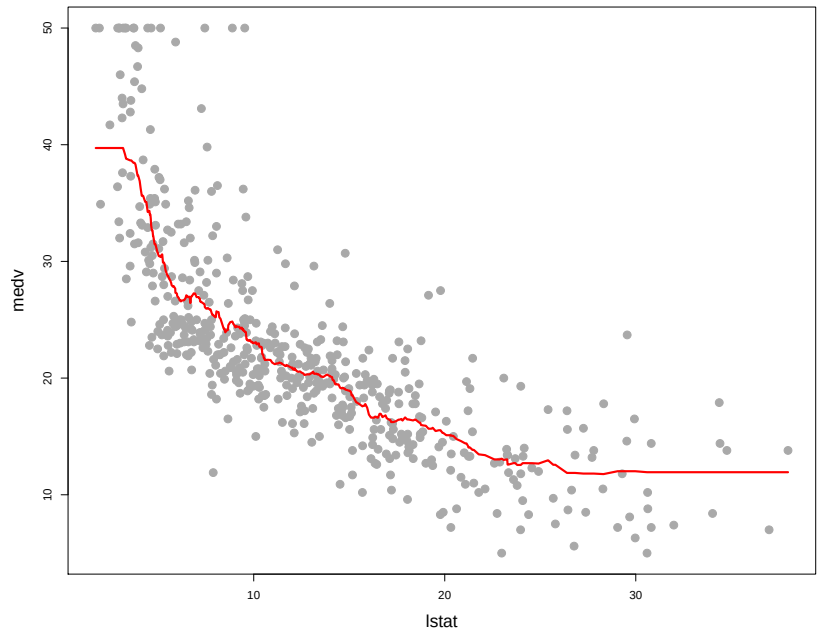


Back to Boston Housing

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



Non-Parametric Model
(k-nearest neighbors)



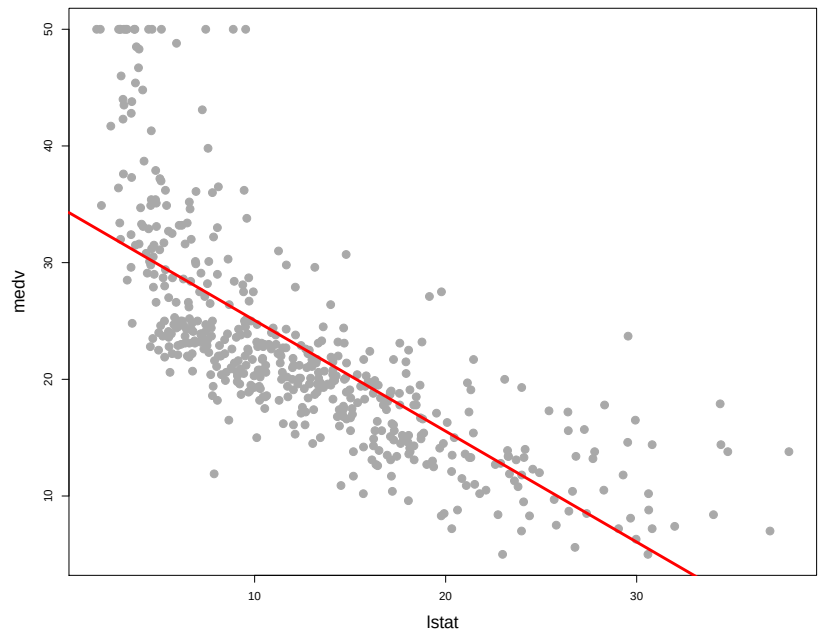
Simple parametric model:

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

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

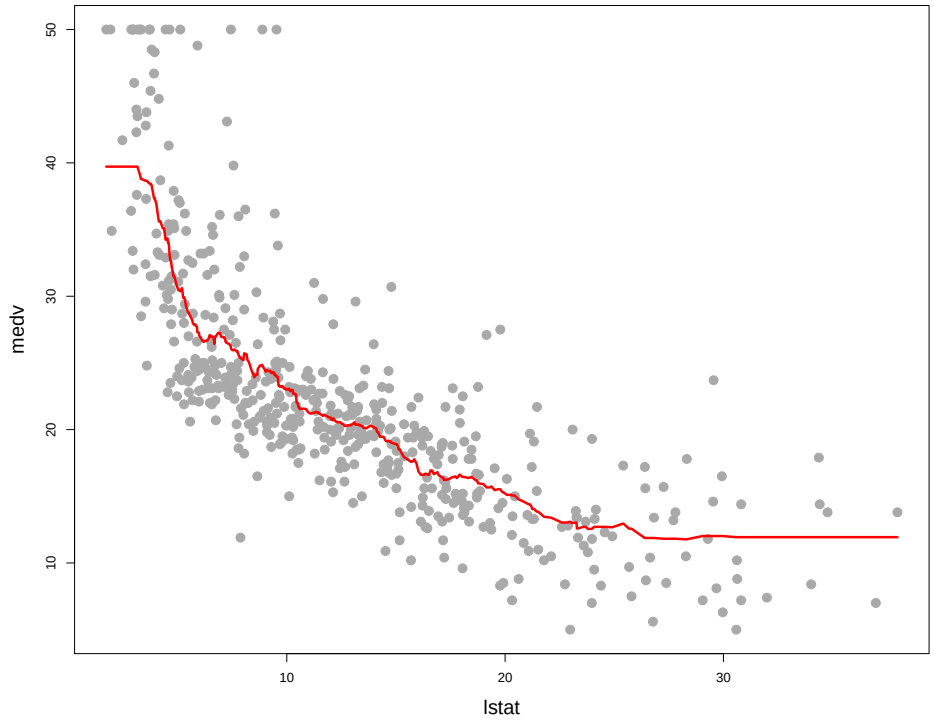
$$\hat{f}(x) = \hat{\alpha} + \hat{\beta}\, x$$

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



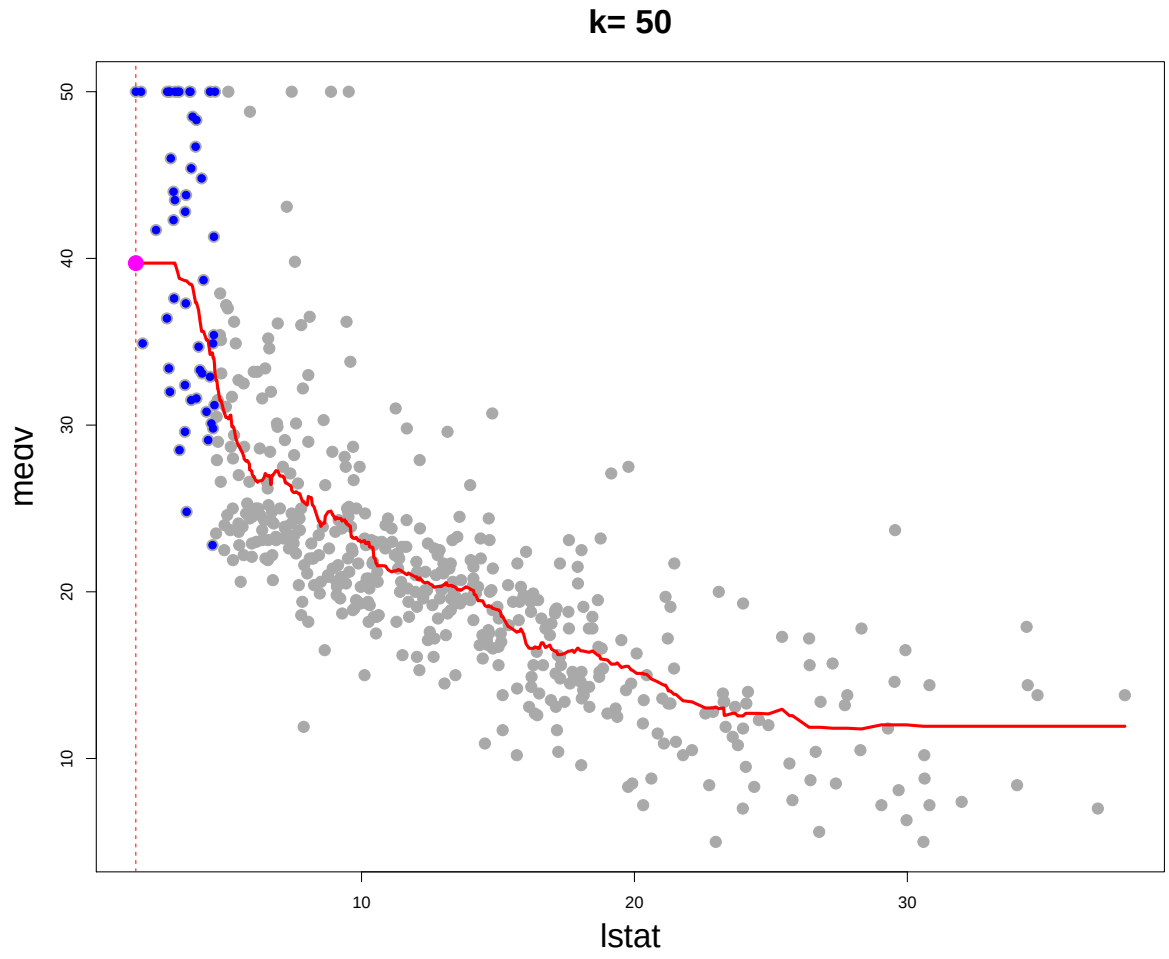
To get this estimate we used *kNN*
- k-nearest neighbors.

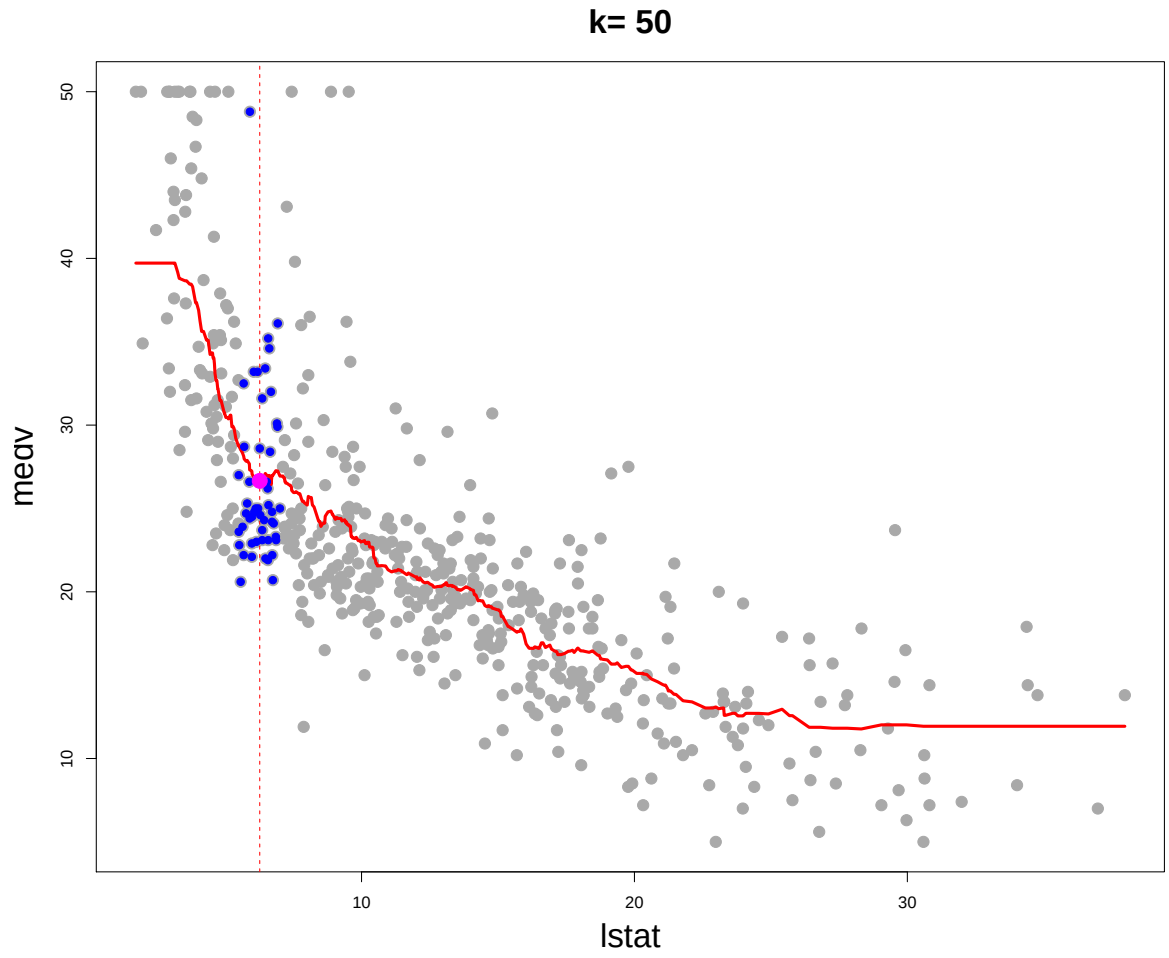
To estimate $f(x_f)$, average the y values for the k training observations with x *closest* to x_f.

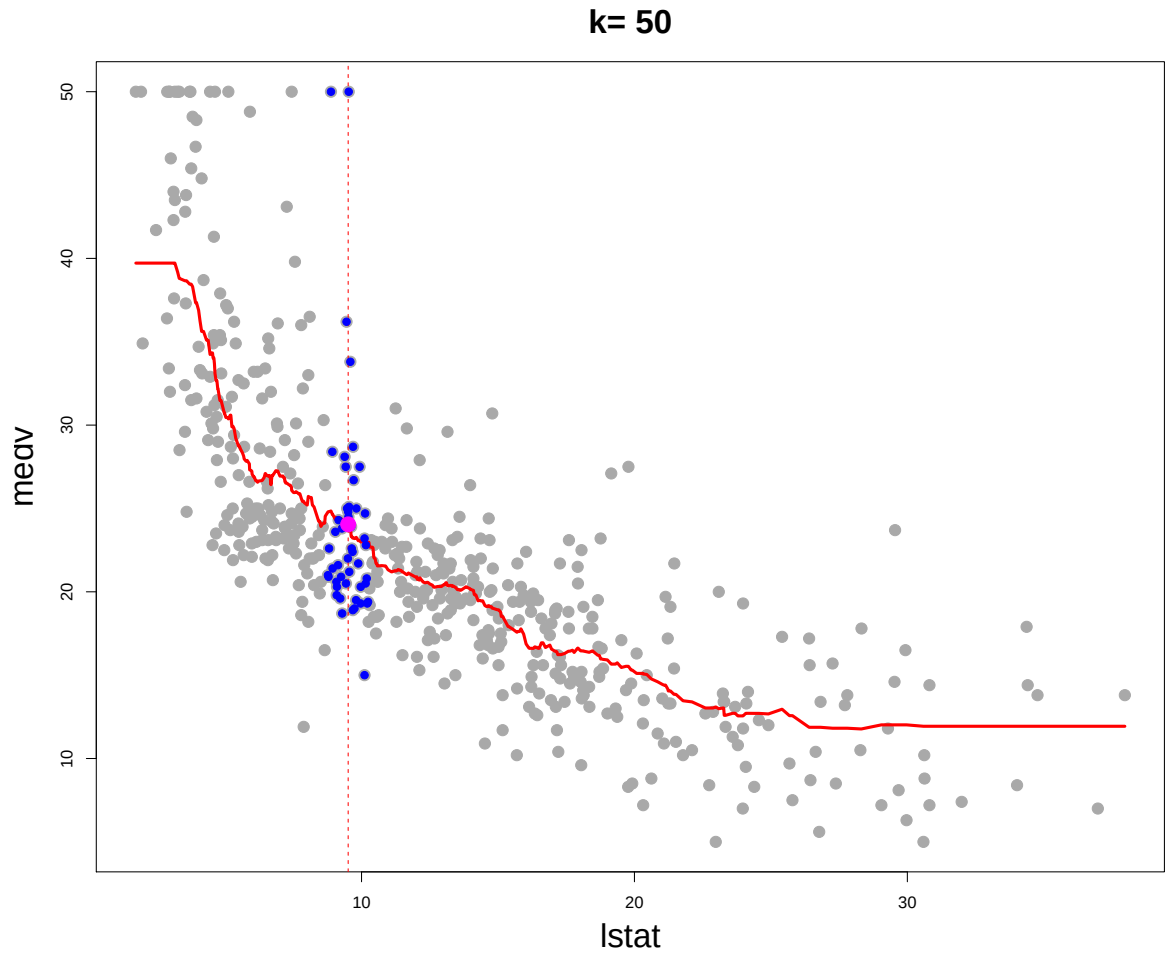


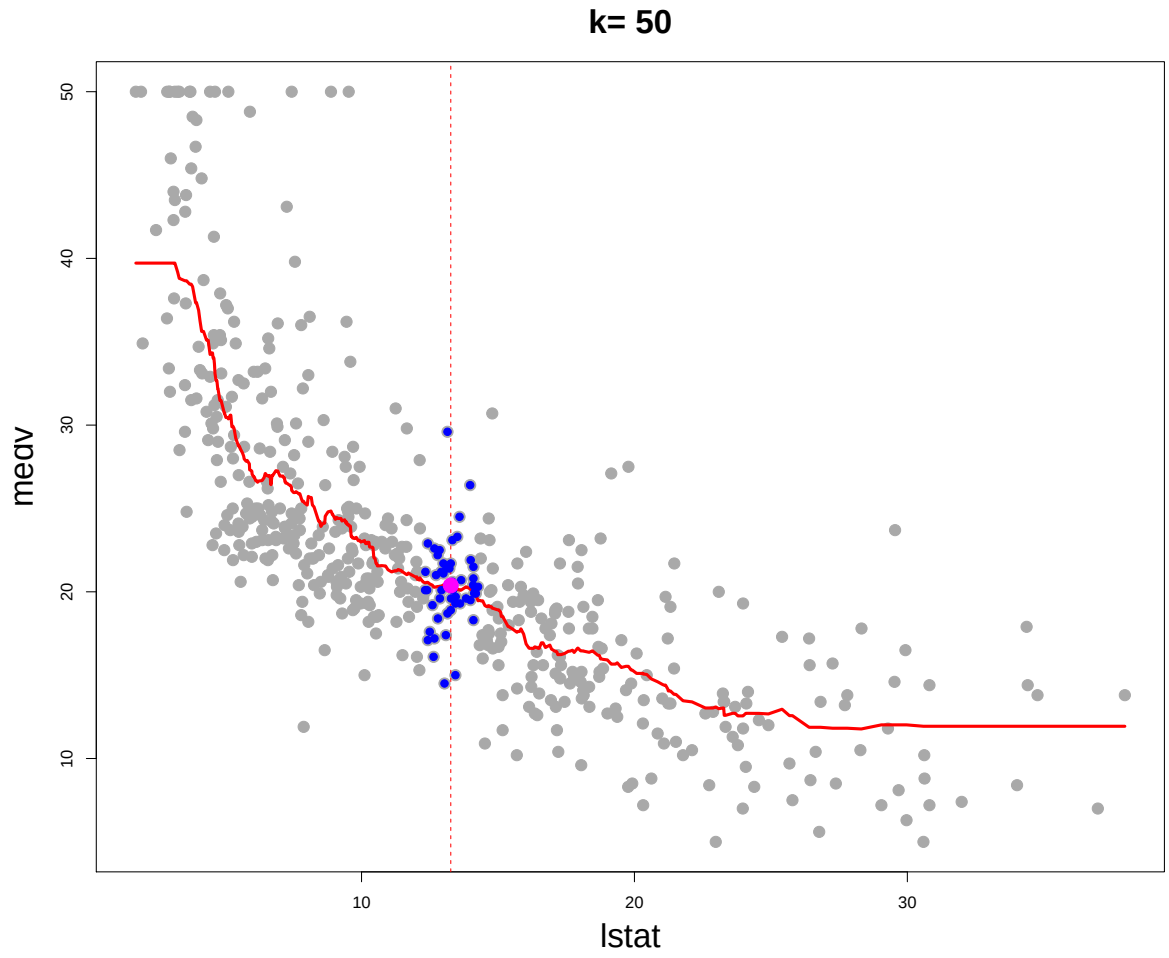
What do I mean by closest?

