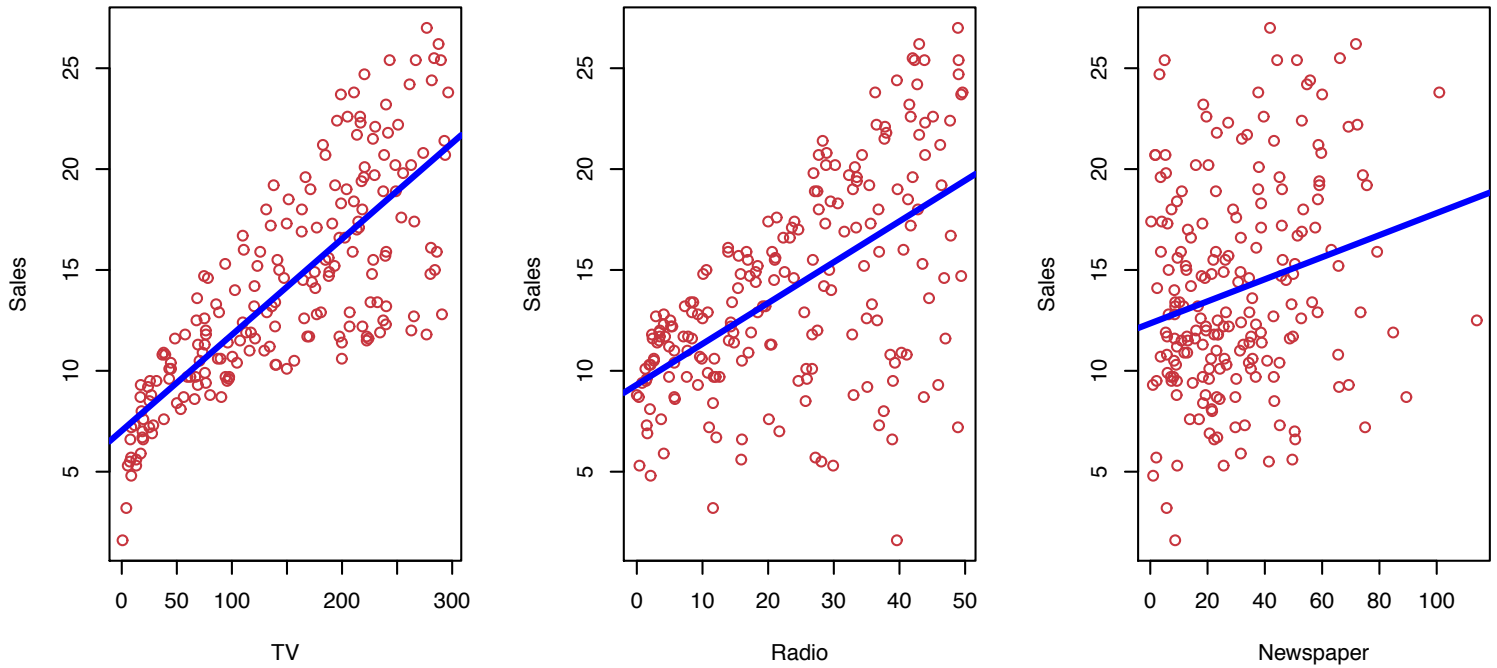
2.1 Introduction to Regression Models

https://youtu.be/WjyuiK5taS8

What is Statistical Learning?



Shown are `Sales` vs `TV`, `Radio` and `Newspaper`, with a blue linear-regression line fit separately to each.
Can we predict `Sales` using these three?
Perhaps we can do better using a model

$$\texttt{Sales} \approx f(\texttt{TV}, \texttt{Radio}, \texttt{Newspaper})$$

Notation

Here `Sales` is a *response* or *target* that we wish to predict. We generically refer to the response as Y.
`TV` is a *feature*, or *input*, or *predictor*; we name it X_1.
Likewise name `Radio` as X_2, and so on.
We can refer to the *input vector* collectively as

$$X = \begin{pmatrix} X_1 \\ X_2 \\ X_3 \end{pmatrix}$$

column vector

Now we write our model as

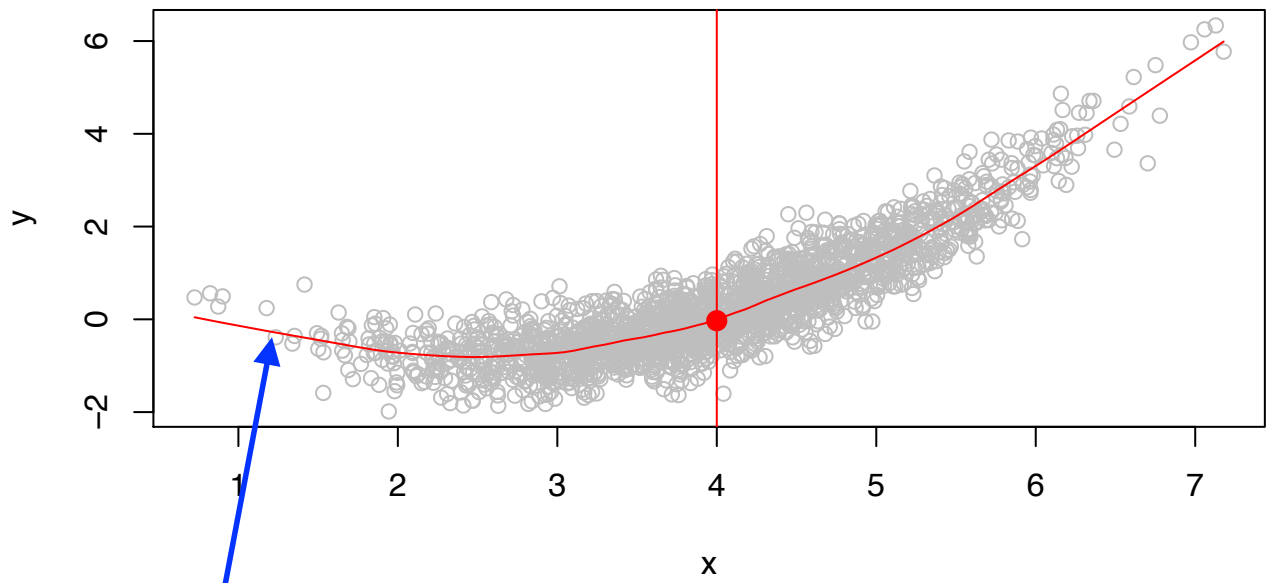
$$Y = f(X) + \epsilon$$

where ϵ captures measurement errors and other discrepancies.

there's going to be a lot of things we can't capture with the function, that's caught up in the error.

What is $f(X)$ good for?

- With a good f we can make predictions of Y at new points $X = x$.
- We can understand which components of $X = (X_1, X_2, \ldots, X_p)$ are important in explaining Y, and which are irrelevant. e.g. `Seniority` and `Years of Education` have a big impact on `Income`, but `Marital Status` typically does not.
- Depending on the complexity of f, we may be able to understand how each component X_j of X affects Y.



Is there an ideal $f(X)$? In particular, what is a good value for $f(X)$ at any selected value of X, say $X = 4$? There can be many Y values at $X = 4$. A good value is

that says the function at the value 4 is the expected value of Y given X equals 4.

$$f(4) = E(Y|X = 4)$$

it's actually a conditional average given X equals 4

$E(Y|X = 4)$ means *expected value* (average) of Y given $X = 4$.

expected value is a fancy word for average

This ideal $f(x) = E(Y|X = x)$ is called the *regression function*.

the regression function gives you the conditional expectation of Y given X at each value of X.

The regression function $f(x)$

- Is also defined for vector X; e.g. $f(x) = f(x_1, x_2, x_3) = E(Y|X_1 = x_1, X_2 = x_2, X_3 = x_3)$
- Is the *ideal* or *optimal* predictor of Y with regard to mean-squared prediction error: $f(x) = E(Y|X = x)$ is the function that minimizes $E[(Y - g(X))^2|X = x]$ over all functions g at all points $X = x$.
- $\epsilon = Y - f(x)$ is the *irreducible* error — i.e. even if we knew $f(x)$, we would still make errors in prediction, since at each $X = x$ there is typically a distribution of possible Y values.
- For any estimate $\hat{f}(x)$ of $f(x)$, we have

$$E[(Y - \hat{f}(X))^2|X = x] = \underbrace{[f(x) - \hat{f}(x)]^2}_{\textit{Reducible}} + \underbrace{\text{Var}(\epsilon)}_{\textit{Irreducible}}$$

So if we want to improve our model, it's this first piece, the reducible piece that we can improve by maybe changing the way we estimate f(X)

How to estimate f

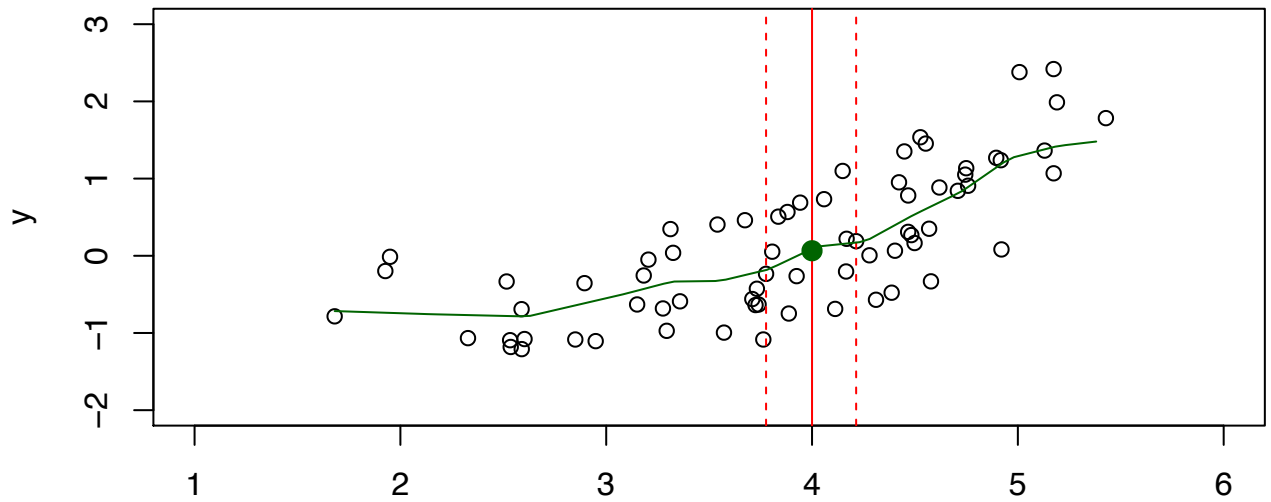
- Typically we have few if any data points with $X = 4$ exactly. We might not have any (or many) points to average
- So we cannot compute $E(Y|X = x)$!
- Relax the definition and let

compute the nearest neighbor average

$$\hat{f}(x) = \text{Ave}(Y|X \in \mathcal{N}(x))$$

where $\mathcal{N}(x)$ is some *neighborhood* of x.

this is called nearest neighbors or local averaging.



