Ch 2 Statistical Learning

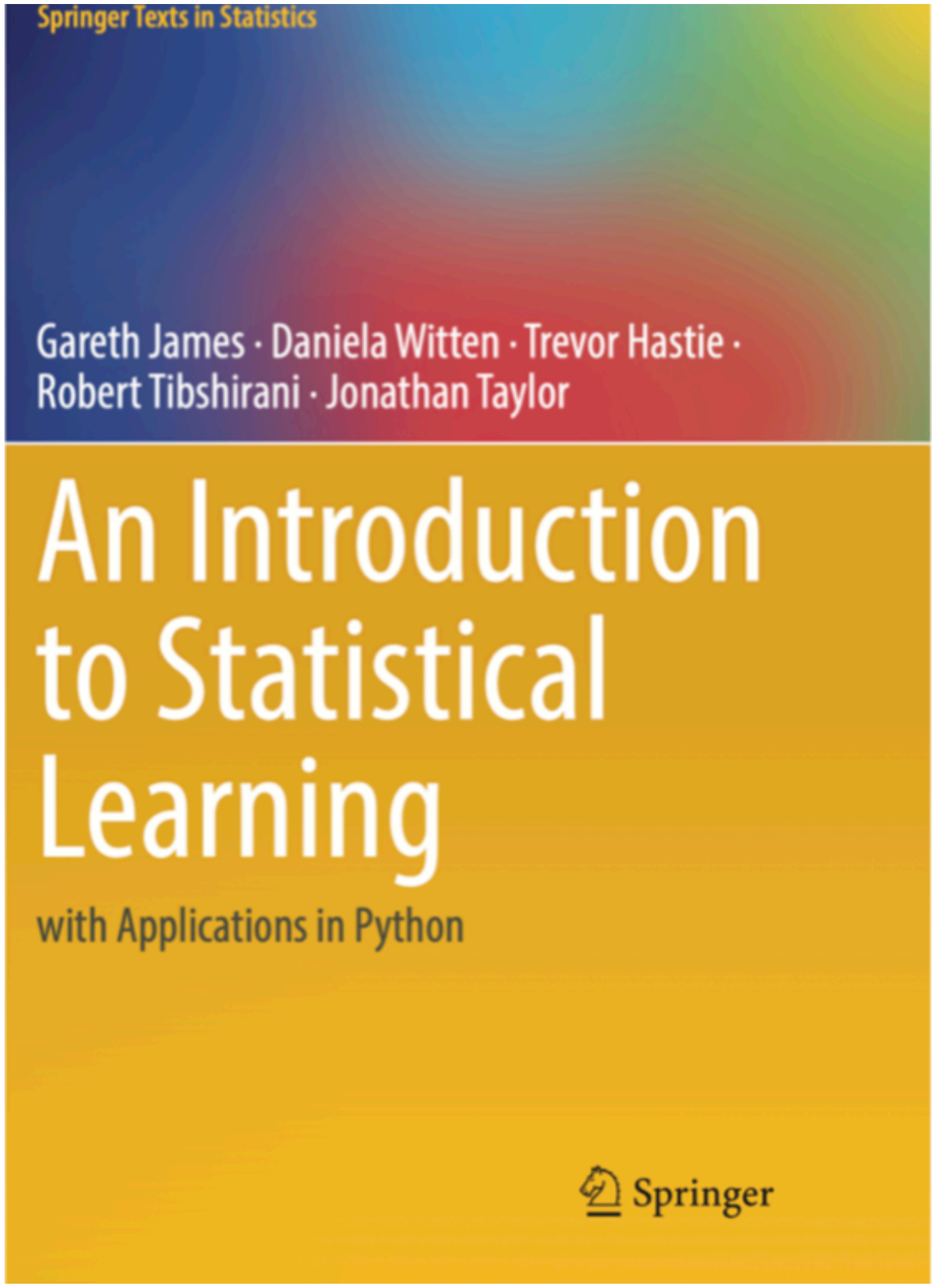
Instructor: Huei-Wen Teng

Slides modified from