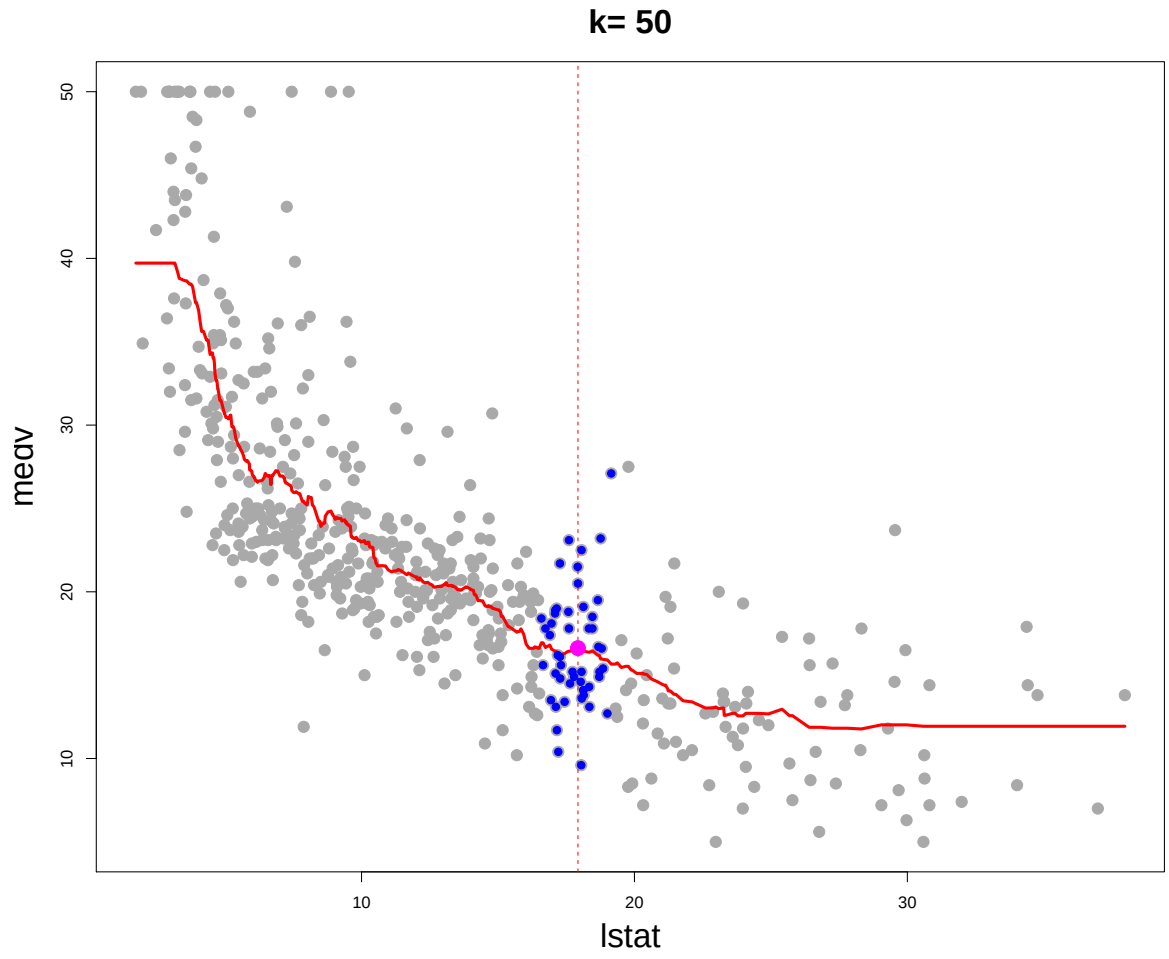
We will choose the k=50 points that are closest to the X value at which we are trying to predict.