It's not going to be perfect because the little window has a certain width, so some points of the true f may be lower and some points higher. But on average, it does quite well.

2.2 Dimensionality and Structured Models

https://youtu.be/UvxHOkYQl8g

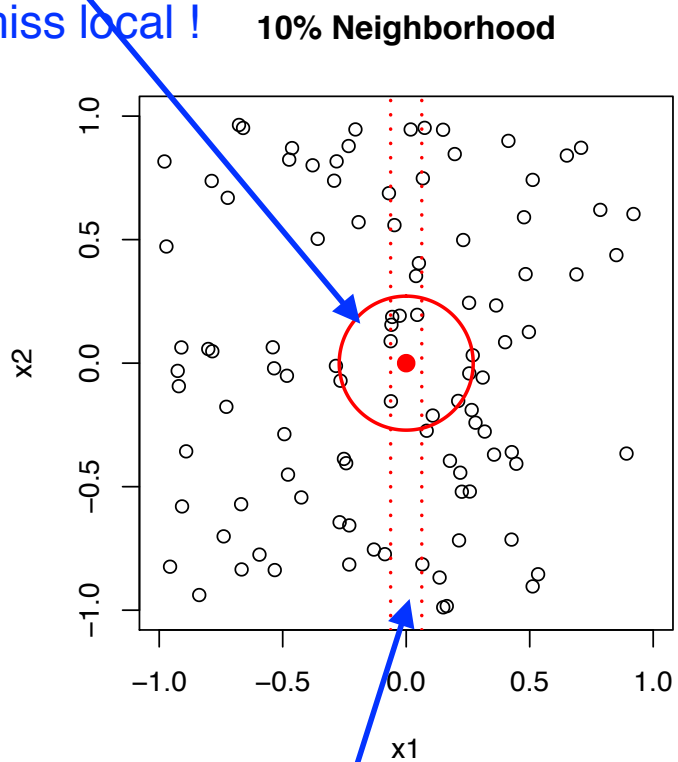
- Nearest neighbor averaging can be pretty good for small p — i.e. $p \leq 4$ and large-ish N.
- We will discuss smoother versions, such as kernel and spline smoothing later in the course.
- Nearest neighbor methods can be *lousy* when p is large. Reason: the *curse of dimensionality*. Nearest neighbors tend to be far away in high dimensions.

Large N so that we have enough points in each neighbor to average to give us our estimate.

  - We need to get a reasonable fraction of the N values of y_i to average to bring the variance down—e.g. 10%.
  - A 10% neighborhood in high dimensions need no longer be local, so we lose the spirit of estimating $E(Y|X = x)$ by local averaging.

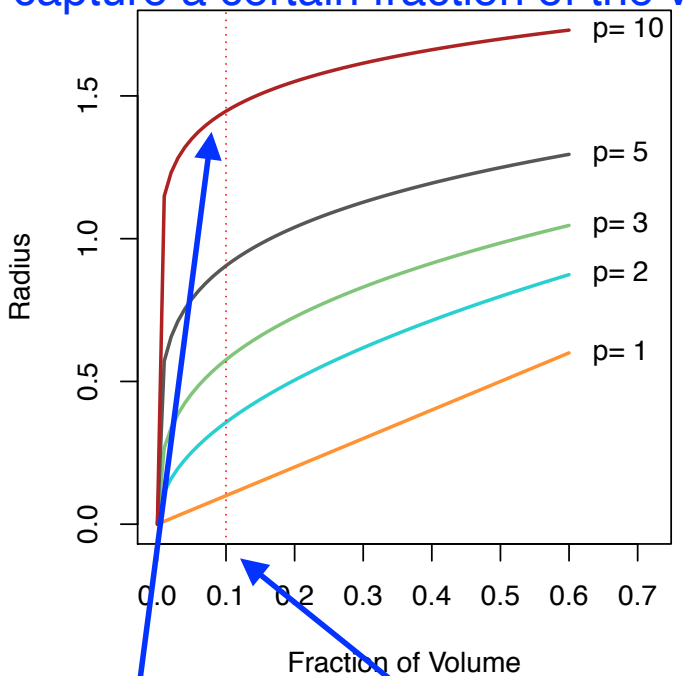
The curse of dimensionality

if we consider 2d (x1,x2), circle region contains 10%, now the radius of circle in 2d is lager than radius of circle in 1d => we miss local !



if we consider 1d (x1, ignore x2), region between dash lines contains 10%

how far you have to go out in 1 ~ 10 dimensions. These are different versions of this problem as the dimensions get higher In order to capture a certain fraction of the volume.



in 10 d, you actually have to go break out of this sphere on each coordinate axes to get 10% of the data.

the bottom line is it's really hard to find new neighborhoods in high dimensions and stay local.

So how do we deal with this? Well, we introduce structure to our models.

Parametric and structured models

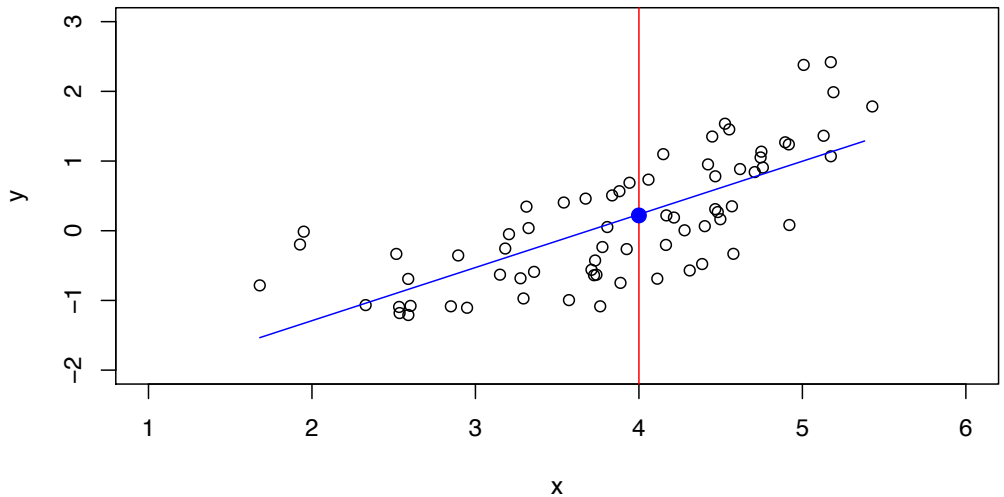
The *linear* model is an important example of a parametric model:

$$f_L(X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots \beta_p X_p.$$

structural model is going to avoid the curse of dimensionality because it's not relying on any local properties and nearest neighbor averaging. That's just fitting a rigid model to all the data.

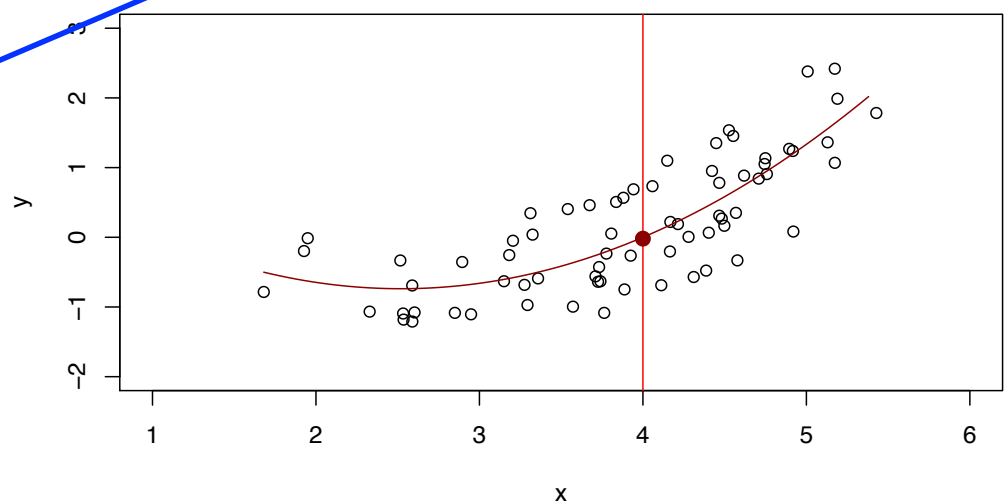
- A linear model is specified in terms of $p + 1$ parameters $\beta_0, \beta_1, \ldots, \beta_p$.
- We estimate the parameters by fitting the model to training data.
- Although it is *almost never correct*, a linear model often serves as a good and interpretable approximation to the unknown true function $f(X)$.