https://www.statlearning.com/

September 22, 2025

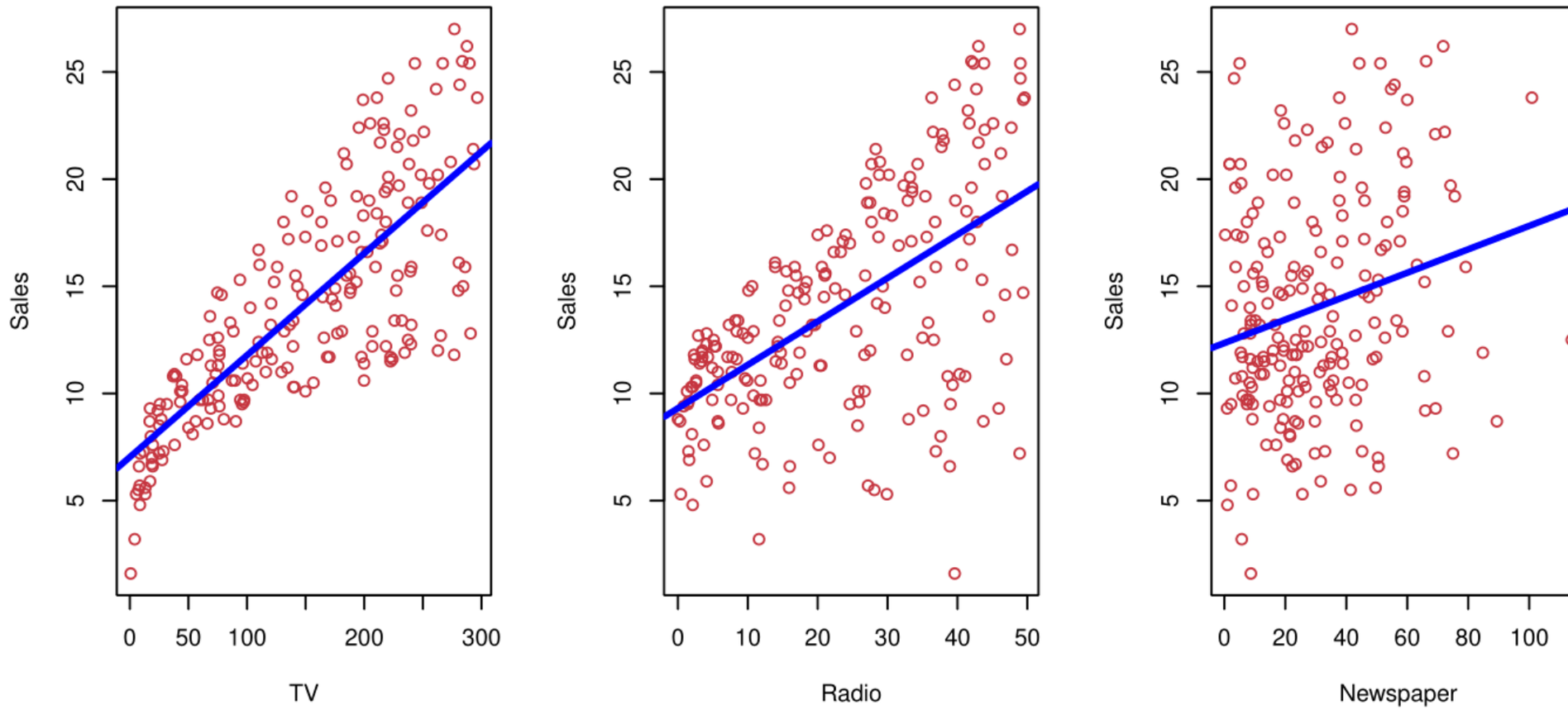
2.1

What is statistical learning?