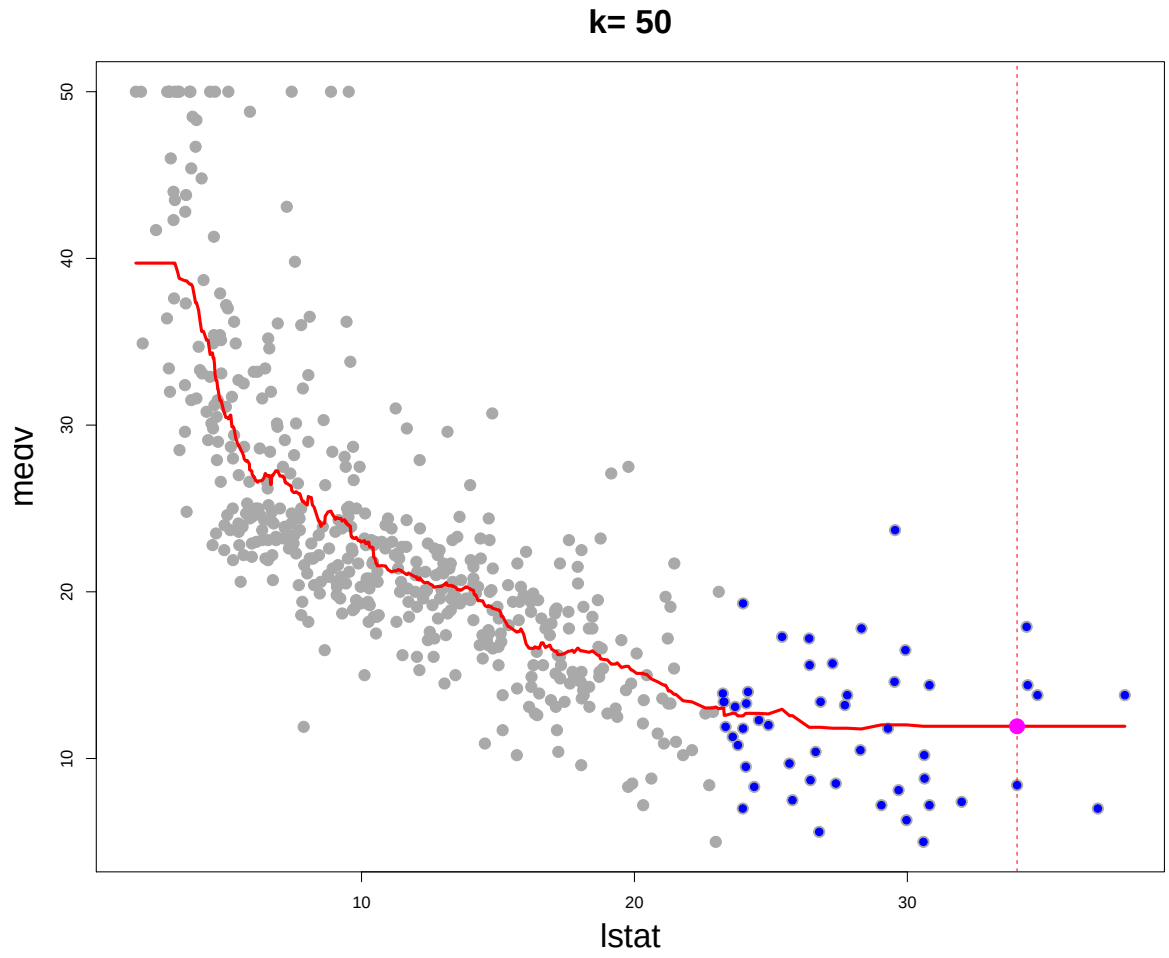

k= 50
medv
lstat
10
20
30
40
50
10
20
30


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

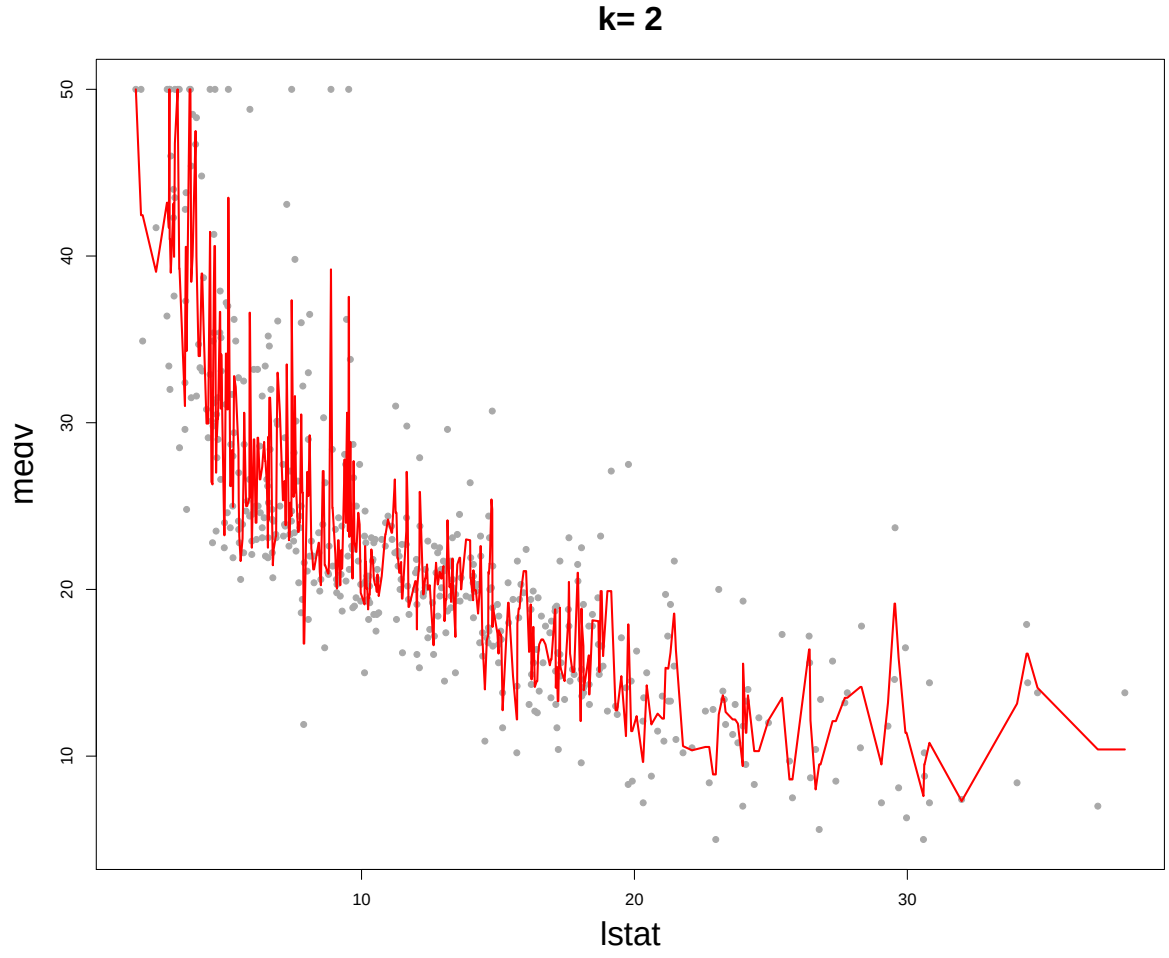

k= 50
medv
lstat
10
20
30
40
50
10
20
30

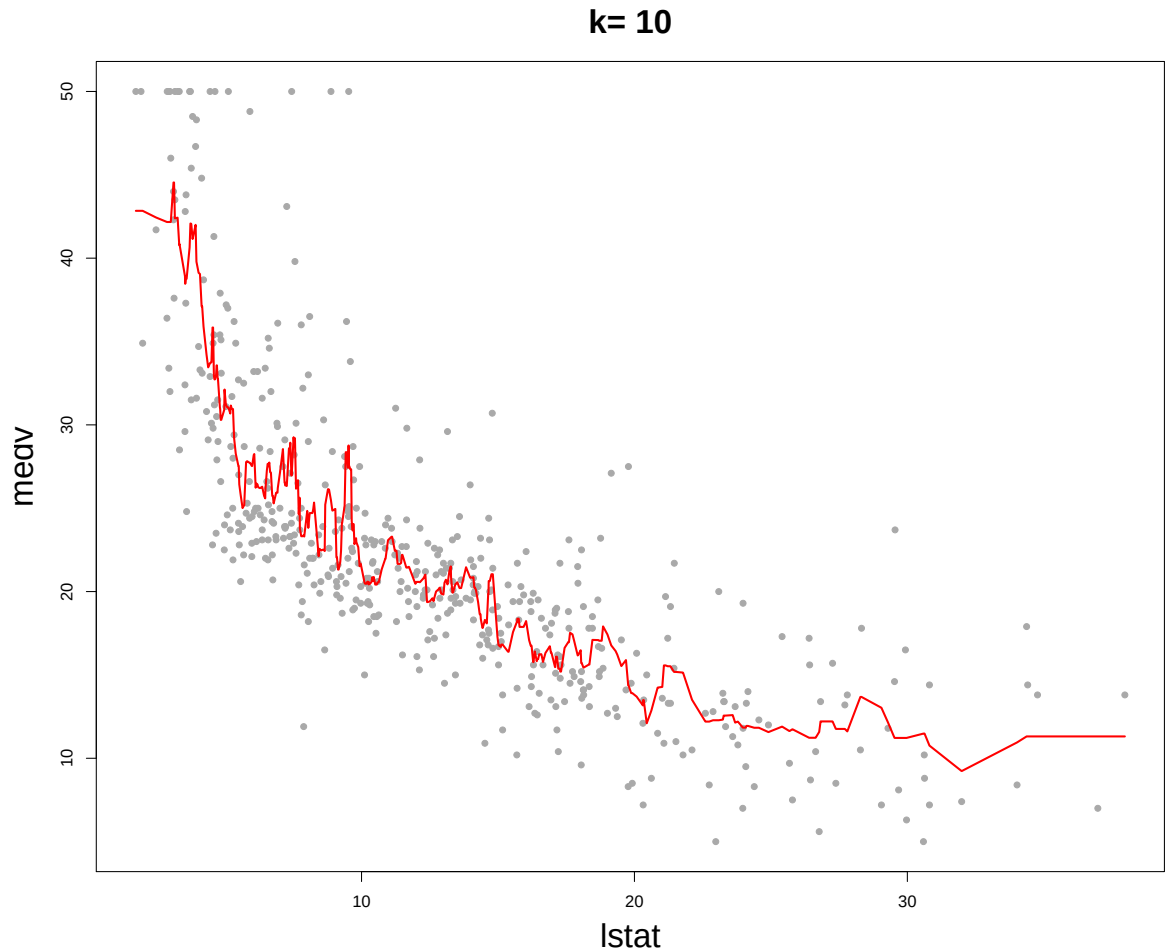

k= 50
medv
lstat
10
20
30
40
50
10
20
30

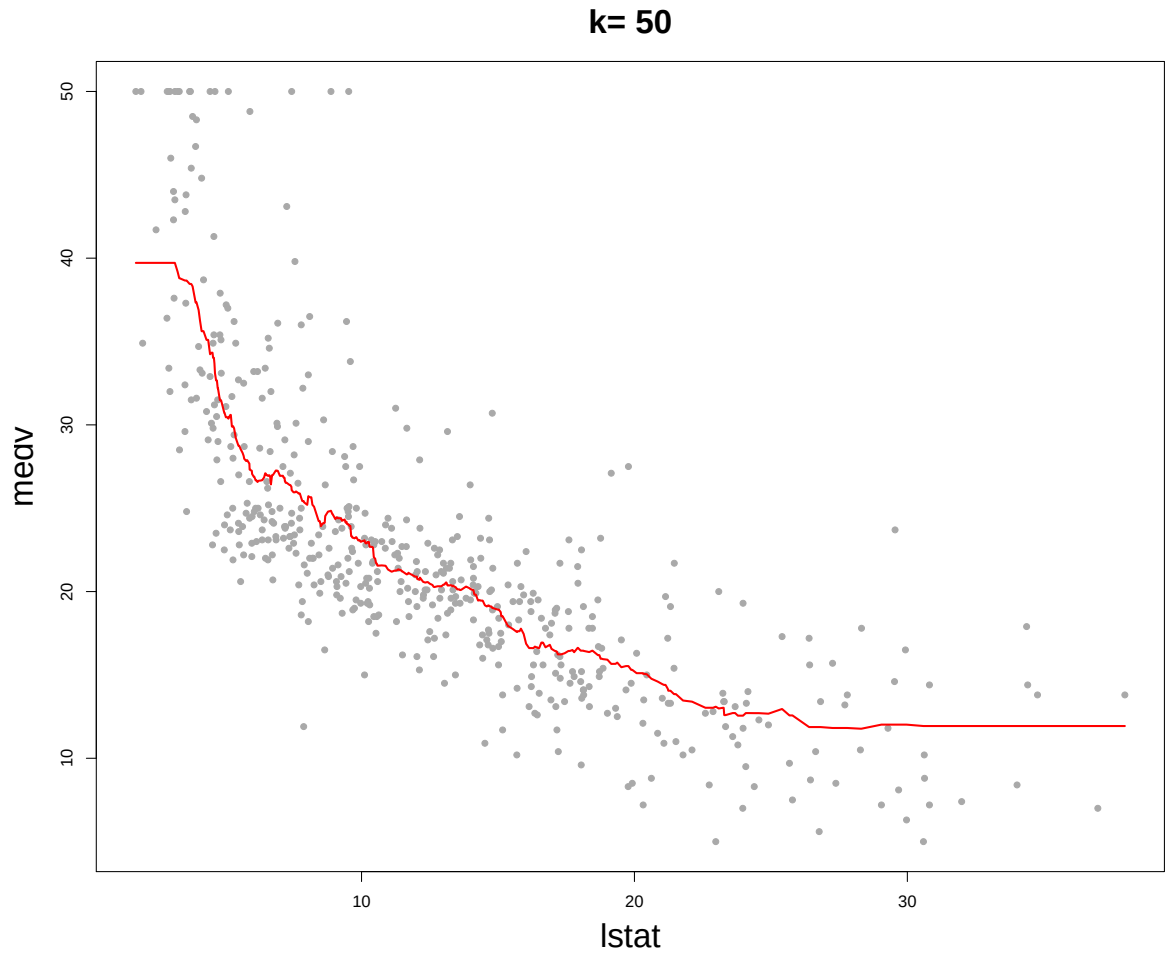

k= 50
medv
lstat
10
20
30
10
20
30
40
50

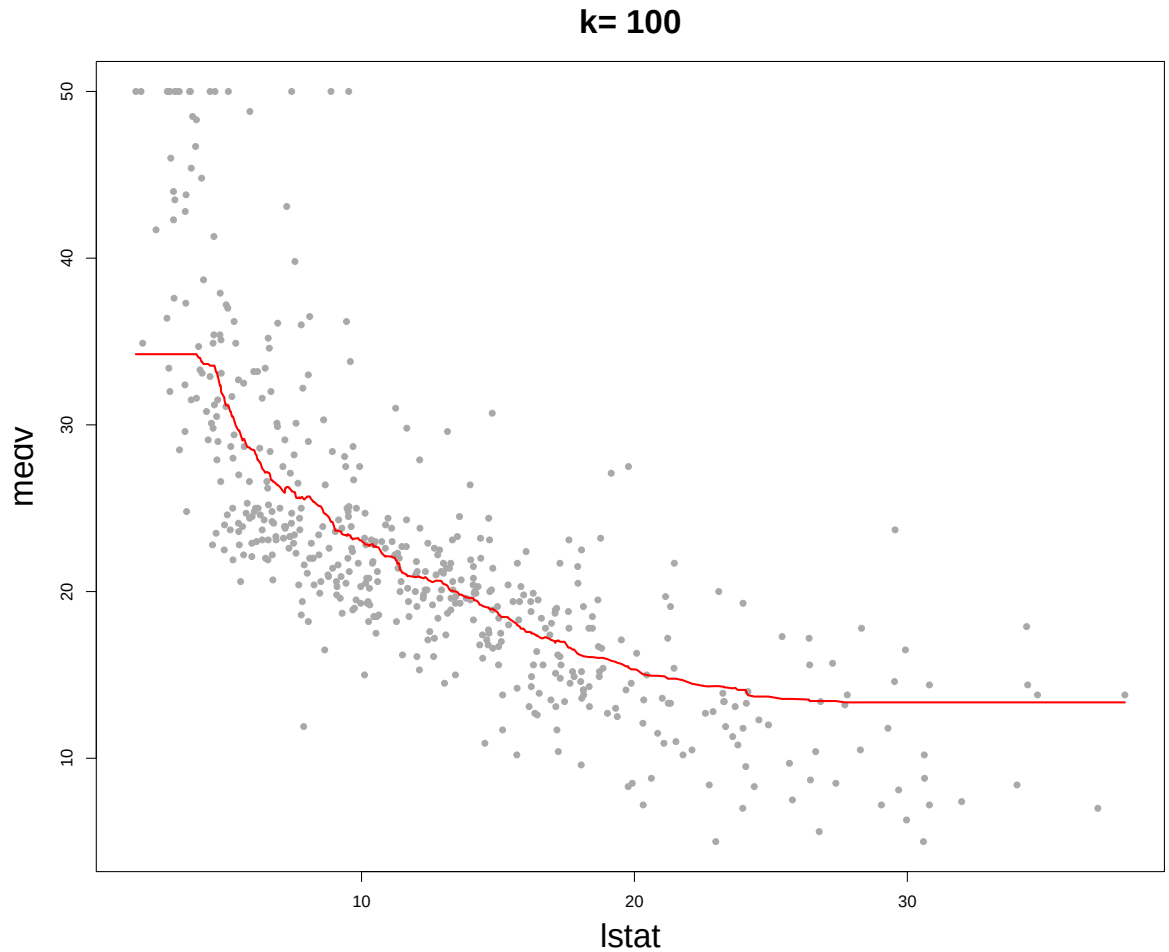

k= 50
medv
lstat
10
20
30
40
50
10
20
30

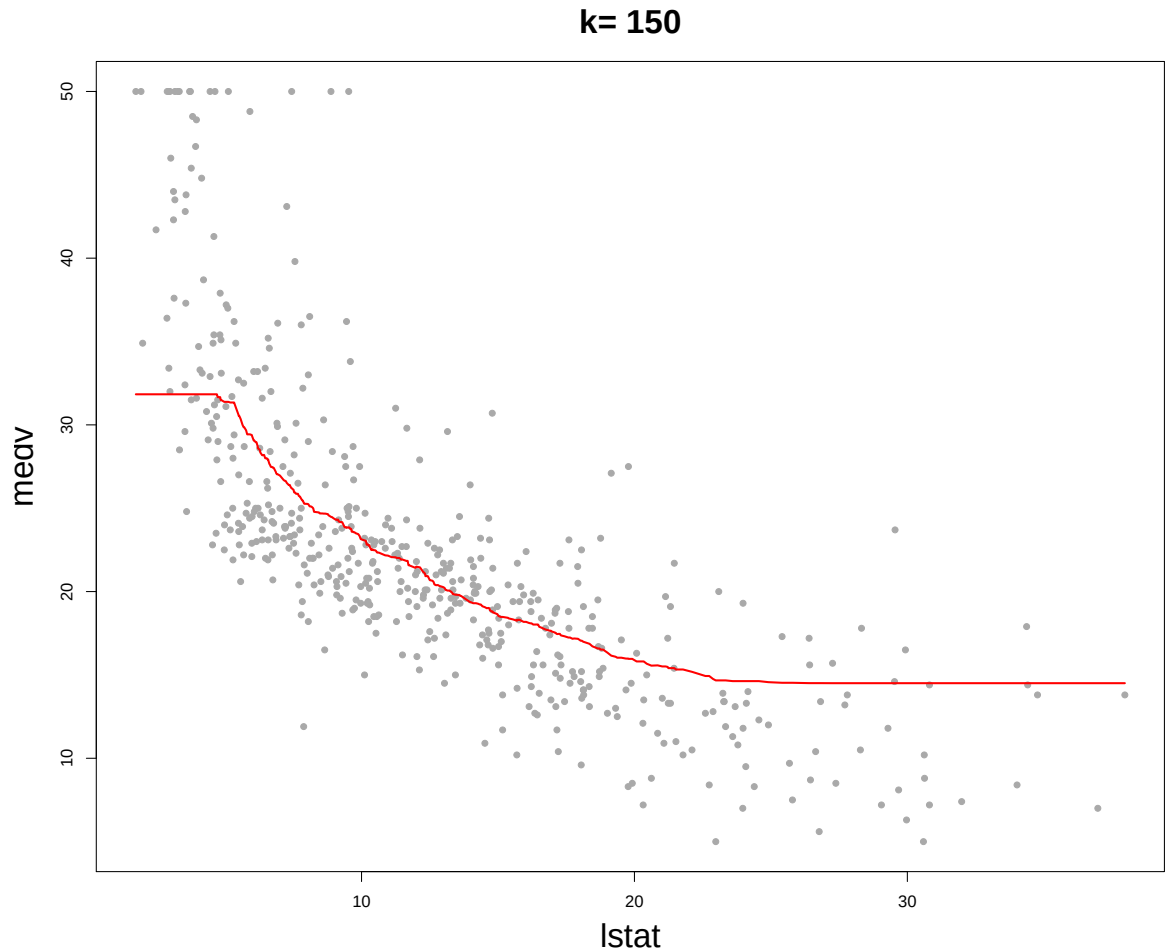
Okay, that seems sensible, but, 2 neighbors or 200 neighbors?

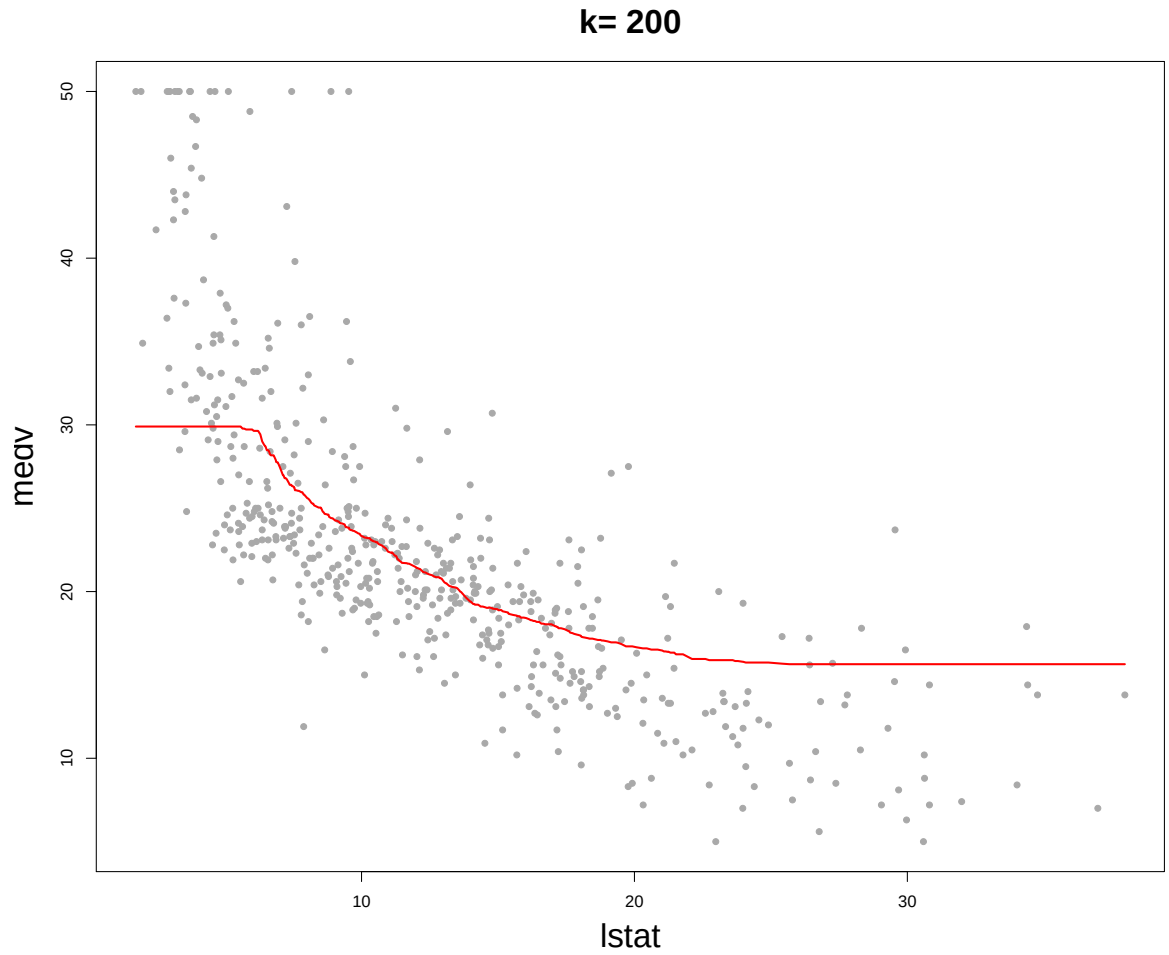


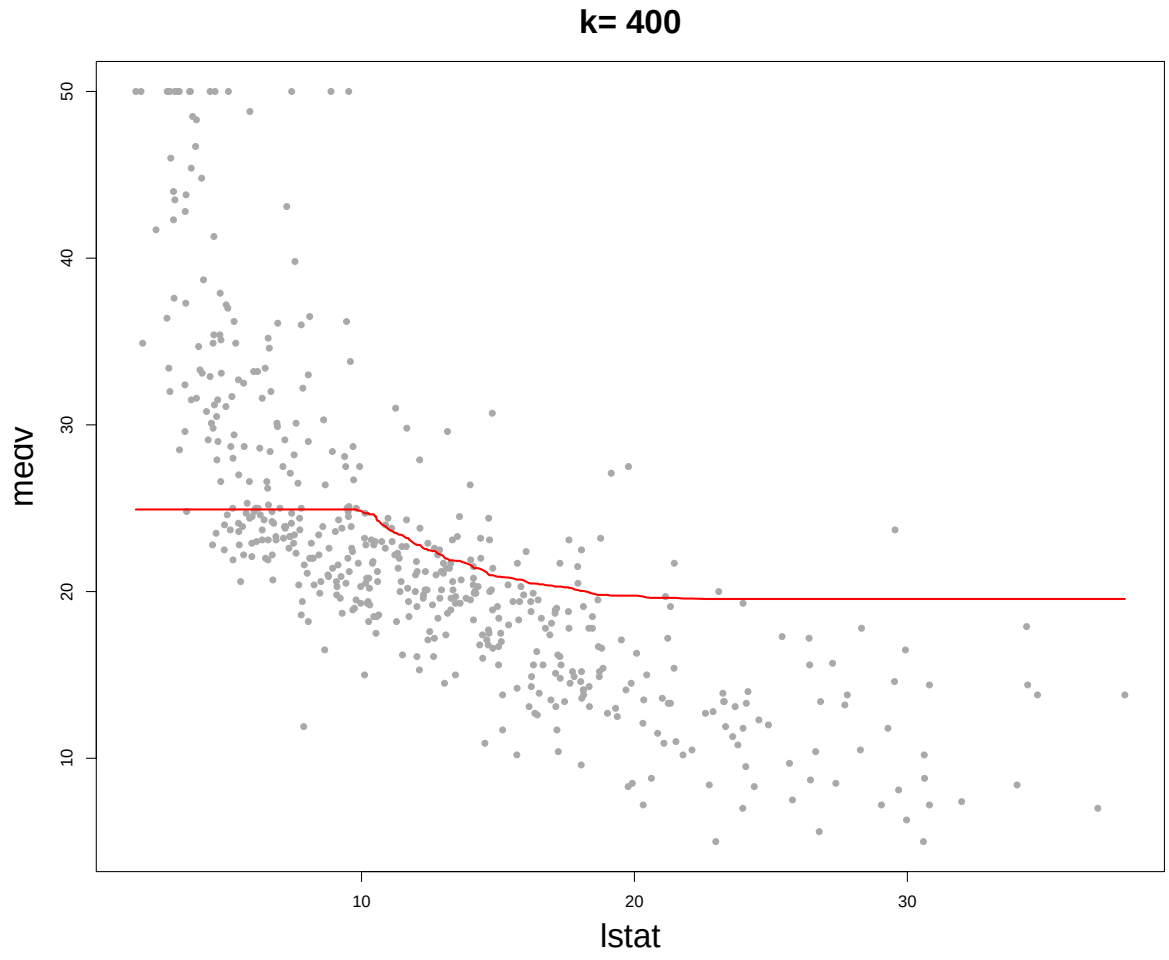

k= 10
medv
lstat
10
20
30
40
50
10
20
30

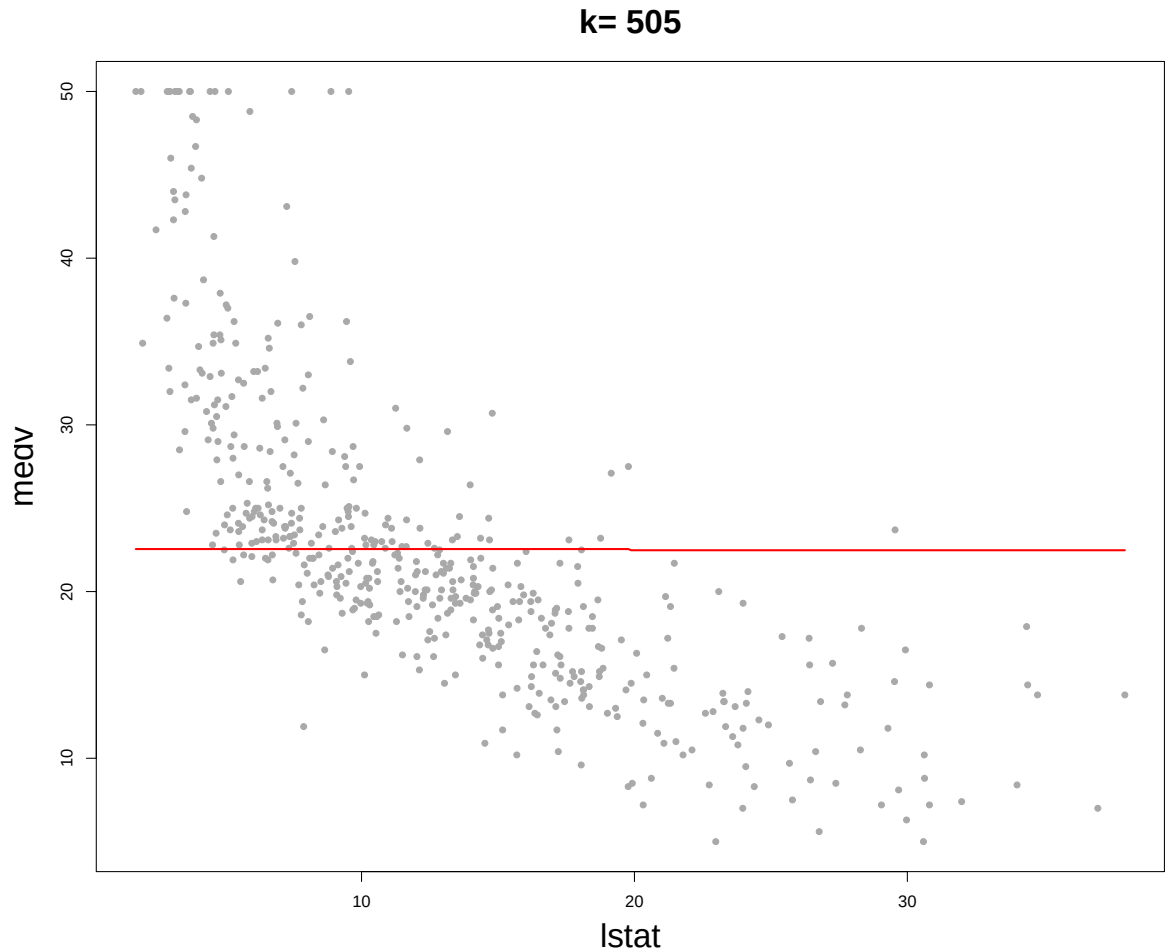

k= 50
medv
lstat
10
20
30
40
50
10
20
30


k= 100
medv
lstat
10
20
30
40
50
10
20
30


k= 150
medv
lstat
10
20
30
40
50
10
20
30


k= 200
medv
lstat
10
20
30
40
50
10
20
30


k= 400
medv
lstat
10
20
30
40
50
10
20
30


k= 505
medv
lstat
10
20
30
40
50
10
20
30

for k-NN:

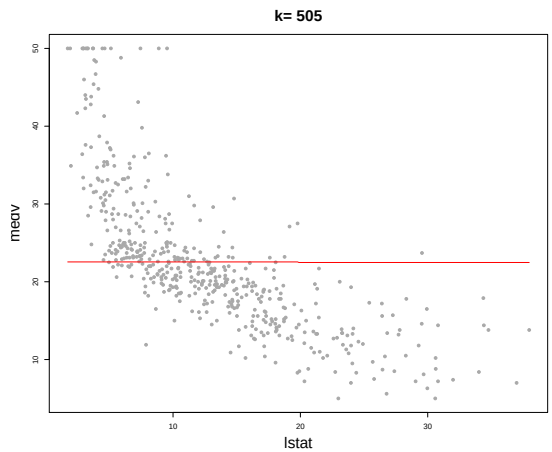
A big k gives us a simple looking function.