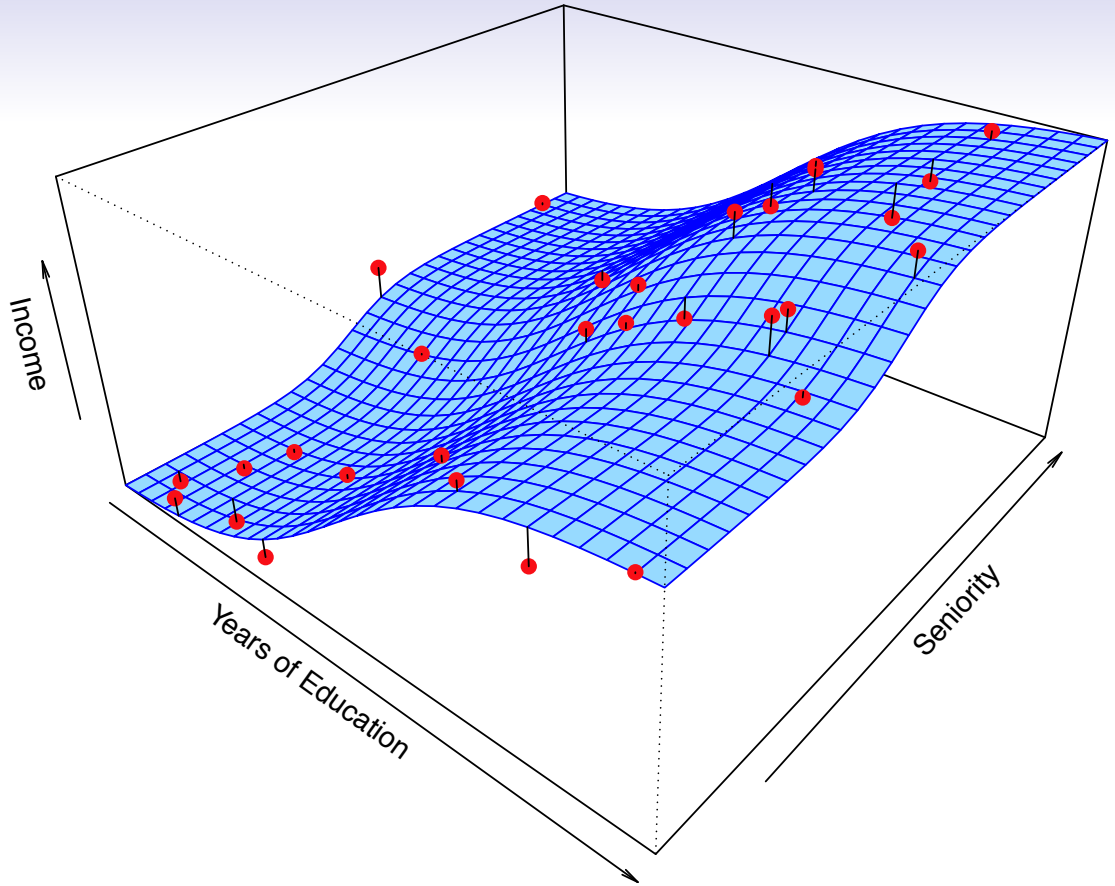
A linear model $\hat{f}_L(X) = \hat{\beta}_0 + \hat{\beta}_1 X$ gives a reasonable fit here



A quadratic model $\hat{f}_Q(X) = \hat{\beta}_0 + \hat{\beta}_1 X + \hat{\beta}_2 X^2$ fits slightly better.

These little hats indicate that they've been estimated from the training data.

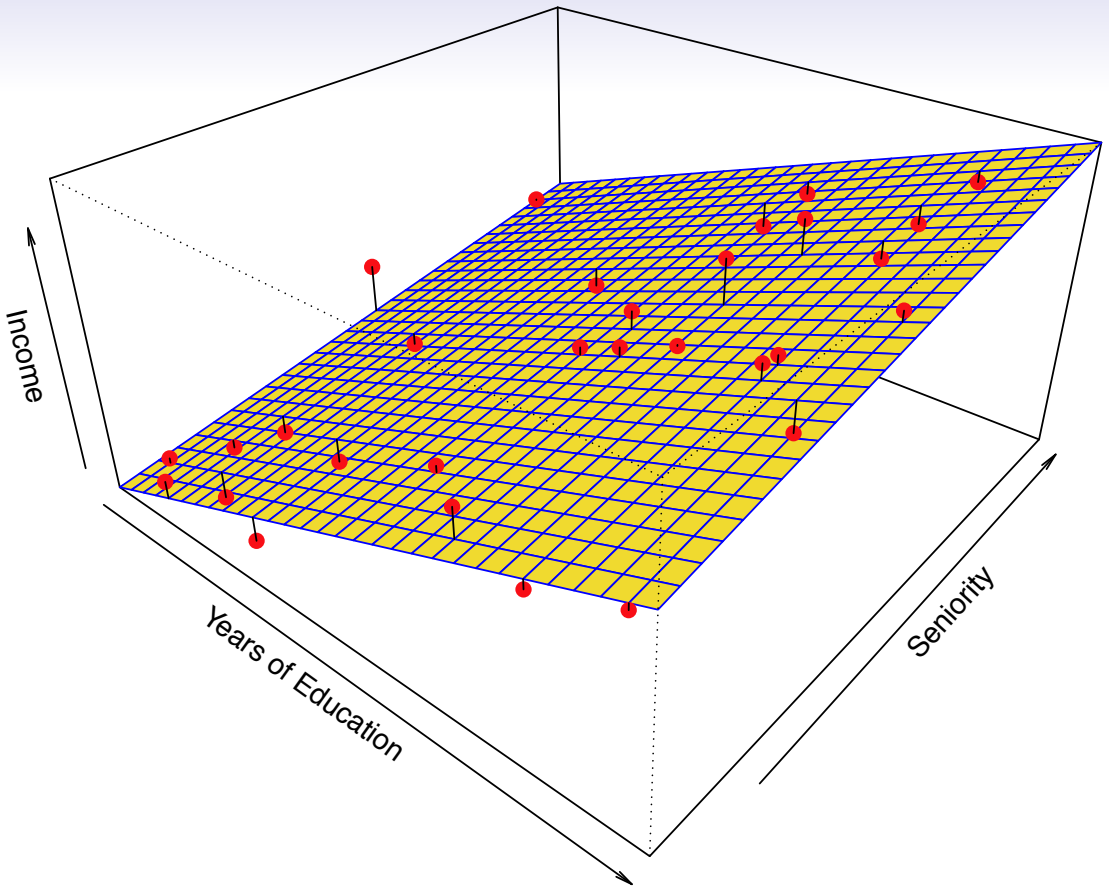




Simulated example. Red points are simulated values for `income` from the model

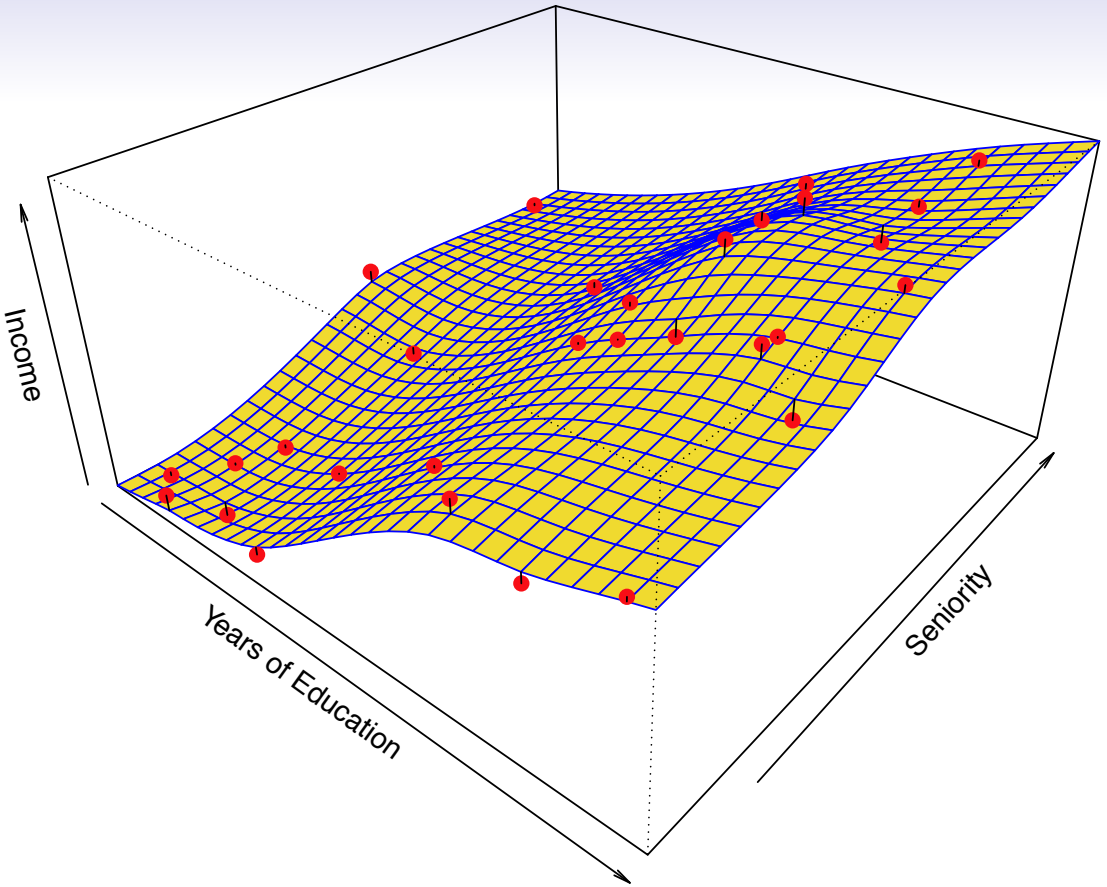
$$\texttt{income} = f(\texttt{education}, \texttt{seniority}) + \epsilon$$

f is the blue surface.