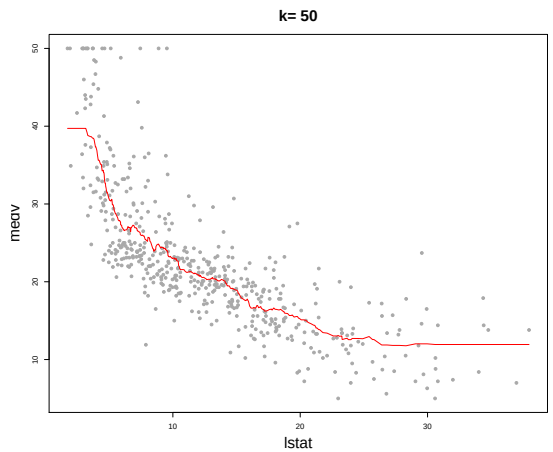
A small k can give us a more *complex, flexible* looking function.

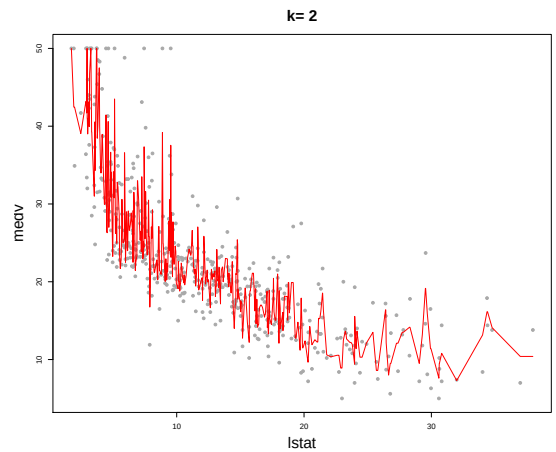
Complexity, Generalization and Interpretation

- 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!







In R:

```
#load knn library (need to have installed this with install.packages("kknn"))
library(kknn)

#get Boston data
library(MASS) ## a library of example datasets
#try:
names(Boston)
dim(Boston)
Boston[1:5,]
summary(Boston)
ls()
mean(Boston$lstat)
mean(Boston[,13])

#make the variables in Boston directly accessible
attach(Boston)
ls(pos=1)
ls(pos=2)

#plot the data
plot(lstat,medv,xlab="% lower status",ylab="median value")
```

```
#run regression, print summary, add line to plot
lmB = lm(medv~lstat,Boston)
print(summary(lmB))
abline(lmB$coef,col="red",lwd=4) #lwd: line width=4
#try:
names(lmB)
cor(lmB$fitted.values,lmB$residuals) #cor is 0!


#fit knn with k=50
train = data.frame(lstat,medv) #data frame with variables of interest
#test is data frame with x you want f(x) at, sort lstat to make plots nice.
test = data.frame(lstat = sort(lstat))
kf50 = kknn(medv~lstat,train,test,k=50,kernel = "rectangular")

#add knn50 fit to plot
lines(test$lstat,kf50$fitted.values,col="blue",lwd=2)

#add k=200
kf200 = kknn(medv~lstat,train,test,k=200,kernel = "rectangular")
lines(test$lstat,kf200$fitted.values,col="magenta",lwd=2,lty=2) #line type 2

#add legend to plot
legend("topright",legend=c("lin","knn50","knn200"),
          col=c("red","blue","magenta"),lty=c(1,1,2))
```

```
#get price prediction at lstat = 30 using k=50
dfp = data.frame(lstat=30)
k50 = kknn(medv~lstat,train,dfp,k=50,kernel = "rectangular")
cat("kNN50: predicted house price at lstat=30 is ",k50$fitted,"\n")
points(30,k50$fitted,pch=4,cex=2,col="black")

#get prediction from linear fit
p30L = predict(lmB,dfp)
cat("Linear: predicted house price at lstat=30 is ",p30L,"\n")
text(30,p30L,"L",cex=2,col="black")
```

2. Measuring Accuracy

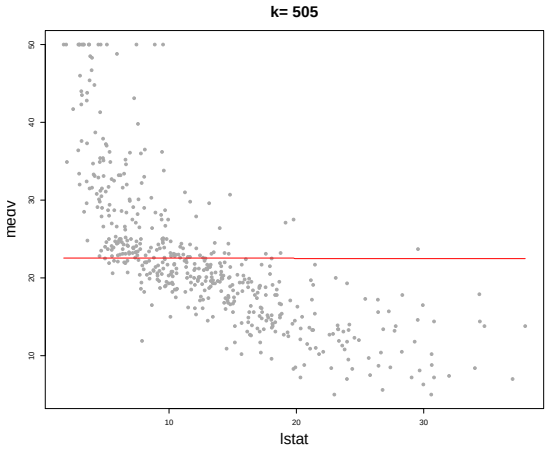
How accurate are each of these models?

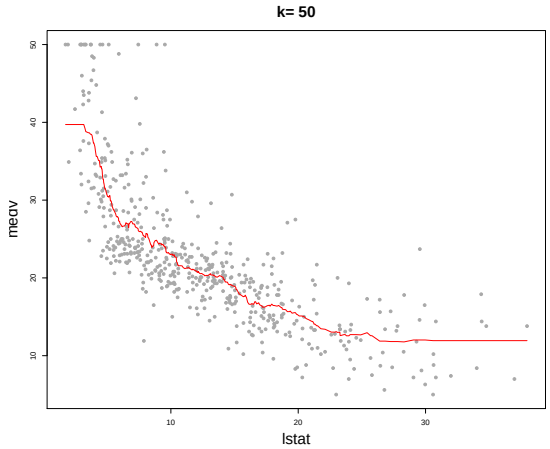
We can measure the *fit* of our model (our fuction estimate) using the root mean squared error (RMSE):

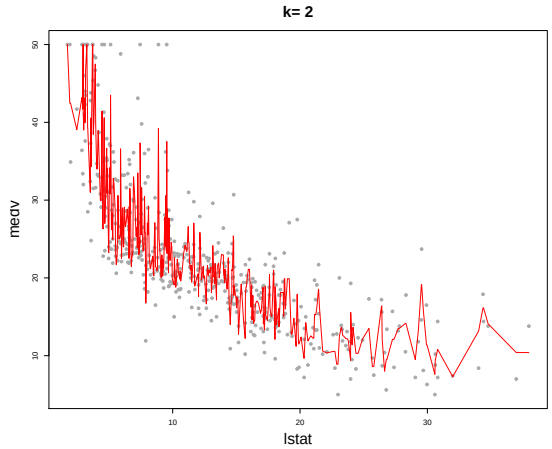
$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left[Y_i - \hat{f}(X_i)\right]^2}$$

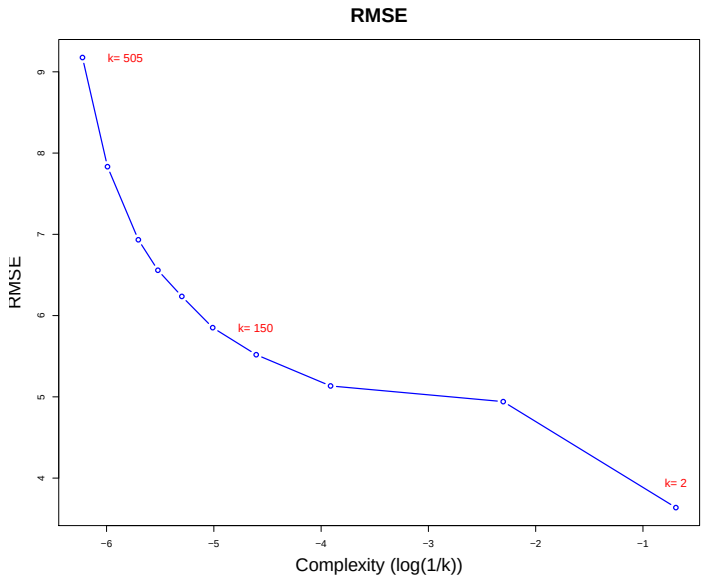
This measures, on average, how large the "mistakes" (errors) made by the model are...

Measuring Accuracy (Boston housing, again)









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

3. 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* or *validation set*)

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

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

NB:

(1)

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

Now we have $\hat{f}$.

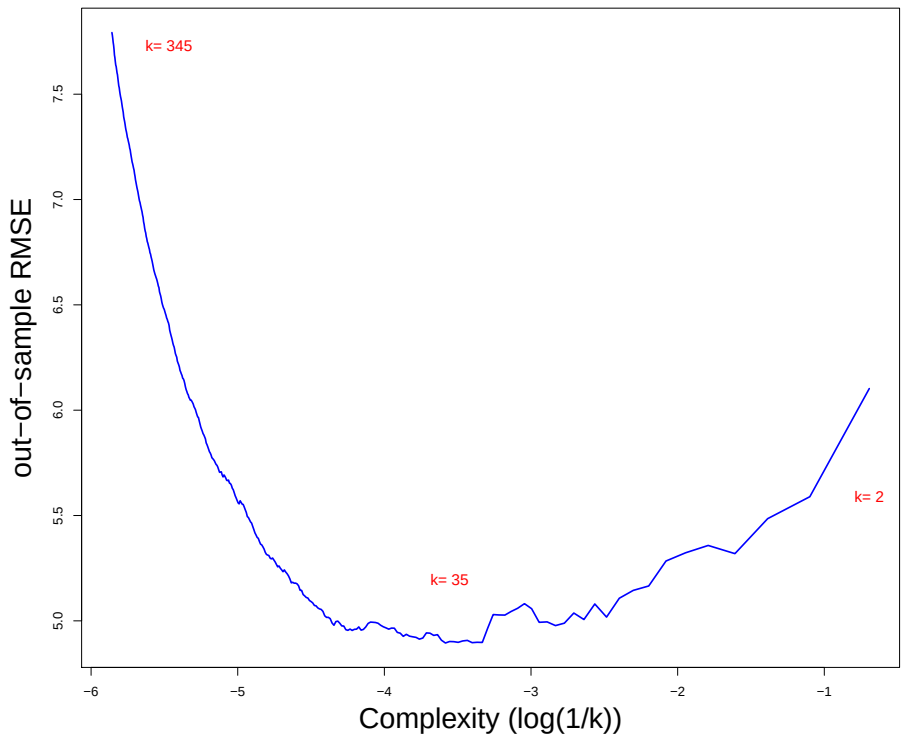
(2)

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

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

Out-of-Sample Predictions

In our Cars 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)...

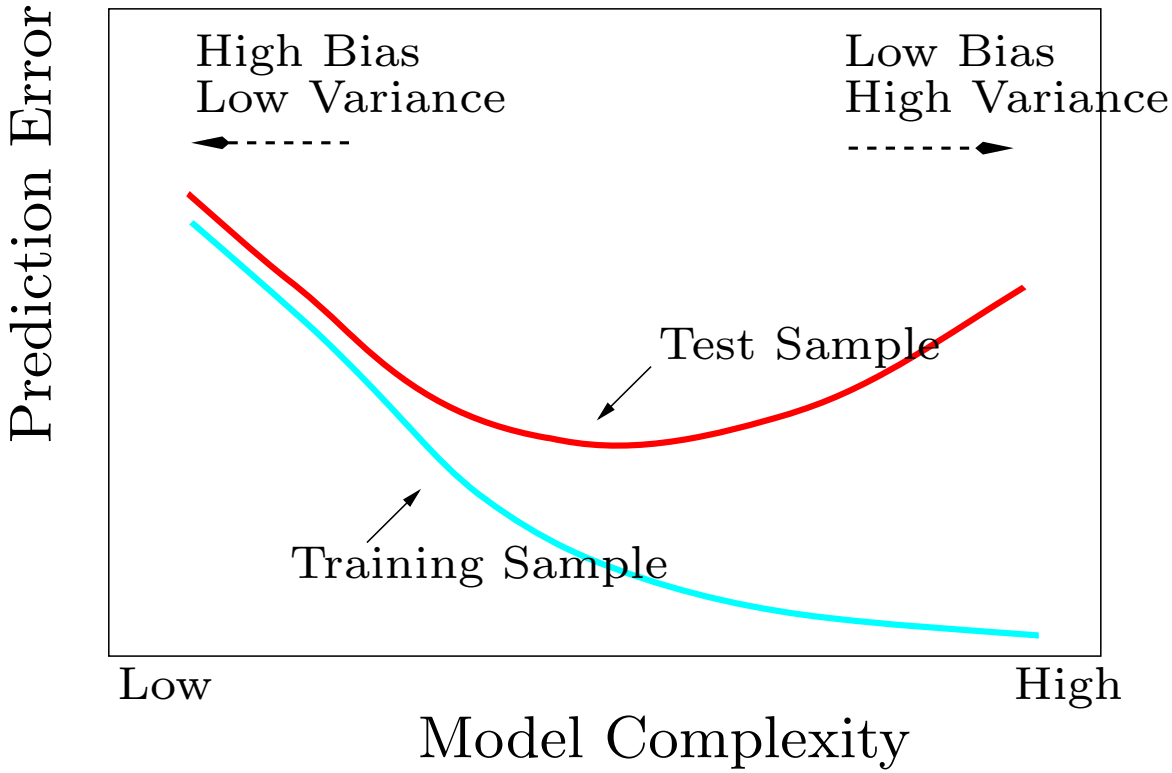


Now, the model where $k = 35$ looks like the most accurate choice!!

$\log(1/35) =$ -3.56

Not too simple but not too complex!!!

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.

In R:

```
#load libraries and get Boston data
library(kknn) ## knn library
library(MASS) ## a library of example datasets
attach(Boston)
n = nrow(Boston)

# get in-sample and out-of-sample data frames
df = data.frame(lstat,medv) #simple data frame for conveniance
ntrain=400 #number of observations for training data
set.seed(99) #set seed for random sampling of training data
tr = sample(1:nrow(df),ntrain)
train = df[tr,] #training data
test = df[-tr,] #test data

#loop over values of k, fit on train, predict on test
kvec=2:350; nk=length(kvec)
outMSE = rep(0,nk) #will will put the out-of-sample MSE here
for(i in 1:nk) {
   near = kknn(medv~lstat,train,test,k=kvec[i],kernel = "rectangular")
   MSE = mean((test[,2]-near$fitted)^2)
   outMSE[i] = MSE
}
```

```
#plot
par(mfrow=c(1,2))
plot(kvec,sqrt(outMSE))
plot(log(1/kvec),sqrt(outMSE))
imin = which.min(outMSE)
cat("best k is ",kvec[imin],"\n")

#fit with all data and best k and plot
test = data.frame(lstat=sort(df$lstat))
near = kknn(medv~lstat,df,test,k=kvec[imin],kernel = "rectangular")
par(mfrow=c(1,1))
plot(df)
lines(test$lstat,near$fitted,col="red",type="b")
```

4. Bias-Variance Trade-Off

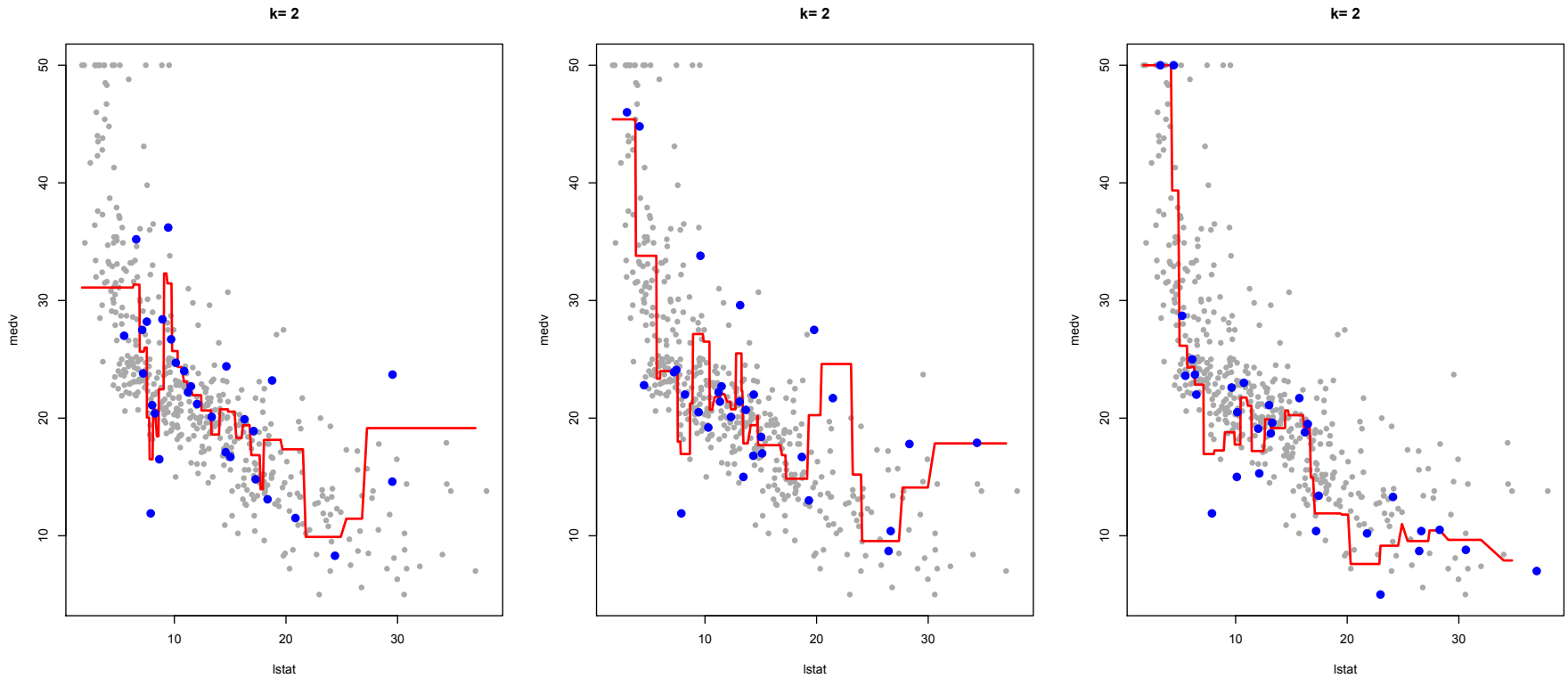
Why do complex models behave poorly in making predictions?