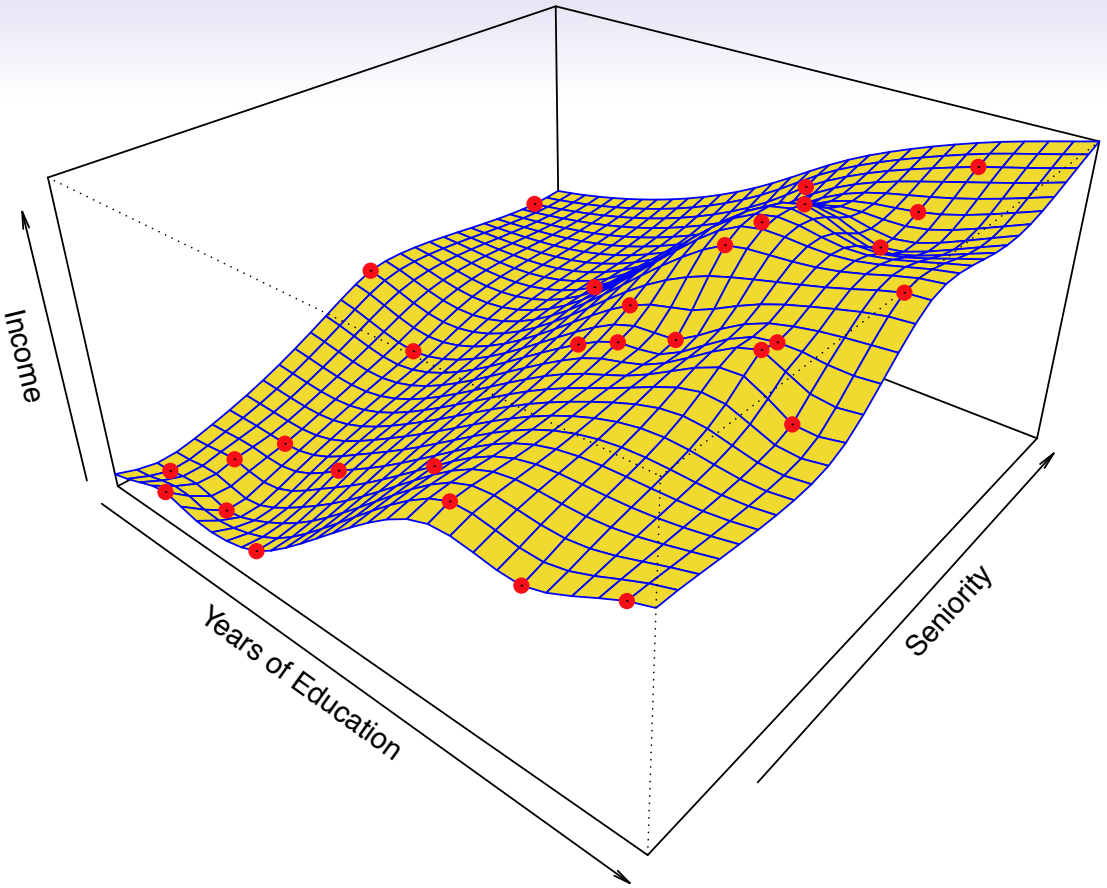


Linear regression model fit to the simulated data.

$$\hat{f}_L(\texttt{education}, \texttt{seniority}) = \hat{\beta}_0 + \hat{\beta}_1 \times \texttt{education} + \hat{\beta}_2 \times \texttt{seniority}$$



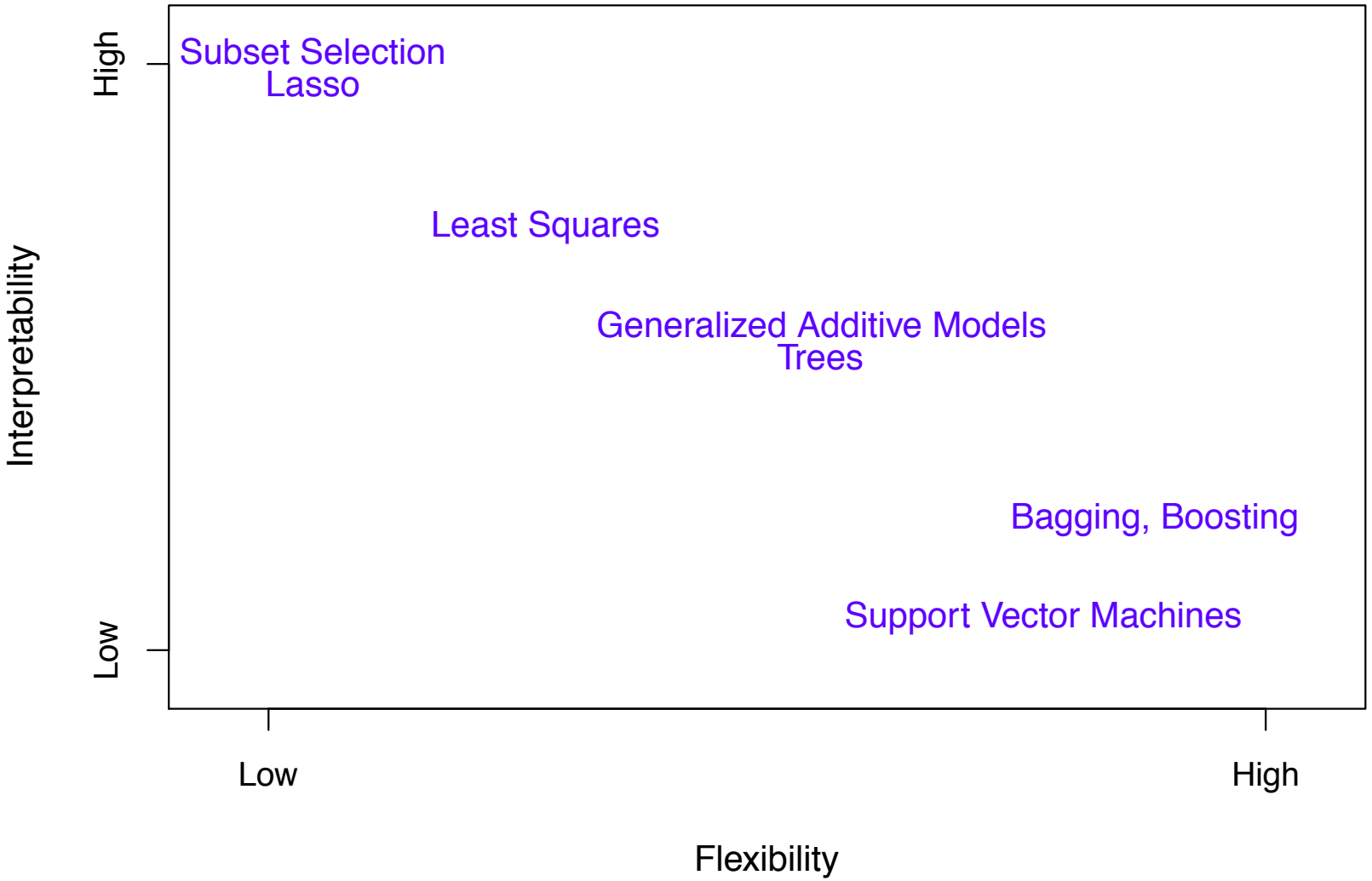
More flexible regression model $\hat{f}_S(\texttt{education}, \texttt{seniority})$ fit to the simulated data. Here we use a technique called a *thin-plate spline* to fit a flexible surface. We control the roughness of the fit (chapter 7).



Even more flexible spline regression model $\hat{f}_S(\texttt{education}, \texttt{seniority})$ fit to the simulated data. Here the fitted model makes no errors on the training data! Also known as *overfitting*.

Some trade-offs

- Prediction accuracy versus interpretability.
  — Linear models are easy to interpret; thin-plate splines are not.
- Good fit versus over-fit or under-fit.
  — How do we know when the fit is just right?
- Parsimony versus black-box.
  — We often prefer a simpler model involving fewer variables over a black-box predictor involving them all.


Subset Selection
Lasso
Least Squares
Generalized Additive Models
Trees
Bagging, Boosting
Support Vector Machines
High
Low
Interpretability
Low
High
Flexibility

2.3 Model Selection and Bias-Variance Tradeoff

https://youtu.be/VusKAosxxyk

Assessing Model Accuracy

Suppose we fit a model $\hat{f}(x)$ to some training data $\mathsf{Tr} = \{x_i, y_i\}_1^N$, and we wish to see how well it performs.

- We could compute the average squared prediction error over Tr:

  training data

  Mean Squared Error (MSE)

  $$\text{MSE}_{\mathsf{Tr}} = \text{Ave}_{i \in \mathsf{Tr}}[y_i - \hat{f}(x_i)]^2$$

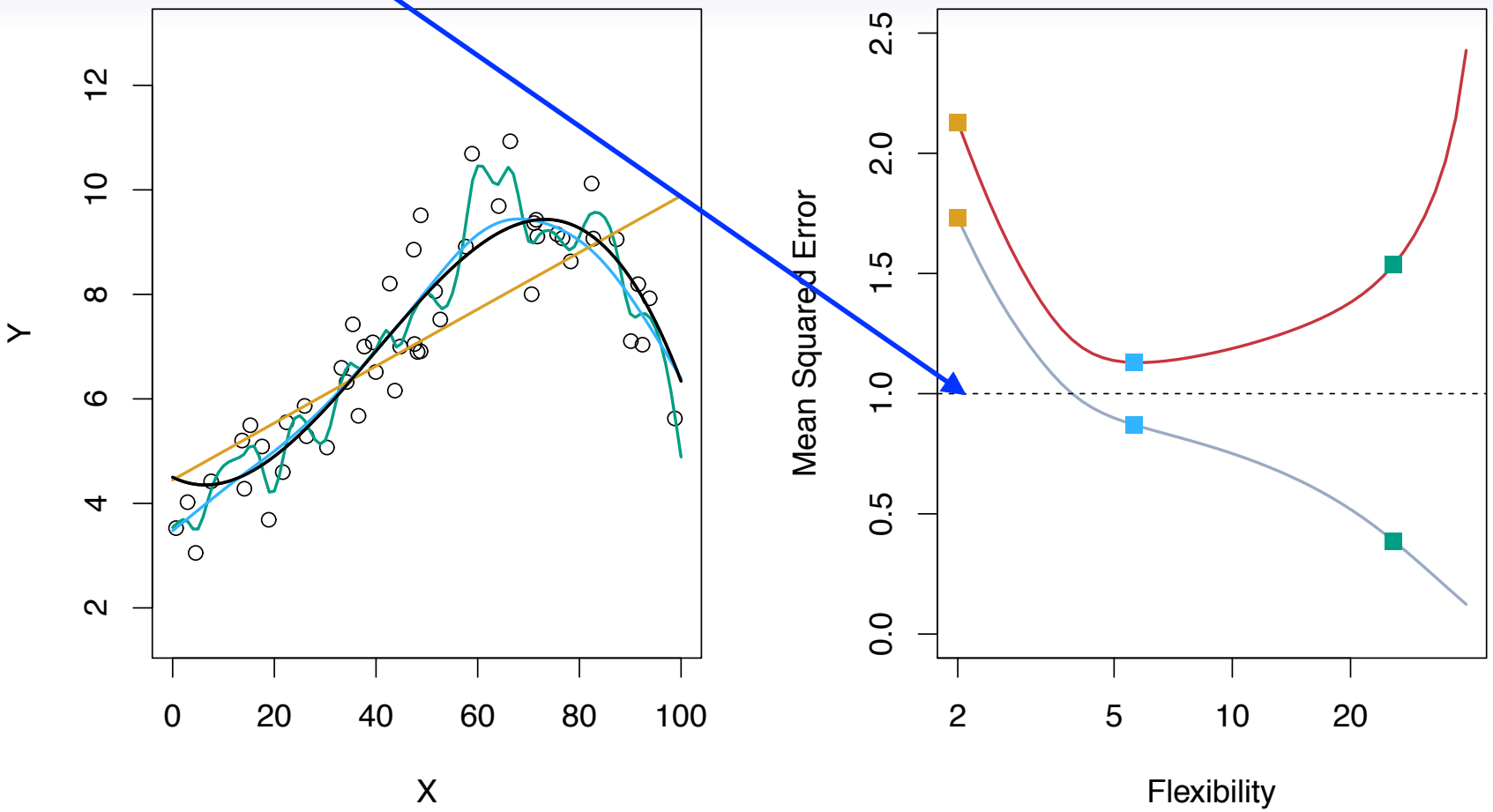
This may be biased toward more overfit models.