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

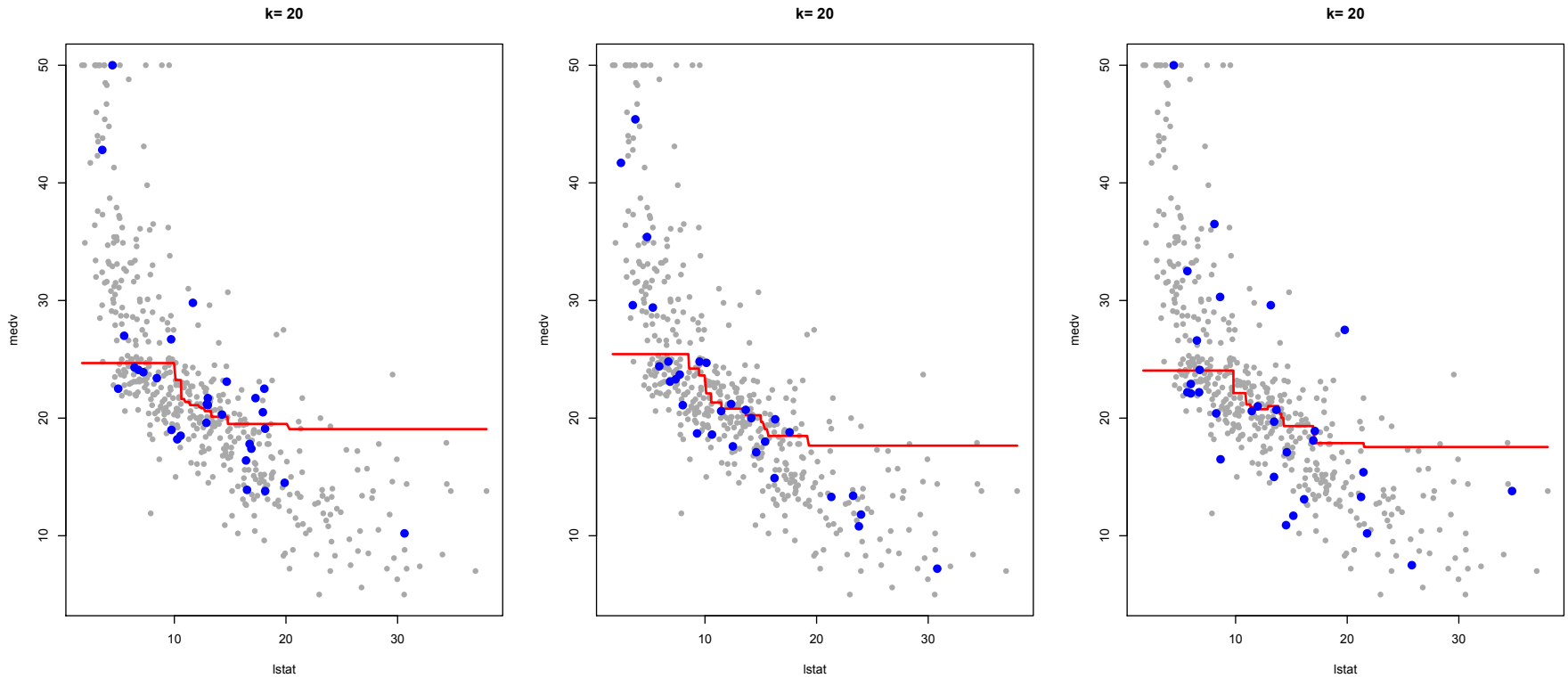
High variability...



(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* $= 20$...

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

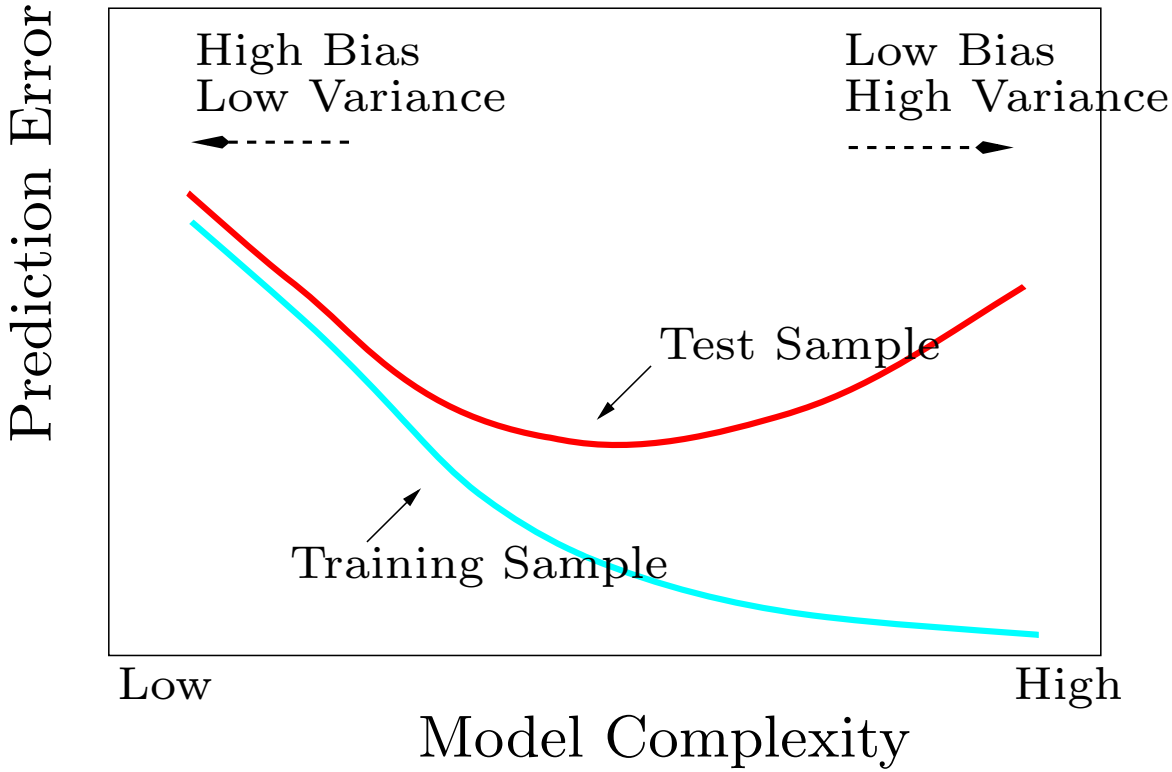
what do you see?

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...

$$Y_f = f(X_f) + \epsilon$$

- 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

5. 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 helps address both of these issues.

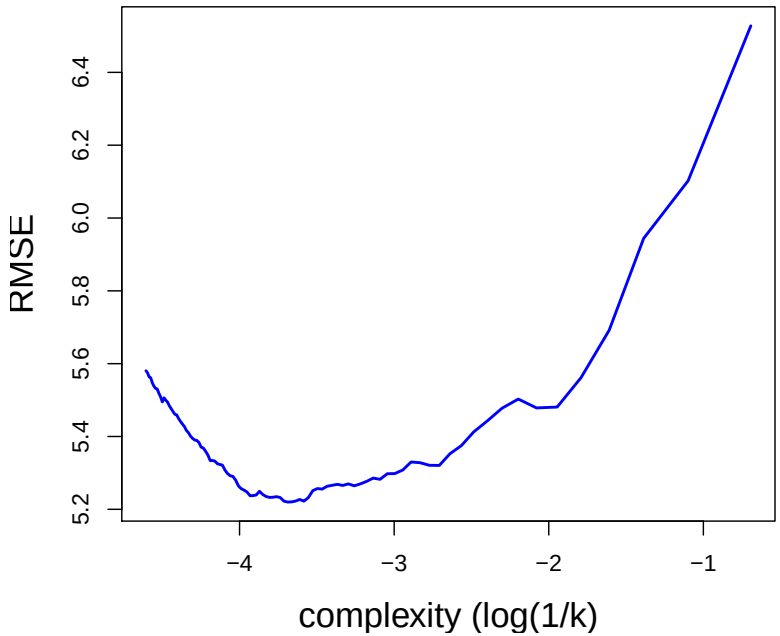
Leave-One-Out Cross-Validation (loocv)

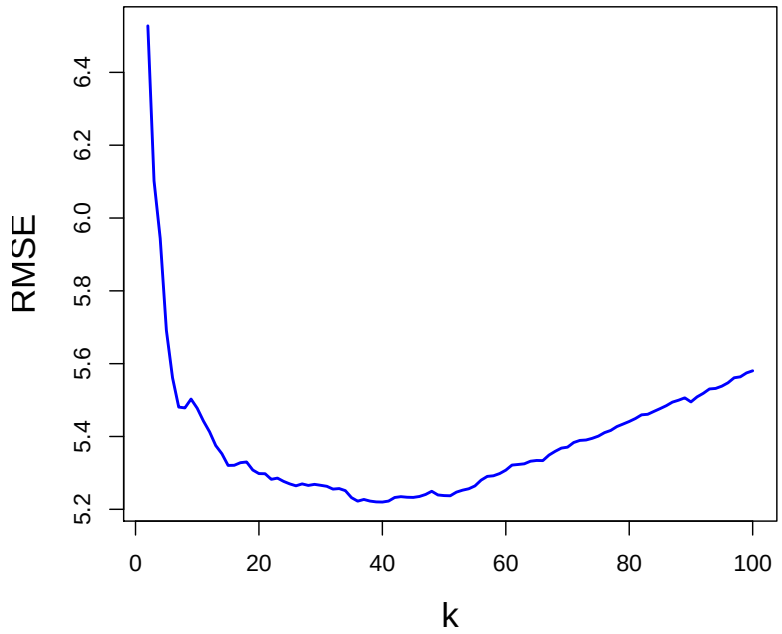
The name says it all!

- Assume we have n observations in our dataset. Define the validation set by choosing only one observation. Call it the i^{th} observation...
- The model is then trained in the remaining $n-1$ observations and the results are used to predict the left-out observation. Compute the squared-error $MSE_i = (Y_i - \hat{Y}_i)^2$
- Repeat the procedure for every observation in the dataset (n times) and compute the average cross-validation MSE:

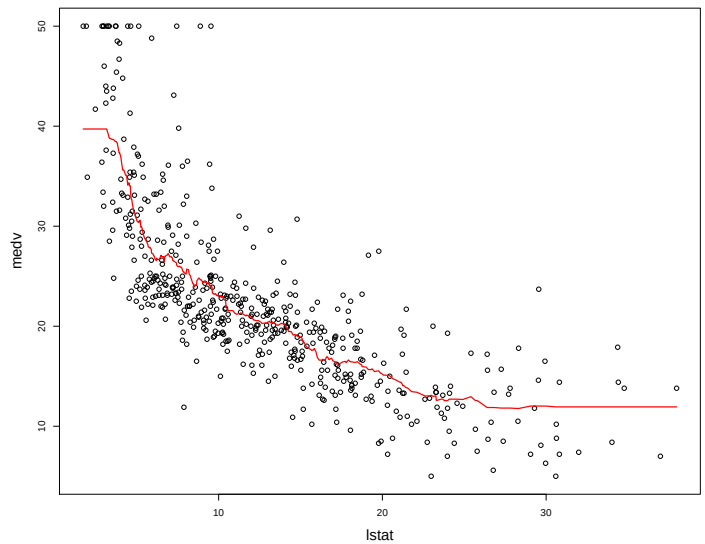
$$MSE^{loocv} = \frac{1}{n}\sum_{i=1}^{n} MSE_i$$

loocv: Boston Data: x=lstat, y=medv





Min is at about
$k = 40$.



k-fold Cross Validation

LOOCV can be computationally expensive as each model being considered has to be estimated n times! A popular alternative is what is called k-fold Cross Validation.

- This approach randomly divides the original dataset into k groups of approximately the same size
- Choose one of the groups as a validation set. Estimate the models with the remaining $k-1$ groups and predict the samples in the validation set. This will give us $\hat{Y}_i$ for each i in the validation set (fold you are predicting).
- Repeat the procedure for every *fold* in the dataset (k times). This will give us a (*out-of-sample*) $\hat{Y}_i$ for every observation in the data set.
- $MSE^{kcv} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$

k-fold Cross Validation

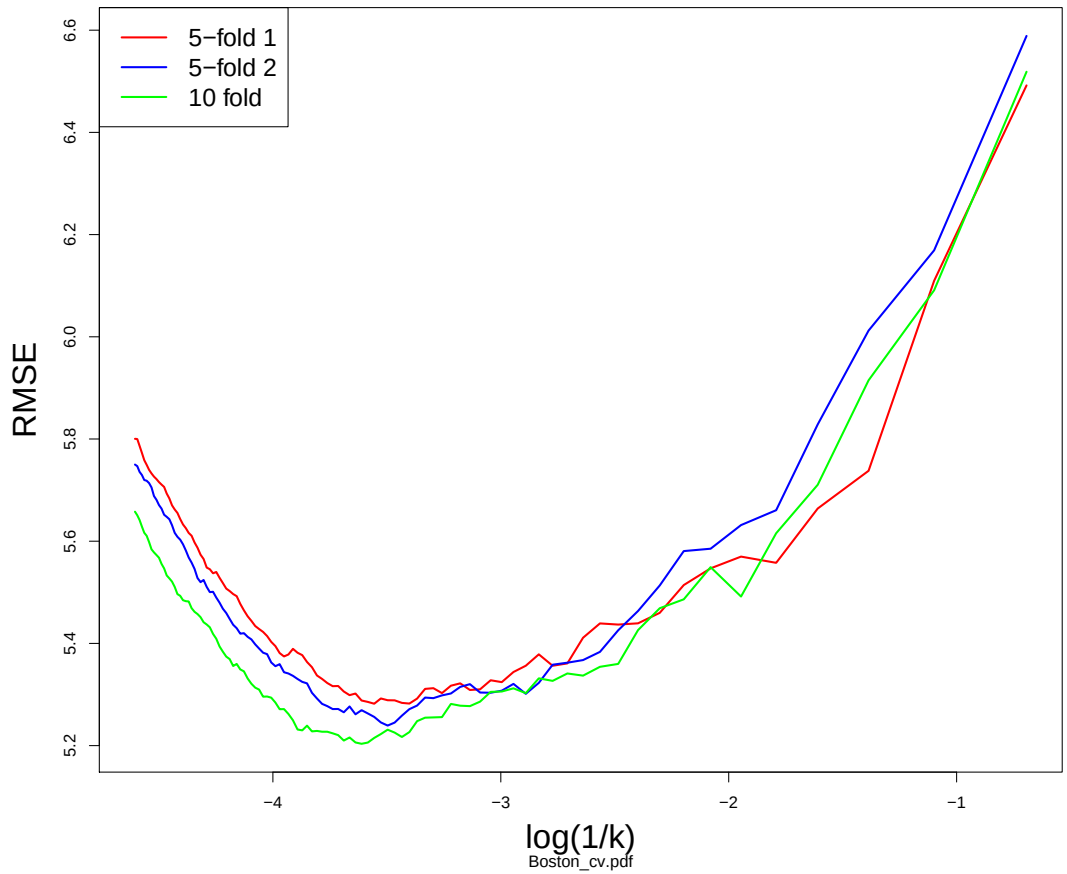
The usual choices are $k = 5$ and $k = 10$...

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 !!!

In R:

```
#load libraries and docv.R
library(MASS)
library(kknn)
source("docv.R") #this has docvknn used below

#make variable names in Boston directly available
attach(Boston)

#do k-fold cross validation, 5 twice, 10 once
set.seed(99) #always set the seed!  (Gretzky was number 99)
kv = 2:100 #these are the k values (k as in kNN) we will try

#docvknn(matrix x, vector y,vector of k values, number of folds),
#does cross-validation for training data (x,y).
cv1 = docvknn(matrix(lstat,ncol=1),medv,kv,nfold=5)
cv2 = docvknn(matrix(lstat,ncol=1),medv,kv,nfold=5)
cv3 = docvknn(matrix(lstat,ncol=1),medv,kv,nfold=10)

#docvknn returns error sum of squares, want RMSE
cv1 = sqrt(cv1/length(medv))
cv2 = sqrt(cv2/length(medv))
cv3 = sqrt(cv3/length(medv))
```

```
#plot
rgy = range(c(cv1,cv2,cv3))
plot(log(1/kv),cv1,type="l",col="red",ylim=rgy,lwd=2,cex.lab=2.0,
                        xlab="log(1/k)", ylab="RMSE")
lines(log(1/kv),cv2,col="blue",lwd=2)
lines(log(1/kv),cv3,col="green",lwd=2)
legend("topleft",legend=c("5-fold 1","5-fold 2","10 fold"),
                           col=c("red","blue","green"),lwd=2,cex=1.5)

#get the min
cv = (cv1+cv2+cv3)/3 #use average
kbest = kv[which.min(cv)]
cat("the best k is: ",kbest,"\n")

#fit kNN with best k and plot the fit.
kfbest = kknn(medv~lstat,data.frame(lstat,medv),data.frame(lstat=sort(lstat)),
                     k=kbest,kernel = "rectangular")
plot(lstat,medv,cex.lab=1.2)
lines(sort(lstat),kfbest$fitted,col="red",lwd=2,cex.lab=2)
```

6. 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:

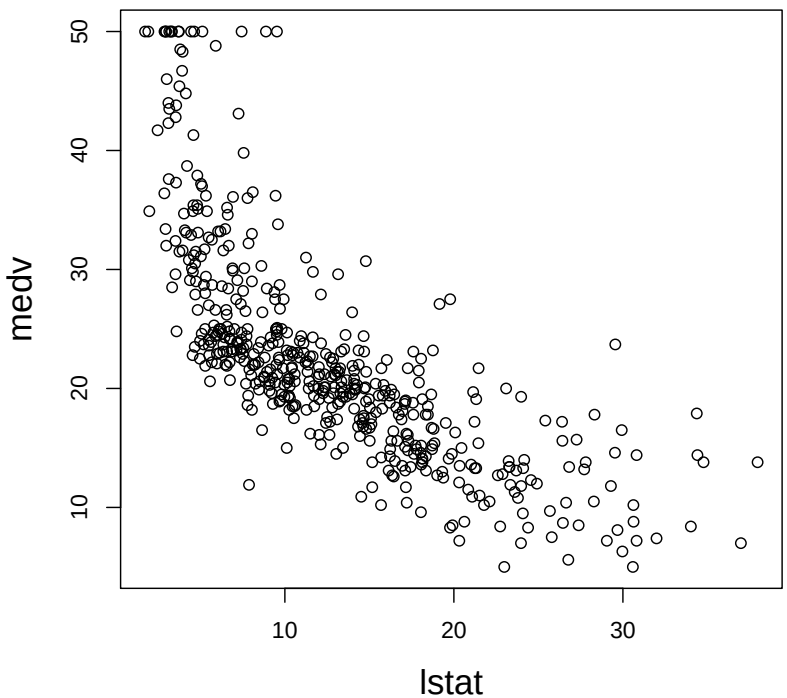
- The distance metric used above is only valid for numerical values of X. When $X's$ are categorical we need to think about a different distance metric or perform some manipulation of the information.
- The scale of X also will have an impact. In general it is a good idea put the $X's$ in the same scale before running kNN.