- Instead we should, if possible, compute it using fresh *test* data $\mathsf{Te} = \{x_i, y_i\}_1^M$:

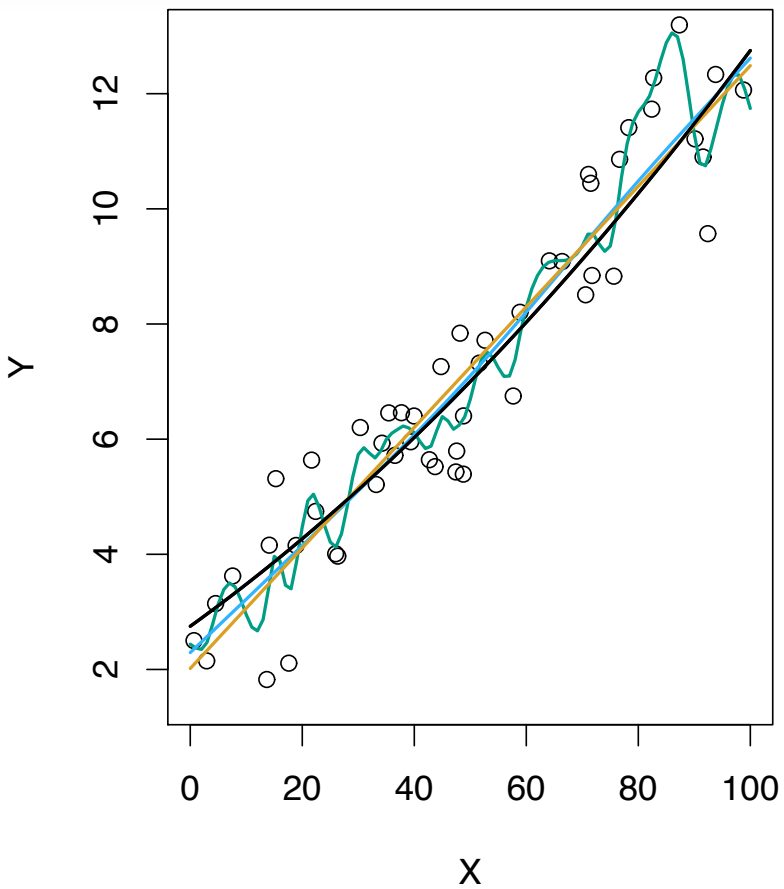
  test data

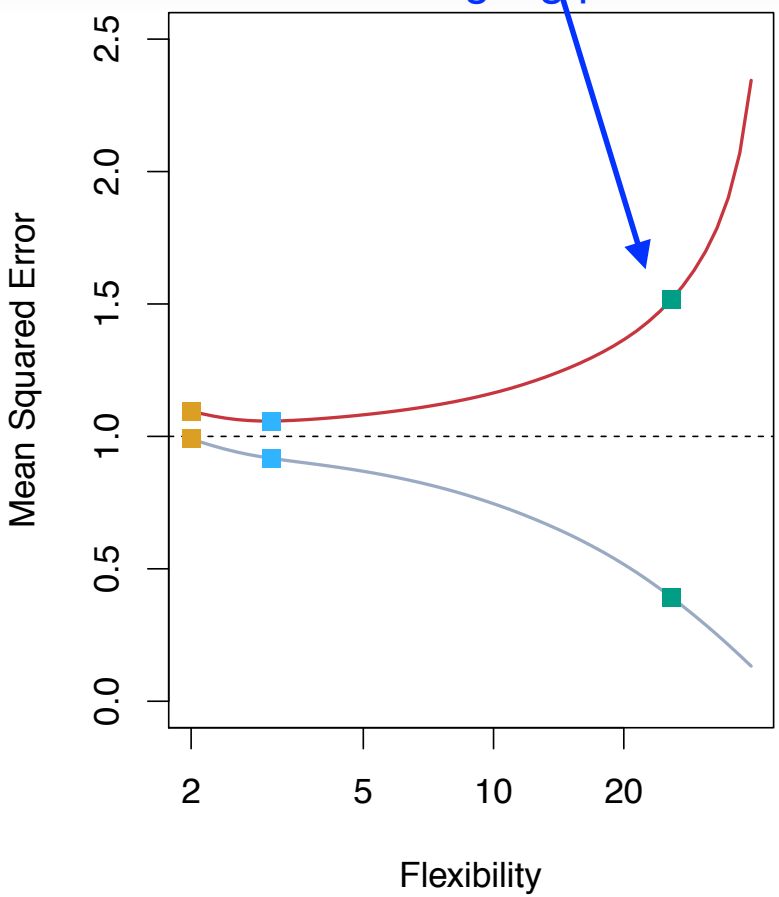
  $$\text{MSE}_{\mathsf{Te}} = \text{Ave}_{i \in \mathsf{Te}}[y_i - \hat{f}(x_i)]^2$$

horizontal dotted line is the mean squared error that the true function makes for data from this population, and that is the “irreducible error”, which we call the “variance of epsilon”

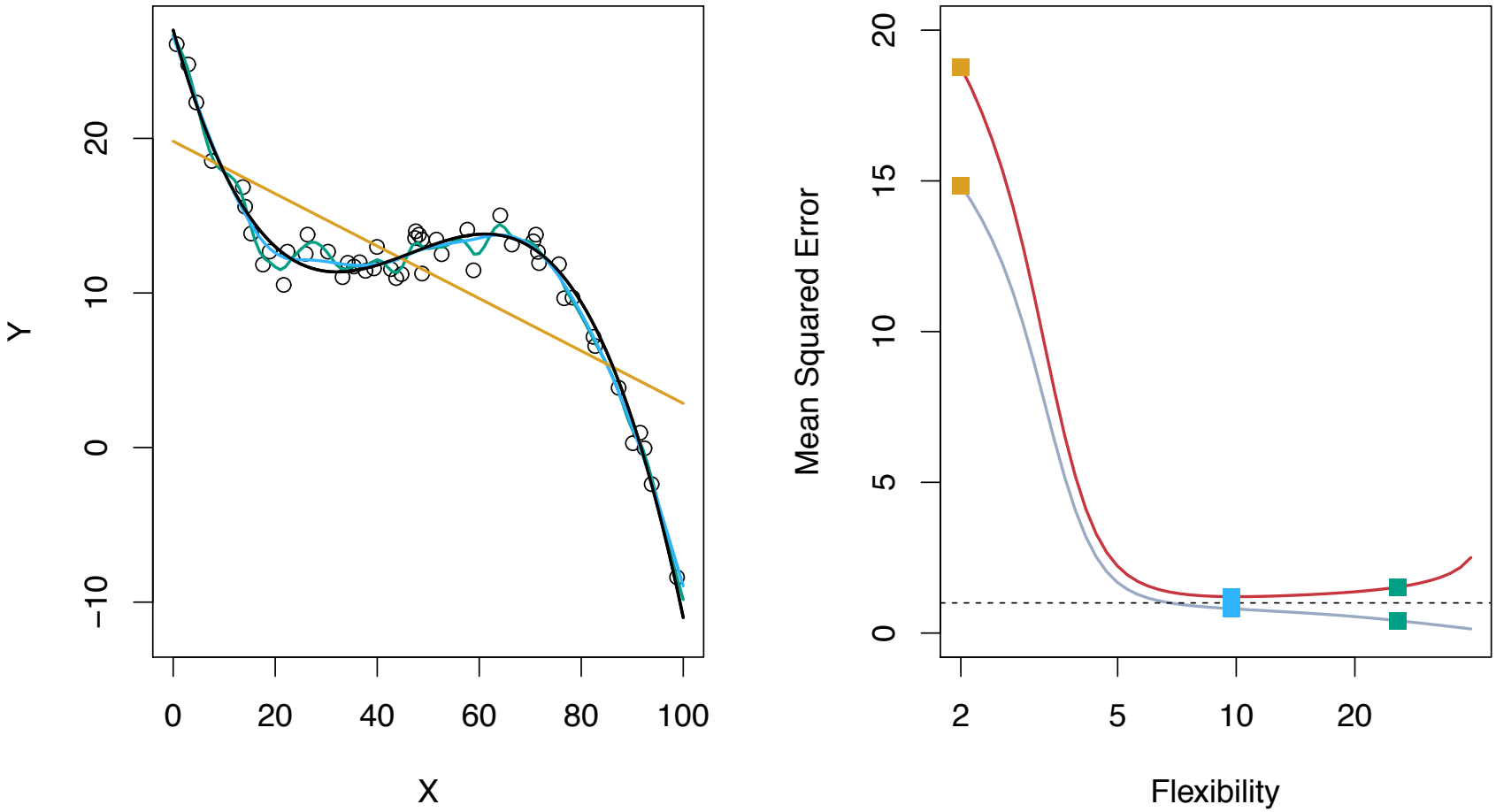


Black curve is truth. Red curve on right is MSE_{Te}, grey curve is MSE_{Tr}. Orange, blue and green curves/squares correspond to fits of different flexibility.