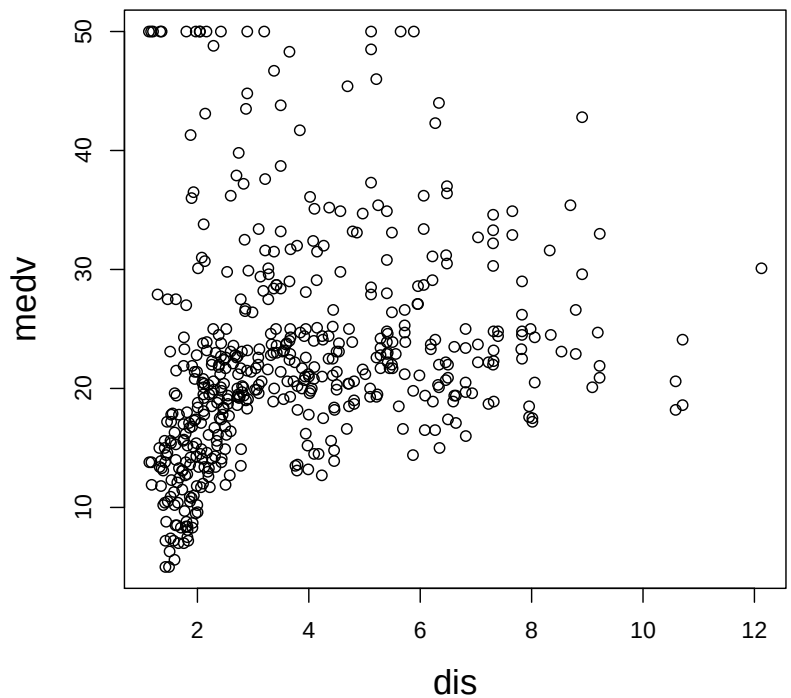
What do we mean by *scale?*

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 $p = 2$:

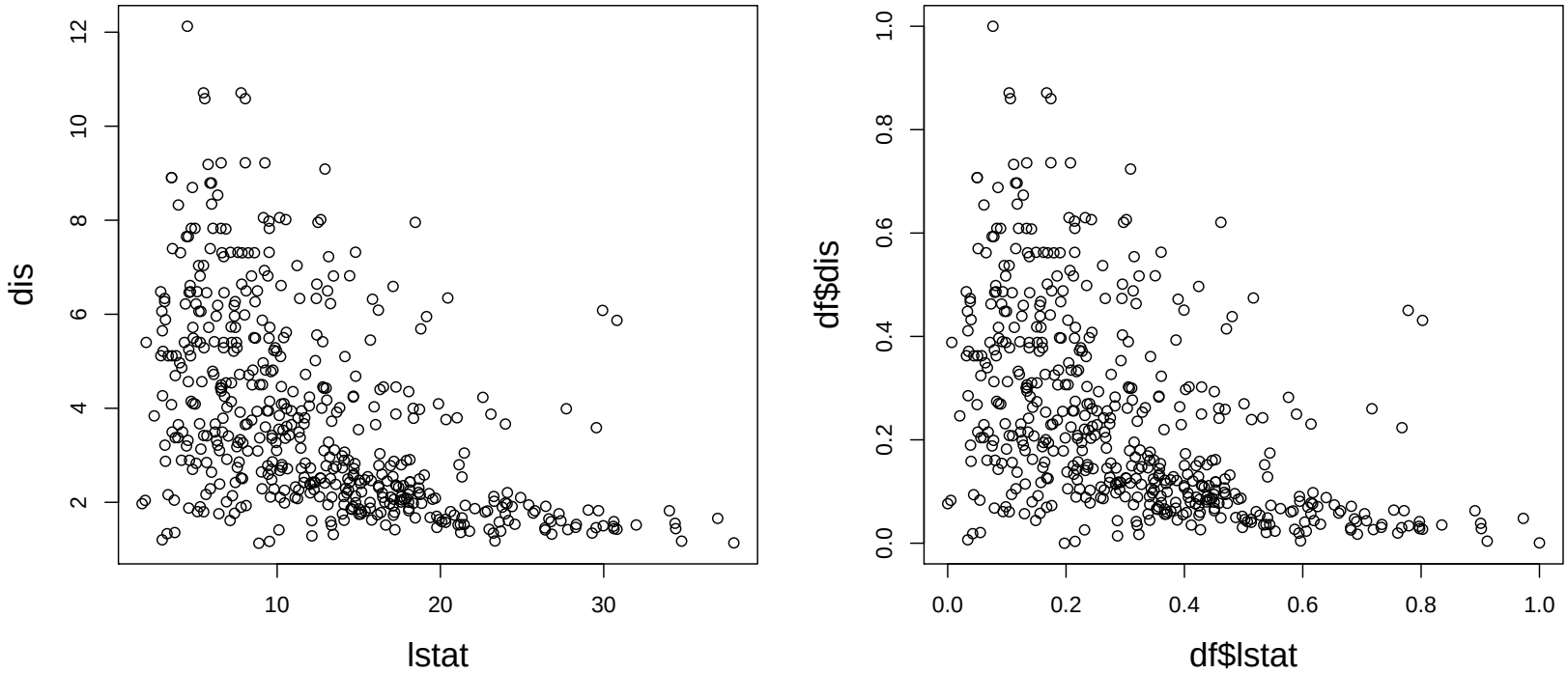
x_1=lstat
x_2=dis: weighted mean of distances to five Boston employment centres





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

Left: x_1 vs. x_2.



Right:

We rescale each x

$$x \Rightarrow \frac{x - \min(x)}{(\max(x) - \min(x))}$$

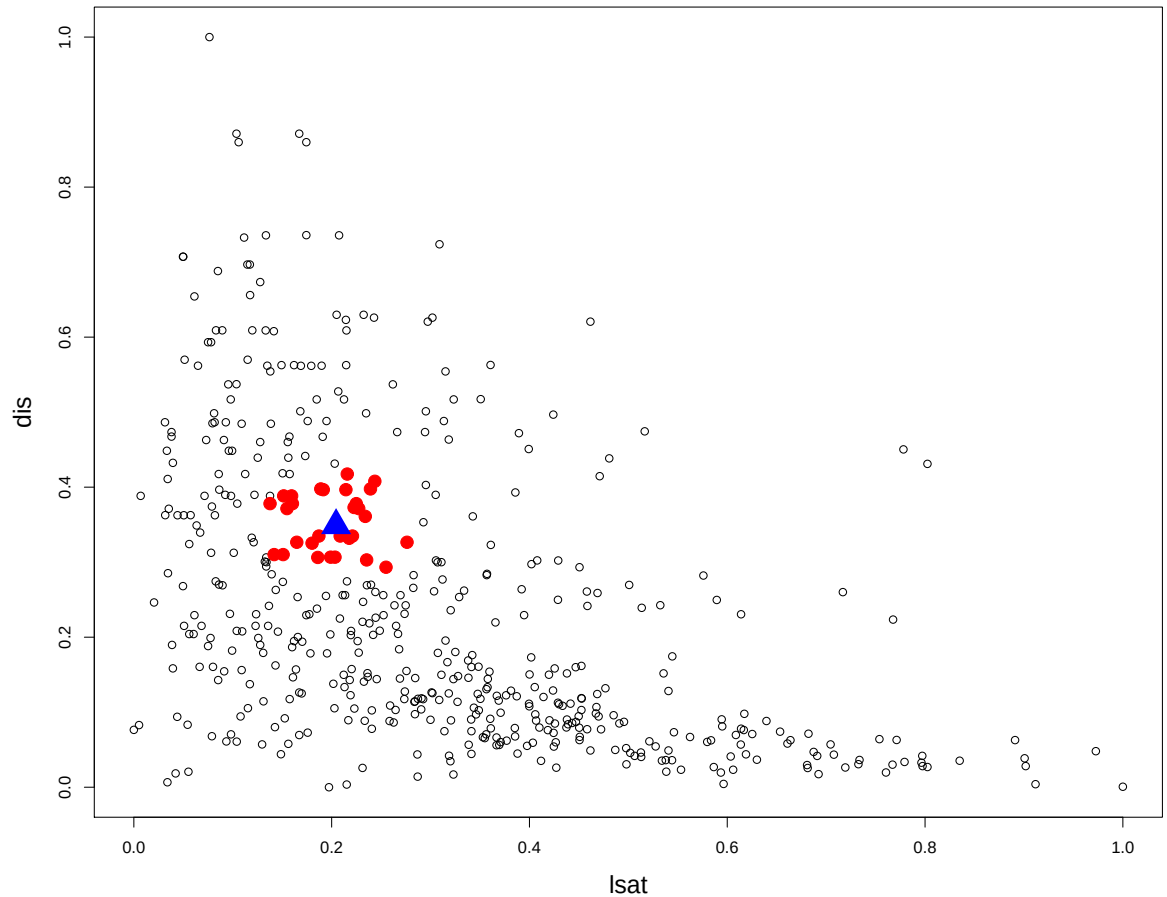
$$x \Rightarrow \frac{x - \min(x)}{(\max(x) - \min(x))}$$

With this scaling, x is always between 0 and 1.

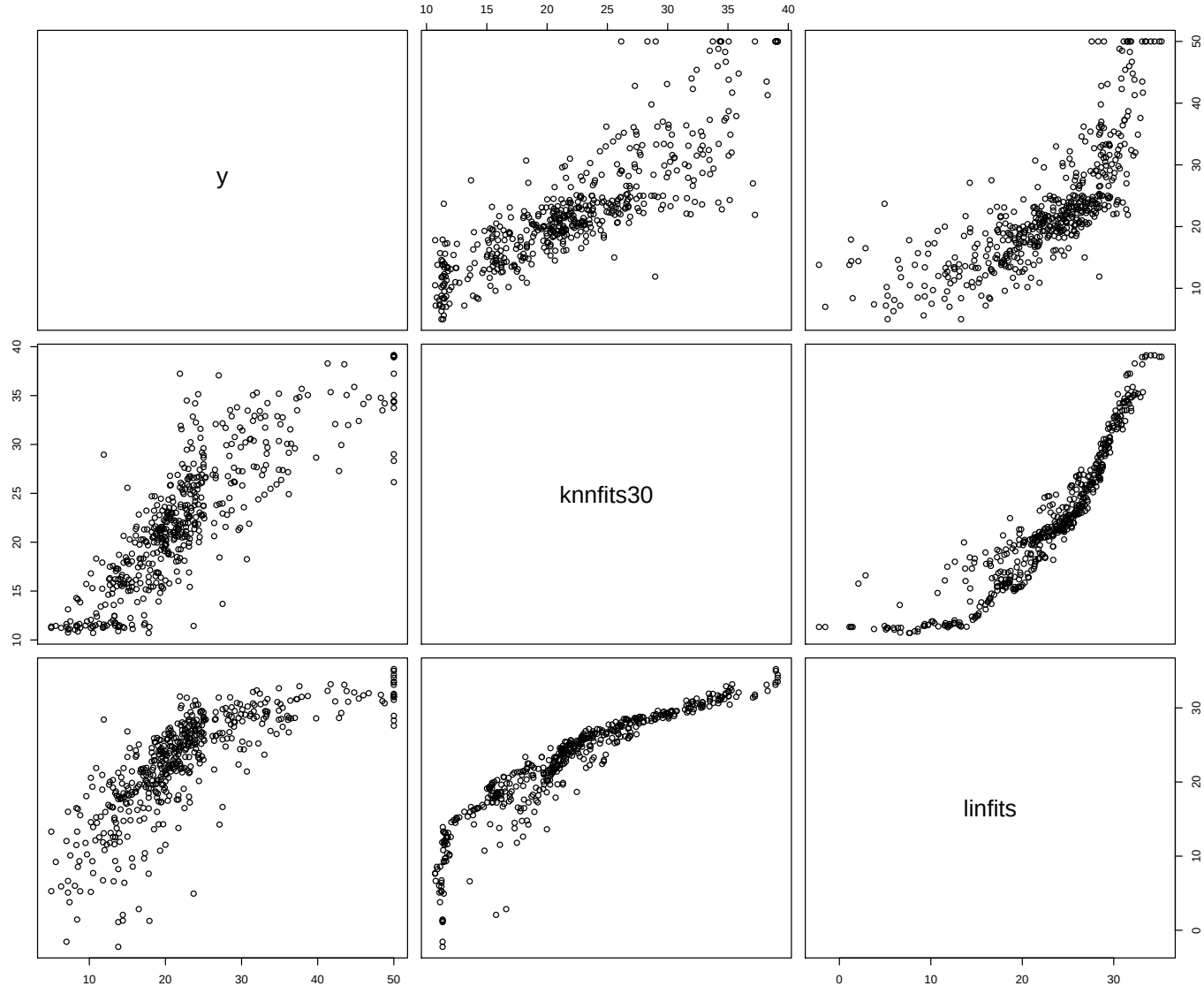
You can interpret x has "% of max".

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.



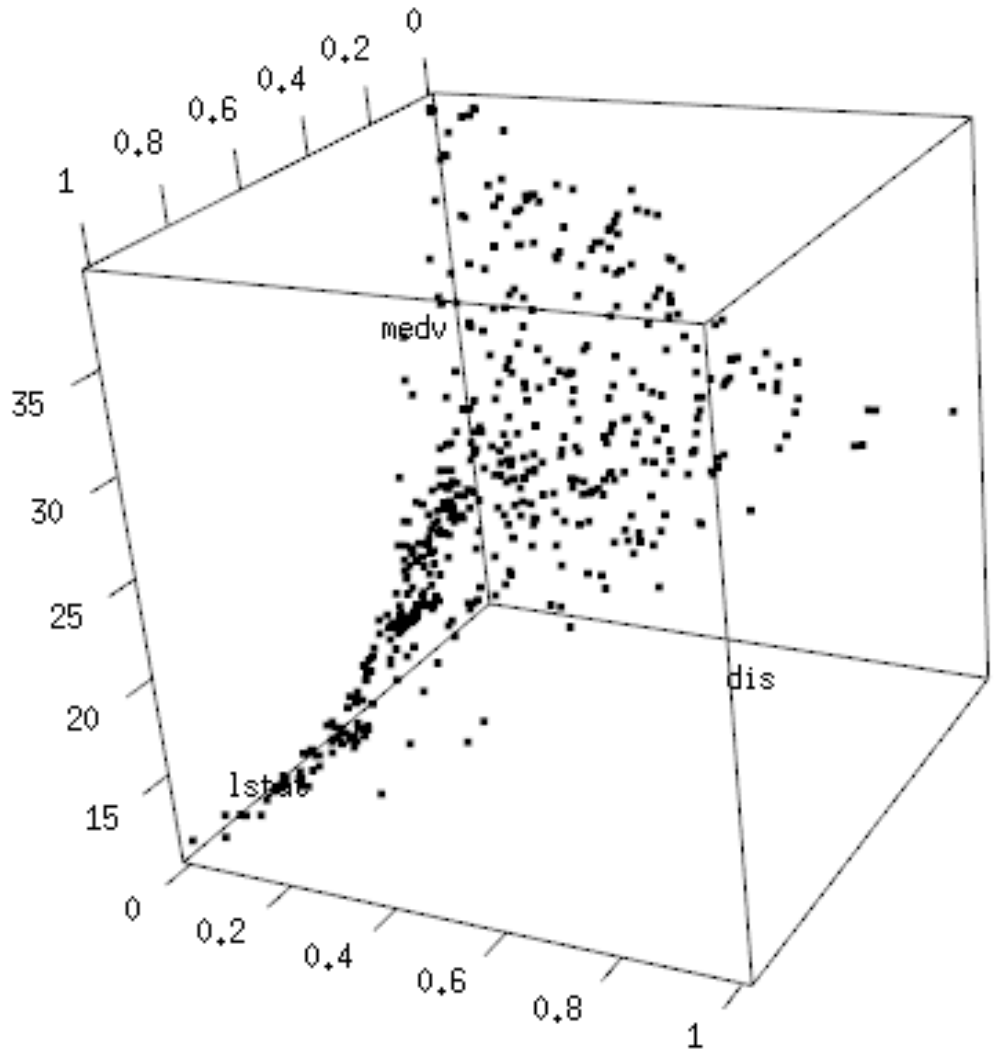
Here are the correlations corresponding to the plots.

	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.