Here the truth is smoother, so the smoother fit and linear model do really well.



Here the truth is wiggly and the noise is low, so the more flexible fits do the best.

Bias-Variance Trade-off

Suppose we have fit a model $\hat{f}(x)$ to some training data Tr, and let (x_0, y_0) be a test observation drawn from the population. If the true model is $Y = f(X) + \epsilon$ (with $f(x) = E(Y|X = x)$), then

f = true model = regression function or the conditional expectation in the population.

$$E\left(y_0 - \hat{f}(x_0)\right)^2 = \text{Var}(\hat{f}(x_0)) + [\text{Bias}(\hat{f}(x_0))]^2 + \text{Var}(\epsilon).$$

The expectation averages over the variability of y_0 as well as the variability in Tr. Note that $\text{Bias}(\hat{f}(x_0))] = E[\hat{f}(x_0)] - f(x_0)$.

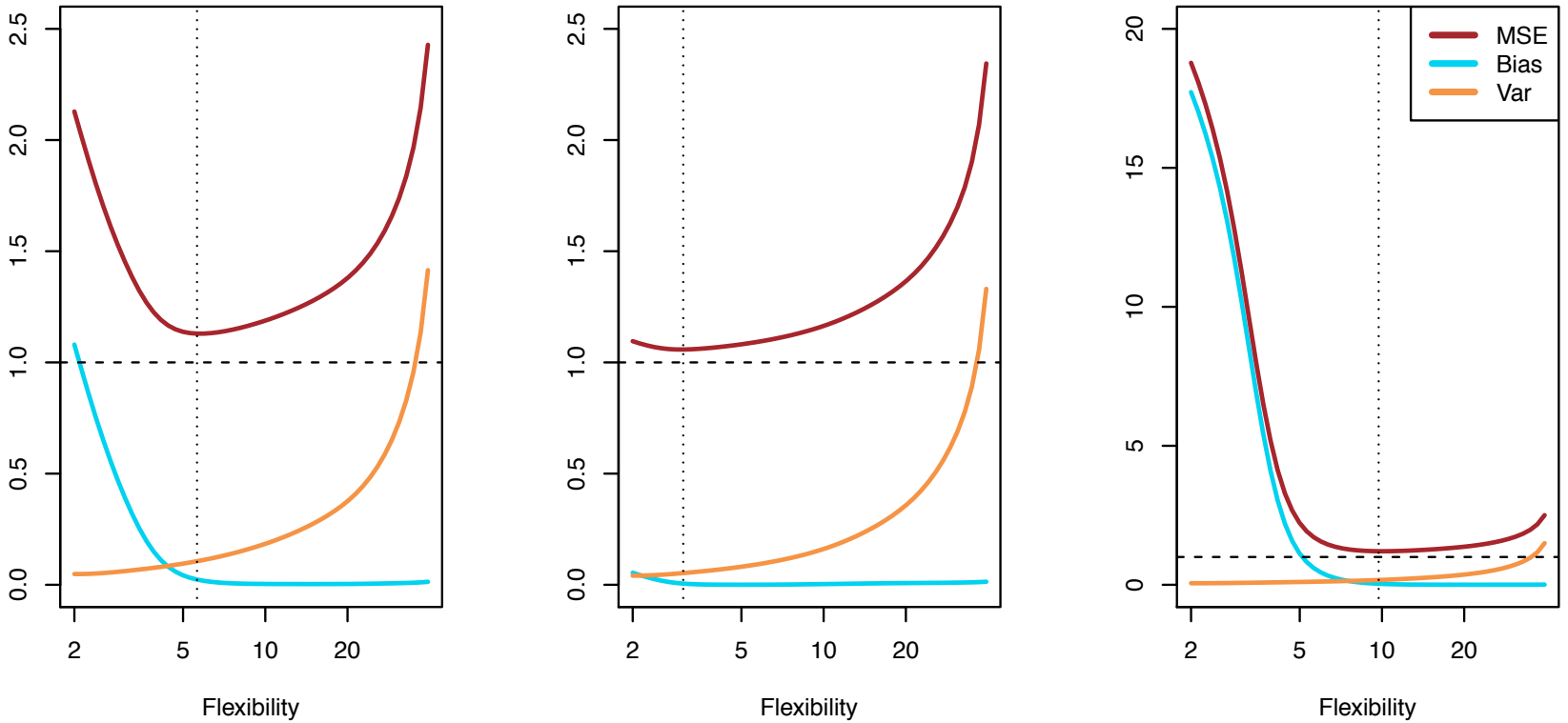
Typically as the *flexibility* of $\hat{f}$ increases, its variance increases, and its bias decreases. So choosing the flexibility based on average test error amounts to a *bias-variance trade-off*.

variance that comes from having different trainings sets. If I got a new training set and I fit my model again, I'd have a different function f hat.

bias is is the difference btw. the average prediction at x0 (averaged over all these different training sets), and the truth f(x0).

irreducible error that comes from the random variation in the new test point, y0, about the true function f.

Bias-variance trade-off for the three examples



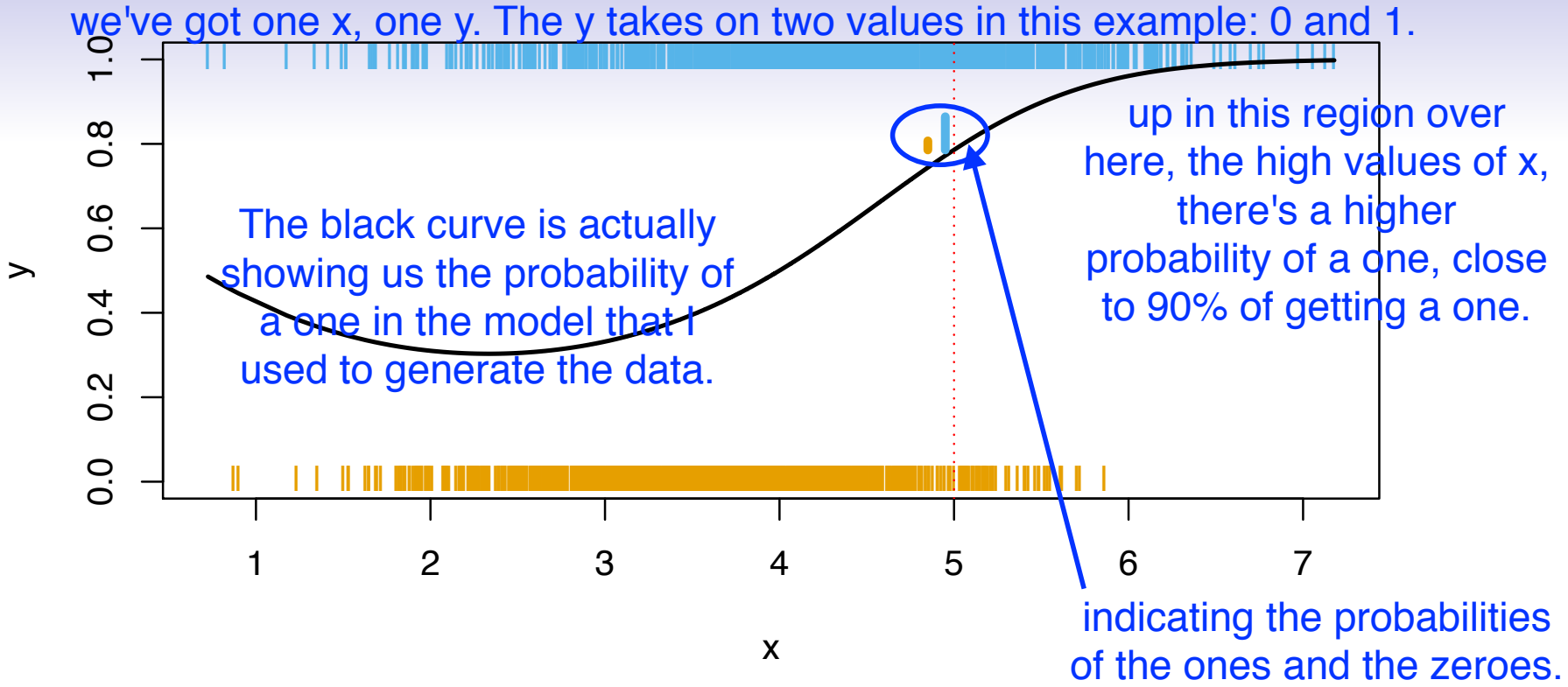
2.3 Classification

https://youtu.be/vVj2itVNku4

Classification Problems

Here the response variable Y is *qualitative* — e.g. email is one of $\mathcal{C} = (\texttt{spam}, \texttt{ham})$ ($\texttt{ham}$=good email), digit class is one of $\mathcal{C} = \{\texttt{0}, \texttt{1}, \ldots, \texttt{9}\}$. Our goals are to:

- Build a classifier $C(X)$ that assigns a class label from $\mathcal{C}$ to a future unlabeled observation X.
- Assess the uncertainty in each classification
- Understand the roles of the different predictors among $X = (X_1, X_2, \ldots, X_p)$.