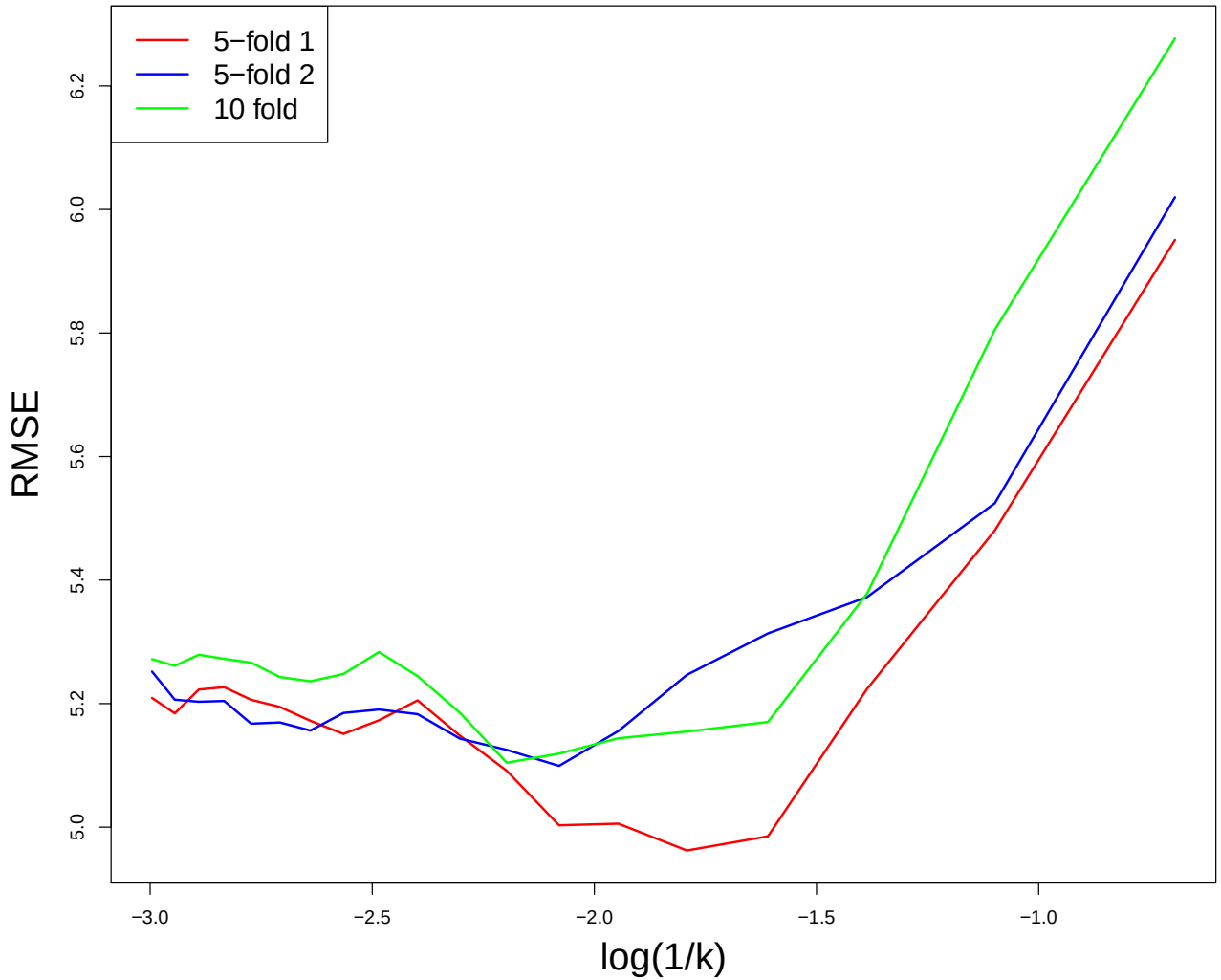
who cares !!

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



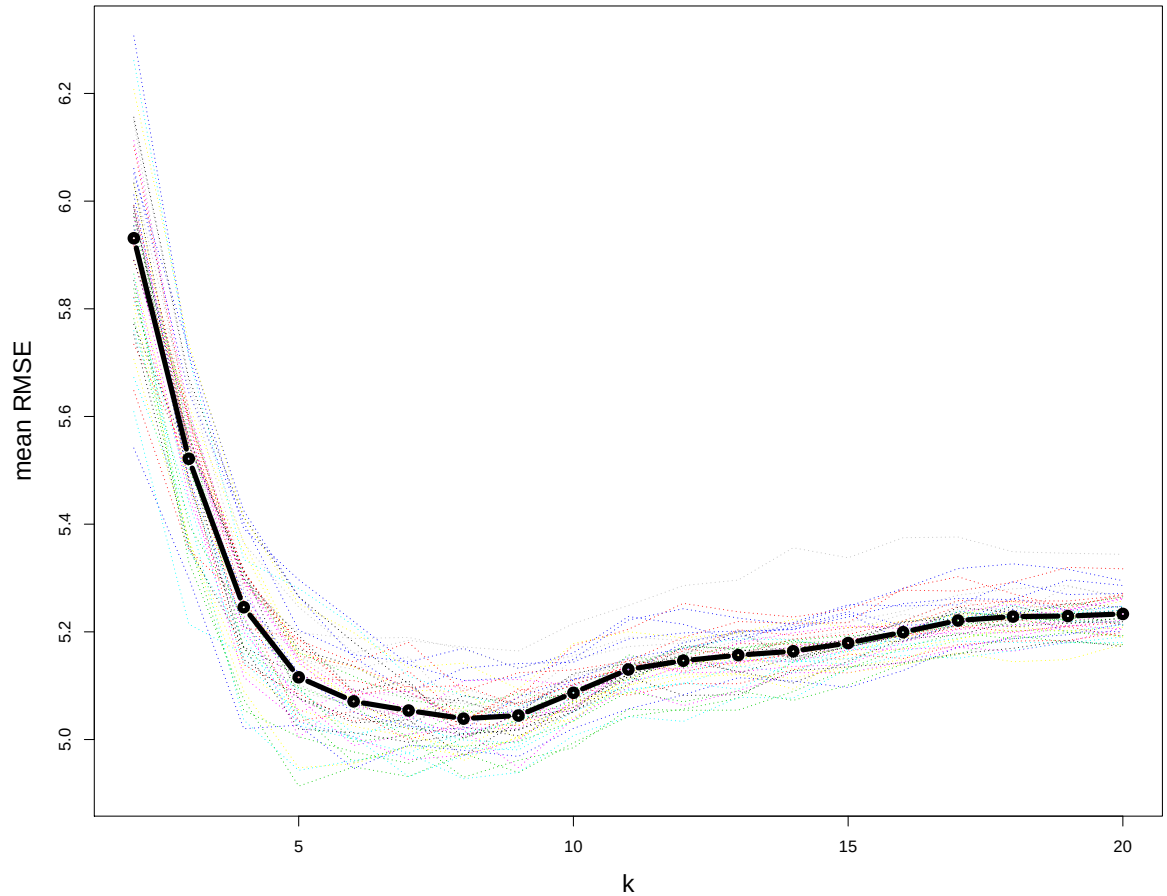
3-D visualization is not easy!!!

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



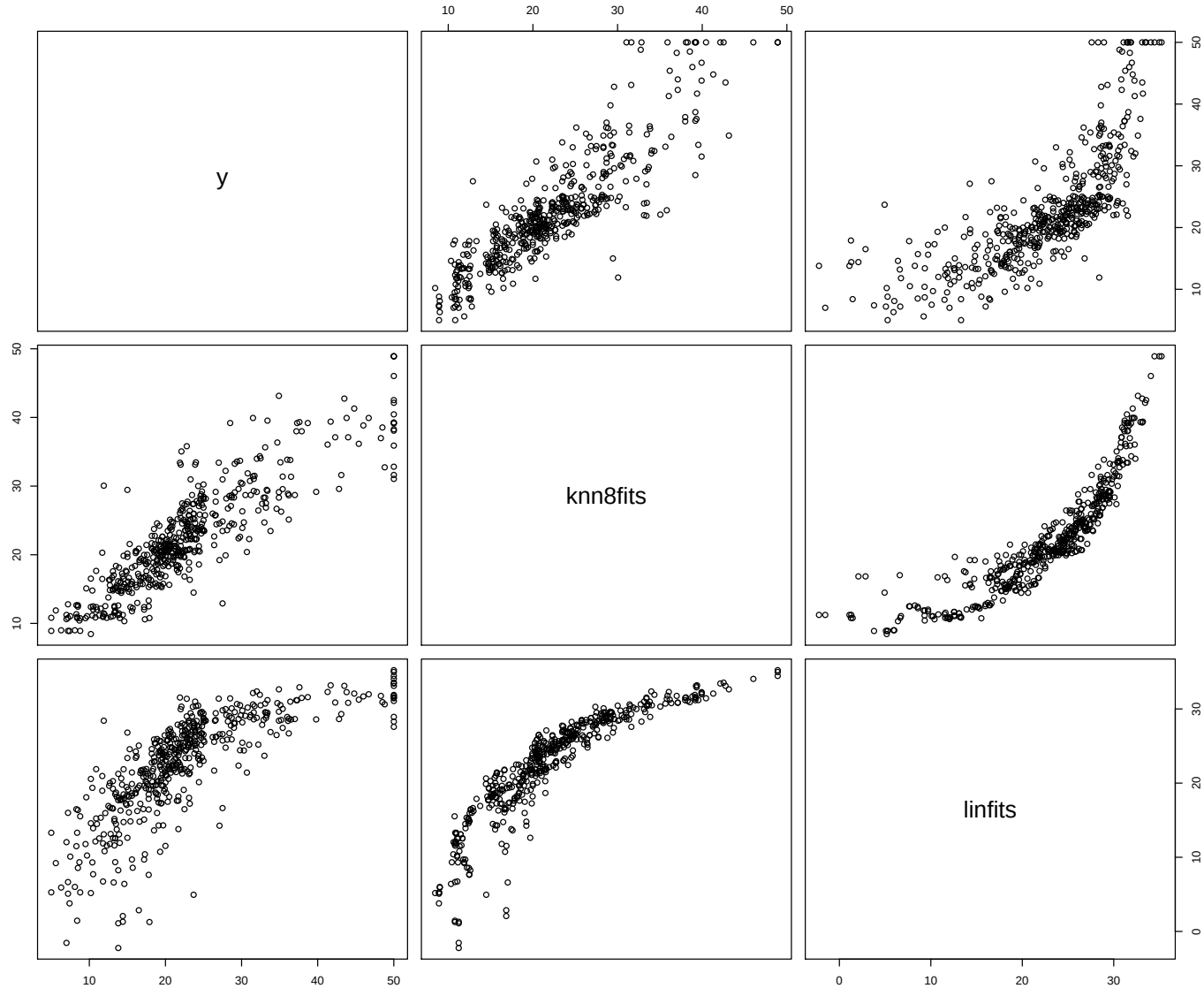
Seems to indicate a much smaller k (than when we just used `lstat`, but it is very noisy.

Let' average many CV's.



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

Refit using all the data and $k = 8$.
Here is a plot of the fits.



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.

Rescaling all the (numeric) x's is common.

Note:

Another rescaling that people often use (besides $(x - \min(x))/(\max(x) - \min(x))$ is

$$x \Rightarrow \frac{(x - \bar{x})}{\mathrm{sd}(x)}$$

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 !!!*

Le't try Boston using more of the x's.

Our Boston data set only has $n = 506$.

p is 13:

```
This data frame contains the following columns:

     crim per capita crime rate by town.
     zn proportion of residential land zoned for lots over 25,000
          sq.ft.
     indus proportion of non-retail business acres per town.
     chas Charles River dummy variable (= 1 if tract bounds river; 0
          otherwise).
     nox nitrogen oxides concentration (parts per 10 million).
     rm average number of rooms per dwelling.
     age proportion of owner-occupied units built prior to 1940.
     dis weighted mean of distances to five Boston employment
          centres.
     rad index of accessibility to radial highways.
     tax full-value property-tax rate per \$10,000.
     ptratio pupil-teacher ratio by town.
     black 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by
          town.
     lstat lower status of the population (percent).
     medv median value of owner-occupied homes in \$1000s.
```

Let's try $p = 4$ with nox, rm, ptratio, and lstat.

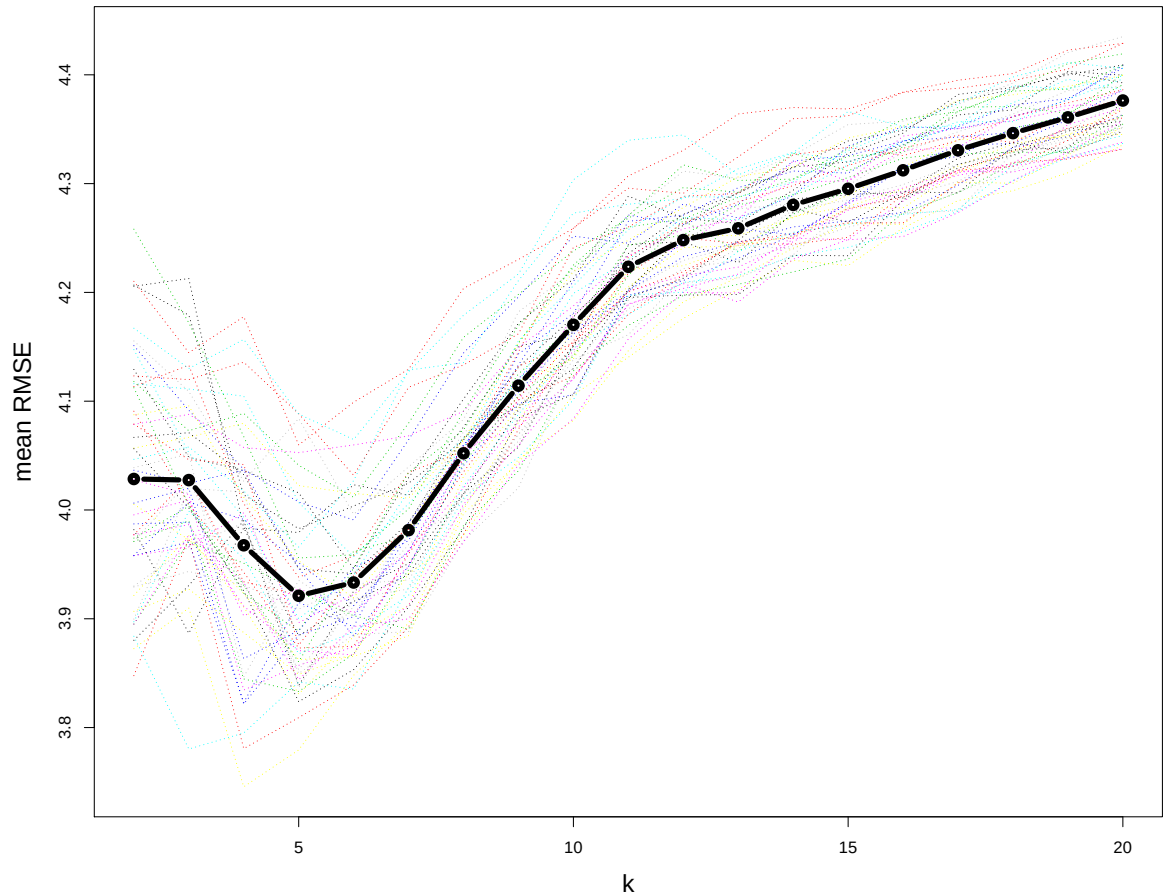
Then we'll try using all the x's ($p = 13$).

In each case we will mechanically rescale each x to be in [0,1].

One of the x's is a 0-1 dummy (chas). Recaling will not change it.

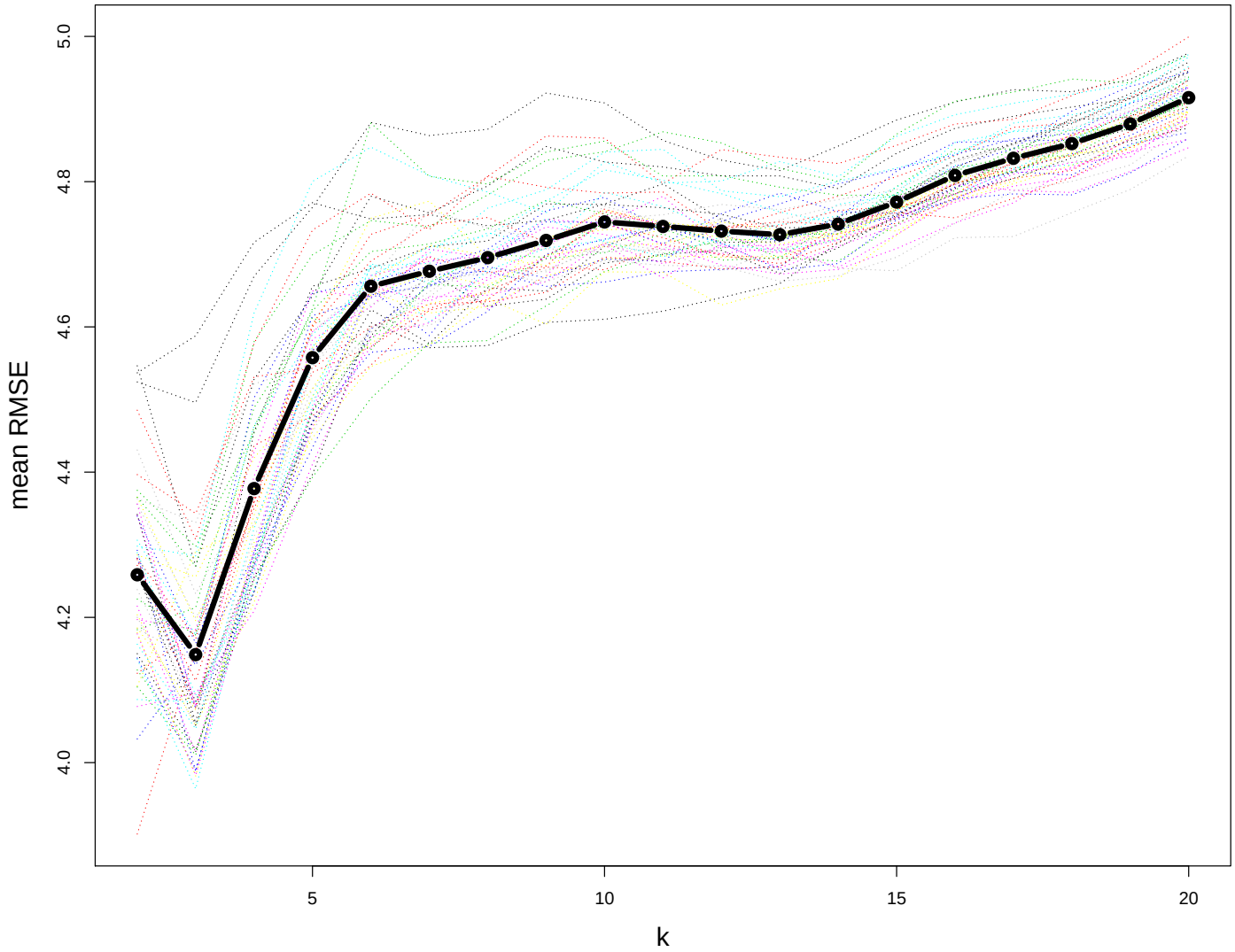
Does this make sense??

Here is the 10-fold CV with $p = 4$.



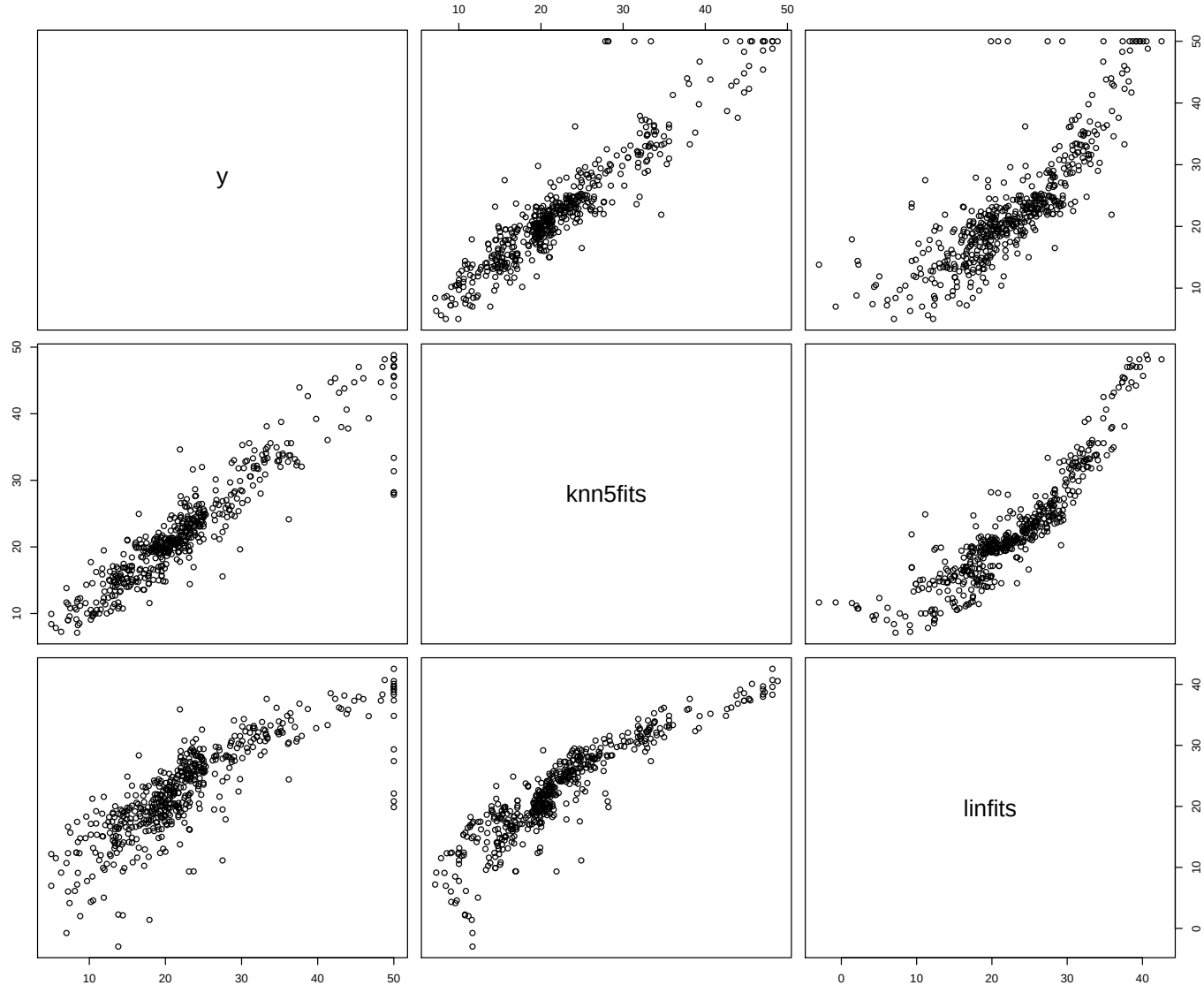
The minimum RMSE seems quite a bit better (3.9 rather than 5) than with $p = 2$
(we used lstat, dis).

Here is the 10-fold CV with $p = 13$.



Using more x's seems to make it worse!!

Refit using all the data, $p = 4$, $k = 5$.



```
              y knn5fits linfits
y          1.00     0.93    0.82
knn5fits 0.93       1.00    0.91
linfits   0.82      0.91    1.00
```

We have a nice result with $p = 4$, but I did not tell you how I got those 4!!

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.

In R:

Let's see how to use kNN and cross-validation in R.

Let's try x1=lstat,x2=indus with the Boston data.

```
#get the Boston data and needed libraries
library(kknn)
library(MASS)
source("docv.R") #we'll use Rob's code for cross-val with kNN

#get variables we want
x = cbind(Boston$lstat,Boston$indus)
colnames(x) = c("lstat","indus")
y = Boston$medv
mmsc=function(x) {return((x-min(x))/(max(x)-min(x)))}
xs = apply(x,2,mmsc) #apply scaling function to each column of x
```

```
#plot y vs each x
par(mfrow=c(1,2)) #two plot frames
plot(x[,1],y,xlab="lstat",ylab="medv")
plot(x[,2],y,xlab="indus",ylab="medv")

#run cross val once
par(mfrow=c(1,1))
set.seed(99)
kv = 2:20 #k values to try
n = length(y)
cvtemp = docvknn(xs,y,kv,nfold=10)
cvtemp = sqrt(cvtemp/n) #docvknn returns sum of squares
plot(kv,cvtemp)
```

```
#run cross val several times
set.seed(99)
cvmean = rep(0,length(kv)) #will keep average rmse here
ndocv = 50 #number of CV splits to try
n=length(y)
cvmat = matrix(0,length(kv),ndocv) #keep results for each split
for(i in 1:ndocv) {
   cvtemp = docvknn(xs,y,kv,nfold=10)
   cvmean = cvmean + cvtemp
   cvmat[,i] = sqrt(cvtemp/n)
}
cvmean = cvmean/ndocv
cvmean = sqrt(cvmean/n)
plot(kv,cvmean,type="n",ylim=range(cvmat),xlab="k",cex.lab=1.5)
for(i in 1:ndocv) lines(kv,cvmat[,i],col=i,lty=3) #plot each result
lines(kv,cvmean,type="b",col="black",lwd=5) #plot average result
```

```
#refit using all the data and k=5
ddf = data.frame(y,xs)
near5 = kknn(y~.,ddf,ddf,k=5,kernel = "rectangular")
lmf = lm(y~.,ddf)
fmat = cbind(y,near5$fitted,lmf$fitted)
colnames(fmat)=c("y","kNN5","linear")
pairs(fmat)
print(cor(fmat))

#predict price of house in place with lstat=10, indus=11.
x1=10; x2=11
x1s = (x1-min(x[,1]))/(max(x[,1])-min(x[,1]))
x2s = (x2-min(x[,2]))/(max(x[,2])-min(x[,2]))
near = kknn(y~.,ddf,data.frame(lstat=x1s,indus=x2s),k=5,kernel = "rectangular")
cat("knn predicted value: ",near$fitted,"\n")

#what does a linear model predict?
print(predict(lmf,data.frame(lstat=x1s,indus=x2s)))
#let's check we did the scaling right
lmtemp = lm(medv~lstat+indus,Boston)
print(predict(lmtemp,data.frame(lstat=10,indus=11)))
```

7. 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:

(1)

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.

(2)

We will start by leaving out a "test" or "validation" 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?

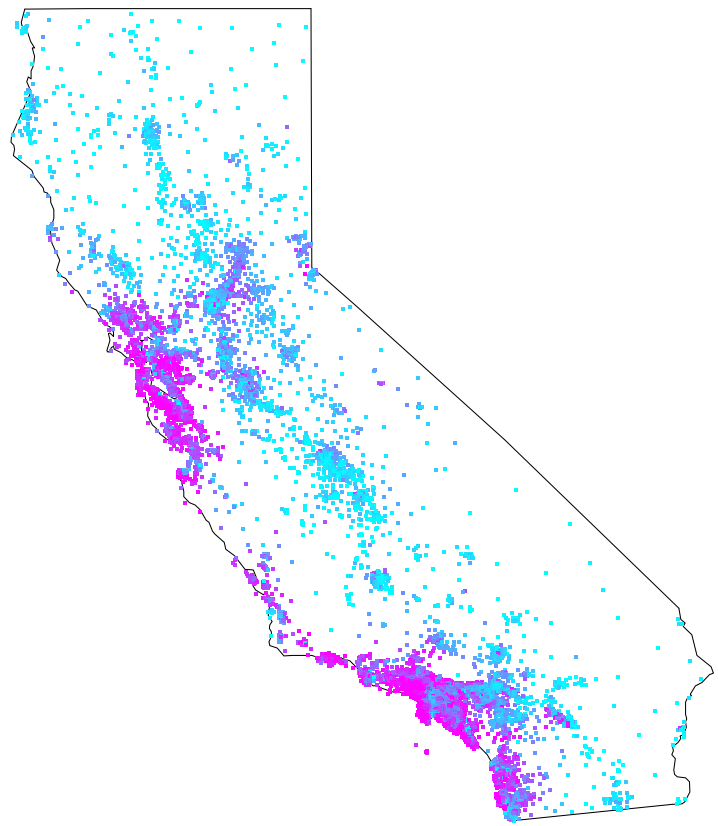
Goal: Predict $log(MedianValue)$ (why logs? more on this later)

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

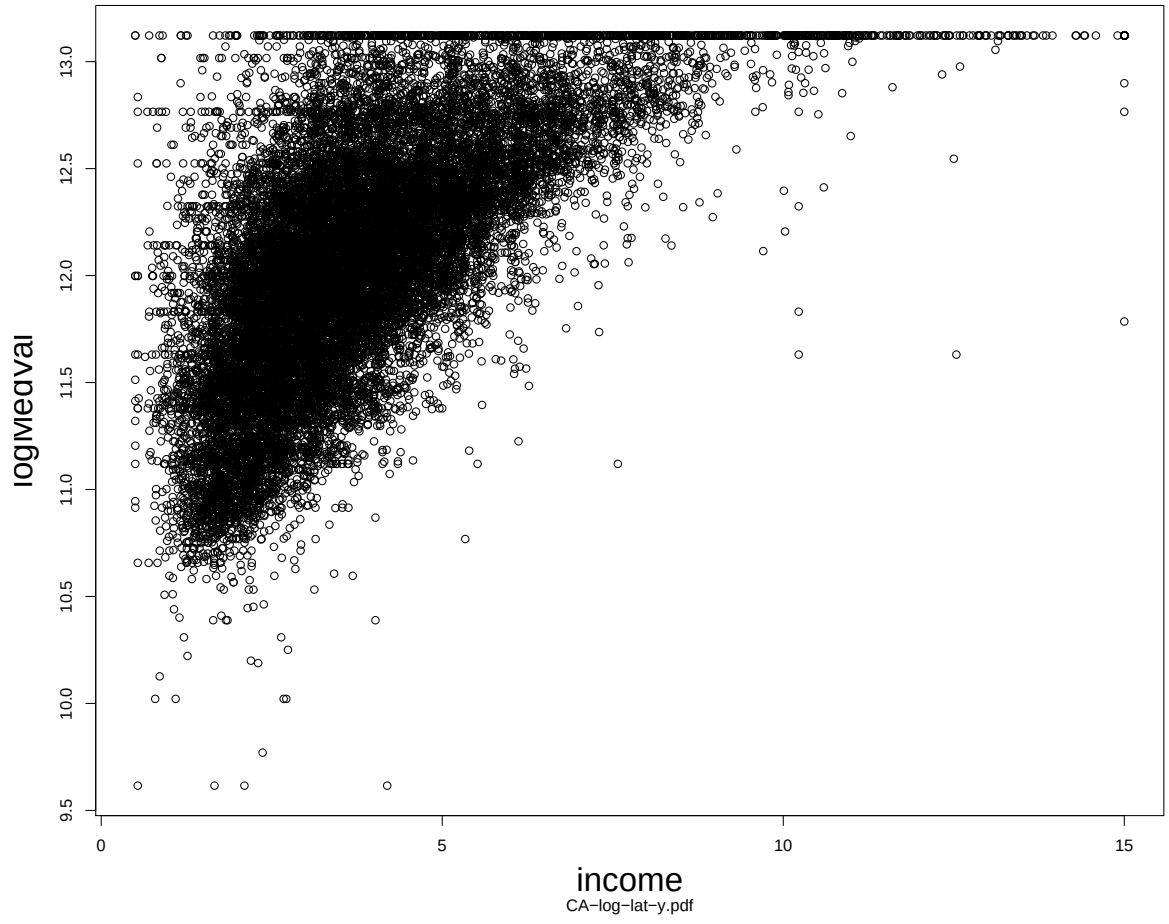
We should spend a long time plotting the data, but let's suppose we are in a hurry.

A couple of plots:

y=logMedVal
vs longitude and latitude.



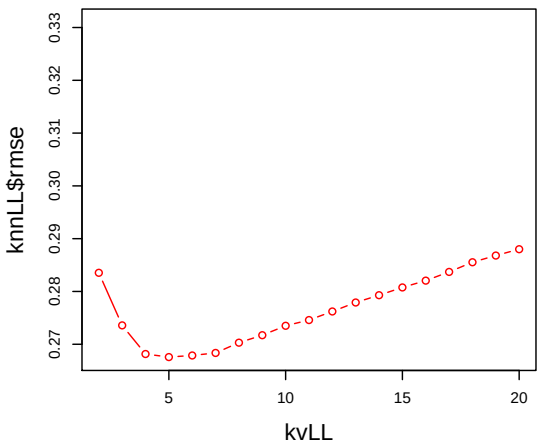
y=logMedVal
vs median income.

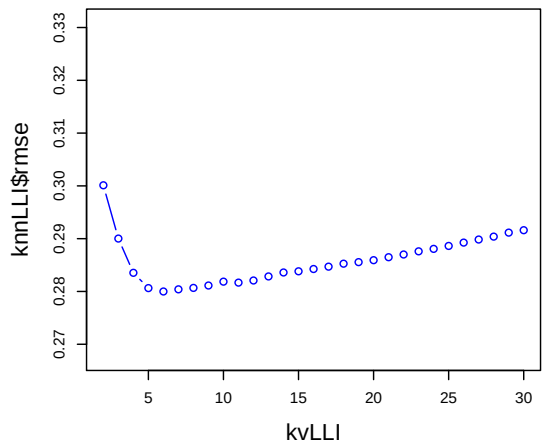


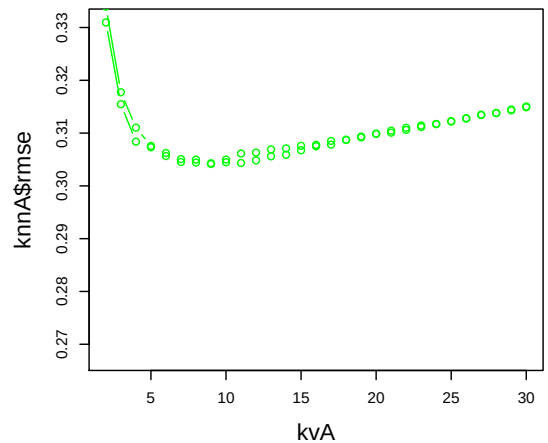
10,000 in train. Rest in test. Standardize each x: (x-m)/s.

Do 5-fold cross-validation on train.

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

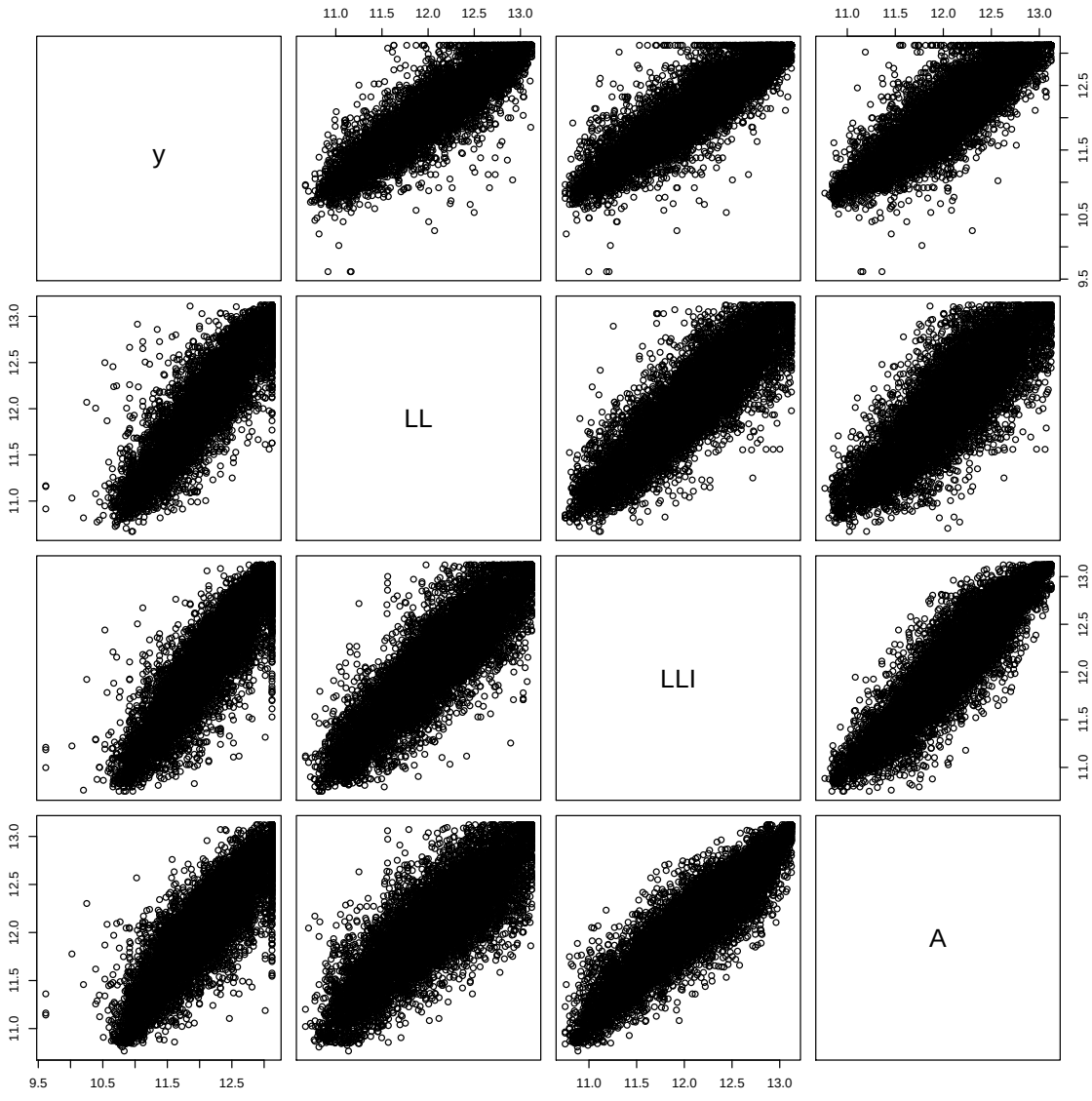






Pretty small k values.
All x is worst.

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



Here are the correlations between the test y=logMedVal and the (out-of-sample!!) predictions.

```
             y         LL        LLI          A
y    1.0000000 0.8963260 0.8877891 0.8615422
LL   0.8963260 1.0000000 0.8788248 0.8323511
LLI  0.8877891 0.8788248 1.0000000 0.9061848
A    0.8615422 0.8323511 0.9061848 1.0000000
```

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.....