Is there an ideal $C(X)$? Suppose the K elements in $\mathcal{C}$ are numbered $1, 2, \ldots, K$. Let

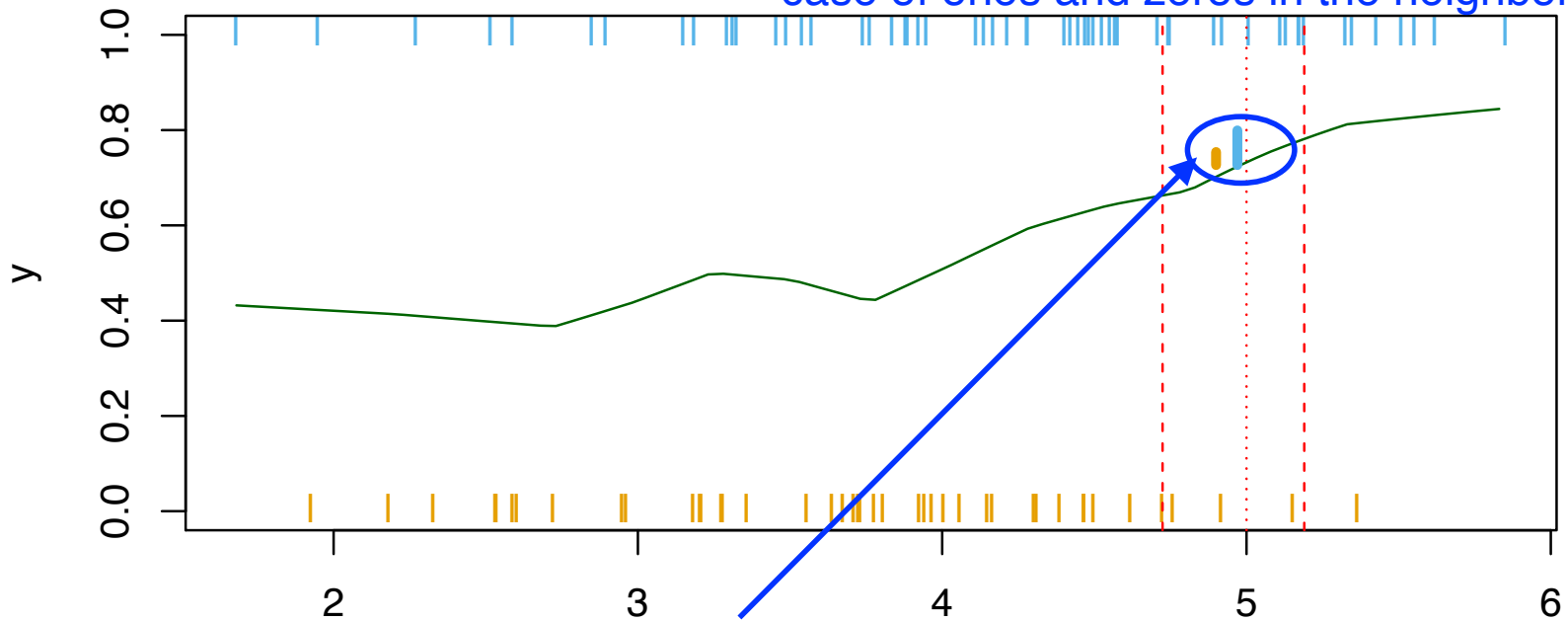
$$p_k(x) = \Pr(Y = k|X = x), \ k = 1, 2, \ldots, K.$$

These are the *conditional class probabilities* at x; e.g. see little barplot at $x = 5$. Then the *Bayes optimal* classifier at x is

$$C(x) = j \text{ if } p_j(x) = \max\{p_1(x), p_2(x), \ldots, p_K(x)\}$$

We've got 100 points having one of the two class labels (0 or 1)

So we send out a neighborhood, and gather, say, 10% of the data points. And then estimate the conditional probabilities by the proportions, in this case of ones and zeros in the neighborhood.



indicating the probabilities of the ones and the zeroes. By classifying to one over here, we are going to make mistakes on about 20% of the conditional population at this value of x. But we'll get it correct 80% of the time.

Nearest-neighbor averaging can be used as before.

Also breaks down as dimension grows. However, the impact on $\hat{C}(x)$ is less than on $\hat{p}_k(x),\ k = 1, \ldots, K$.

The high dimensional problem is worse for modeling the probabilities than it is for actually building the classifier. For the classifier, the classifier just has to be accurate with regard to which of the probabilities is largest. Whereas if we're really interested in the probabilities themselves, we going to be measuring them on a much finer scale.

Classification: some details

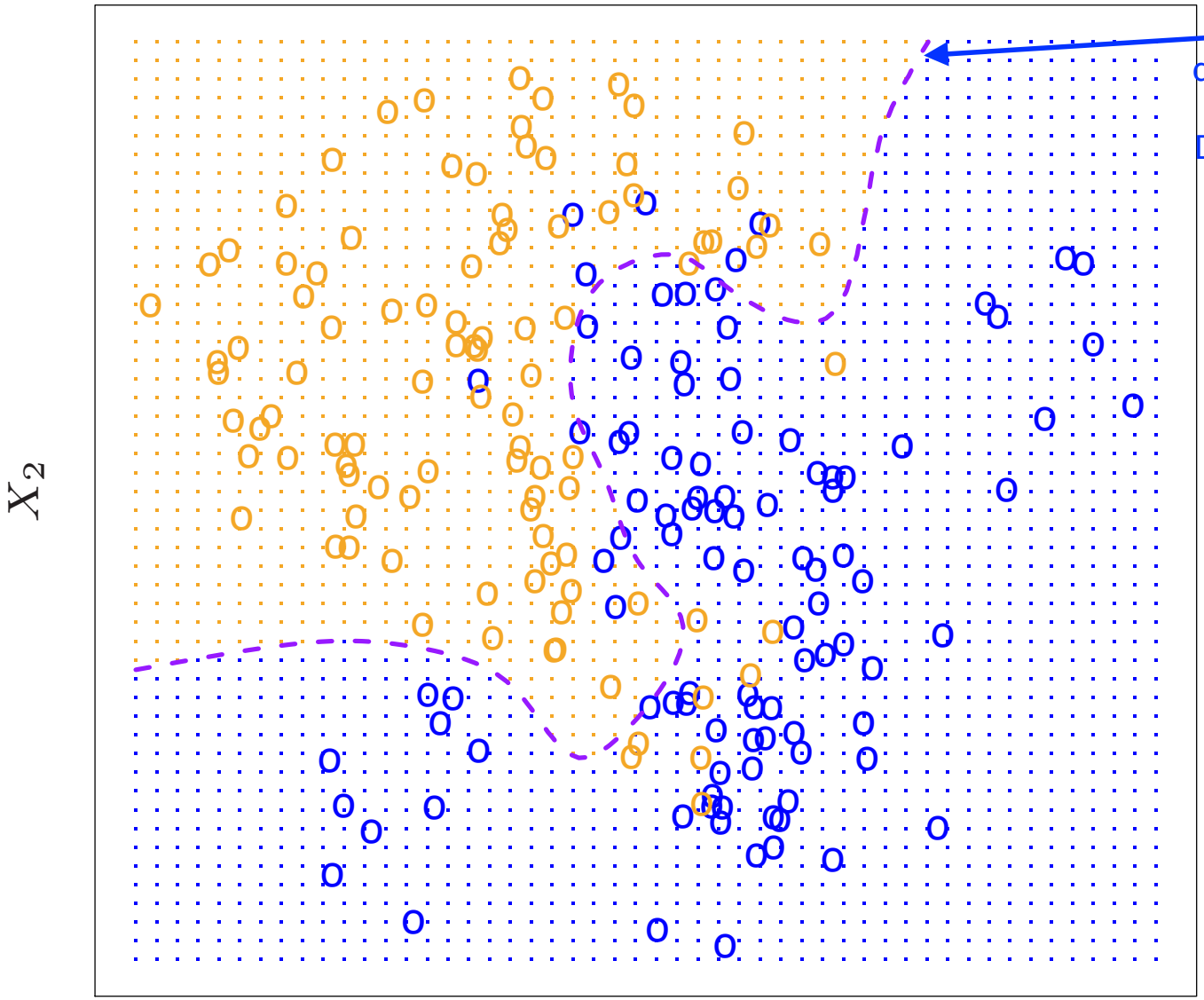
- Typically we measure the performance of $\hat{C}(x)$ using the misclassification error rate:

$$\text{Err}_{\mathsf{Te}} = \text{Ave}_{i \in \mathsf{Te}} I[y_i \neq \hat{C}(x_i)]$$

- The Bayes classifier (using the true $p_k(x)$) has smallest error (in the population).

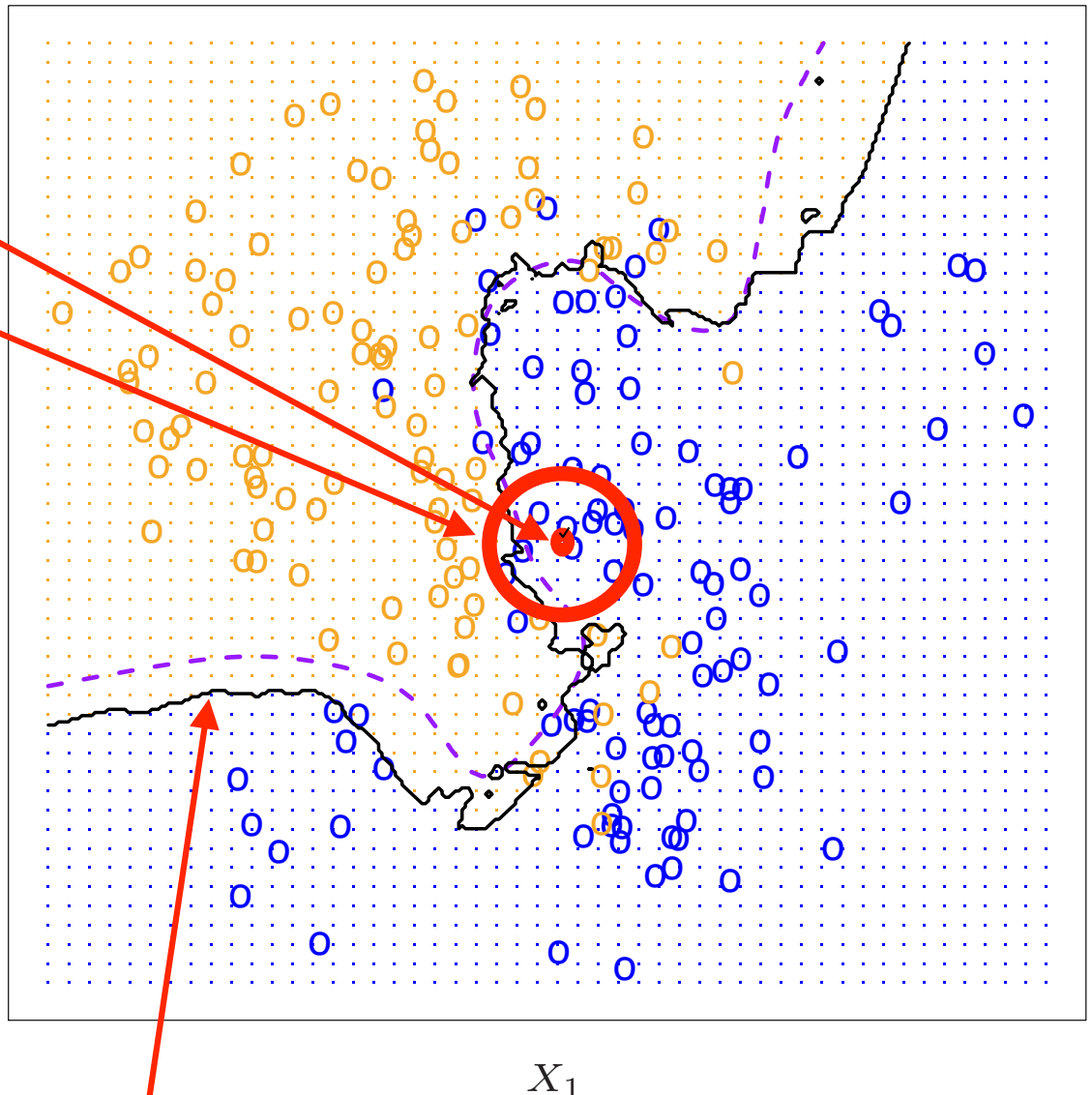
- Support-vector machines build structured models for $C(x)$.
- We will also build structured models for representing the $p_k(x)$. e.g. Logistic regression, generalized additive models.

Example: K-nearest neighbors in two dimensions



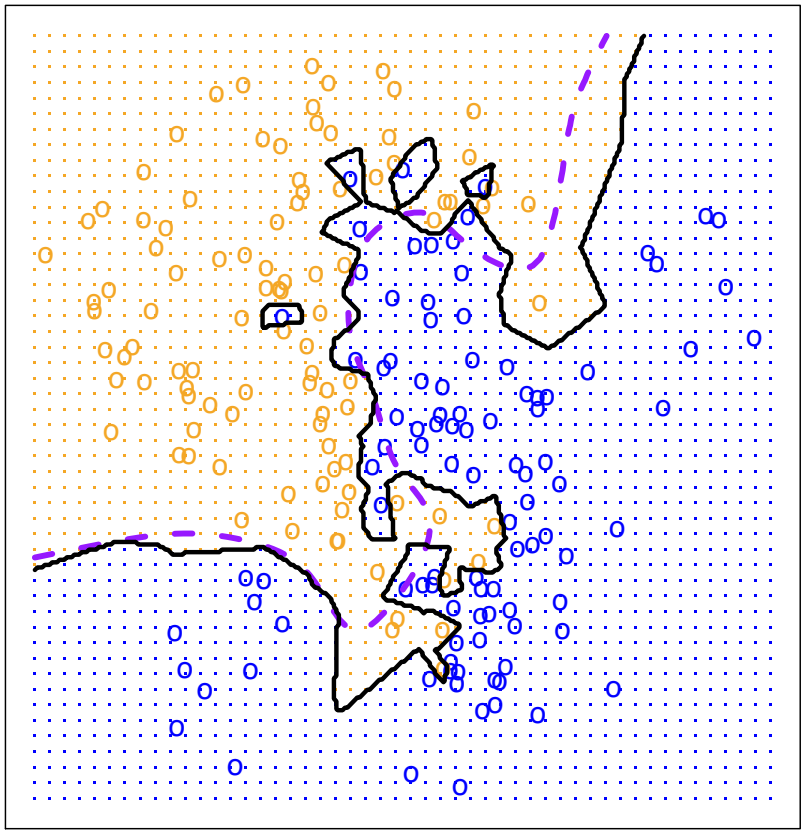
KNN: K=10

let's say we pick this point over here, we spread out a little neighborhood, in this case, until we find the 10 closest points to the target point. And we'll estimate the probability at this center point here by the proportion of blues versus oranges. And you do that at every point.

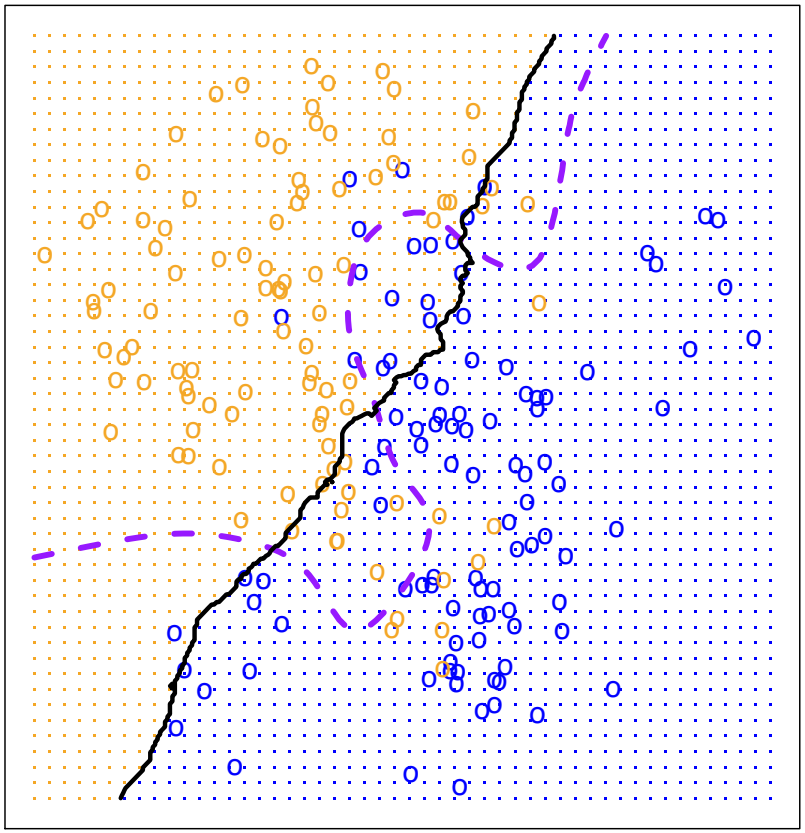


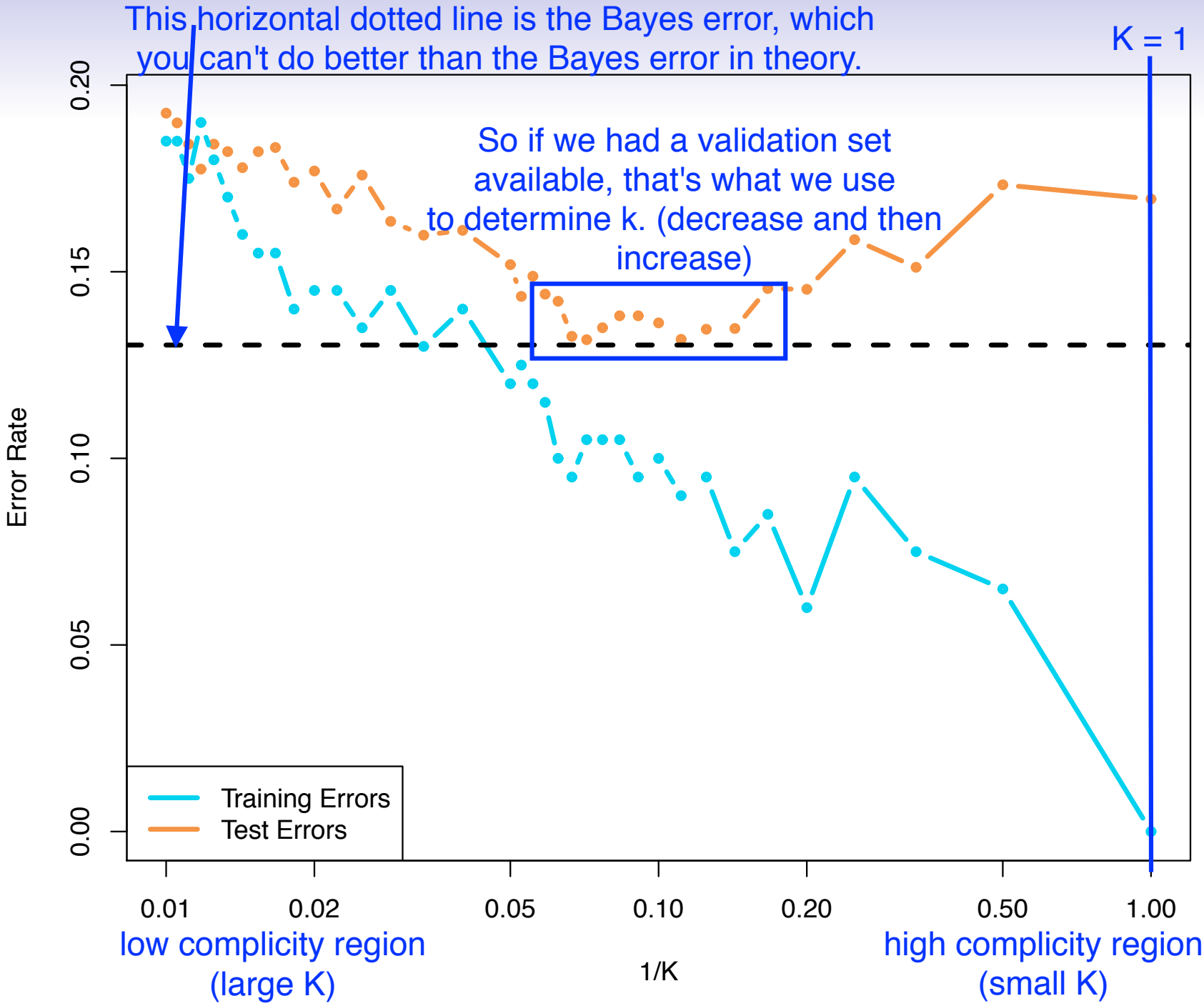
then you get this estimated decision boundary.

KNN: K=1



KNN: K=100




This horizontal dotted line is the Bayes error, which you can't do better than the Bayes error in theory.
K = 1
So if we had a validation set available, that's what we use to determine k. (decrease and then increase)
Error Rate
0.00
0.05
0.10
0.15
0.20
Training Errors
Test Errors
0.01
0.02
0.05
0.10
0.20
0.50
1.00
1/K
low complicity region (large K)
high complicity region (small K)