https://www.statlearning.com/

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

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

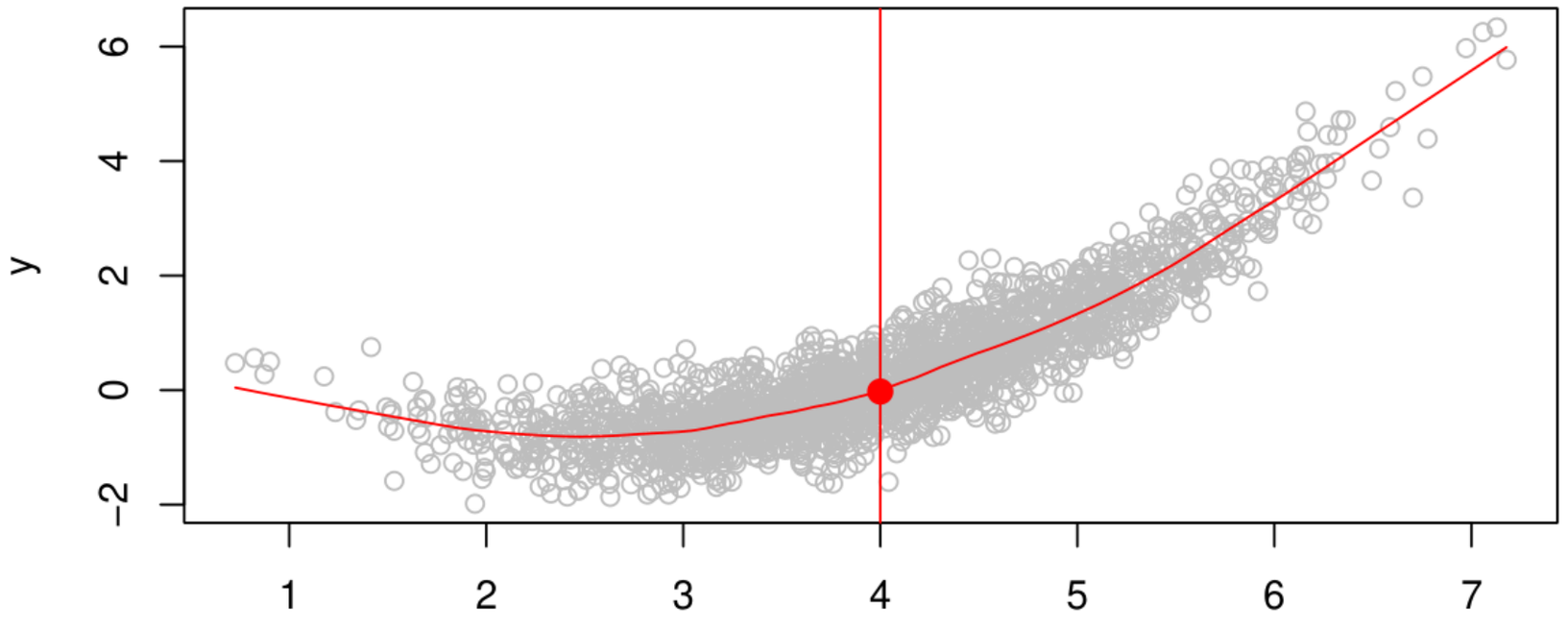
Now we write our model as

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

where ϵ captures measurement errors and other discrepancies.

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

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

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

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

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

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

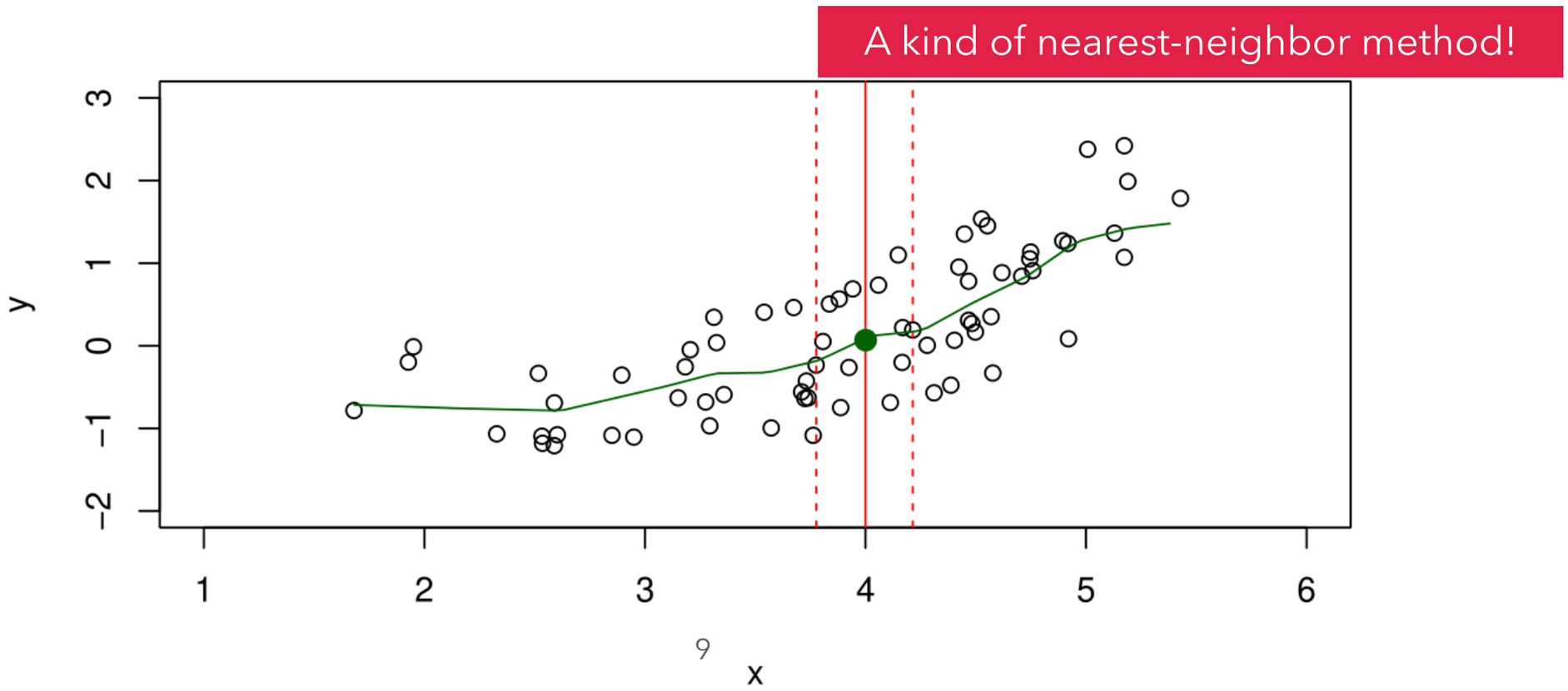
- Random error ϵ
  - $E[\epsilon] = 0$
  - $Var(\epsilon) = \sigma^2$
  - $Var(\epsilon) = E(\epsilon - \mu_\epsilon)^2 = E(\epsilon^2)$

How to estimate f

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

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

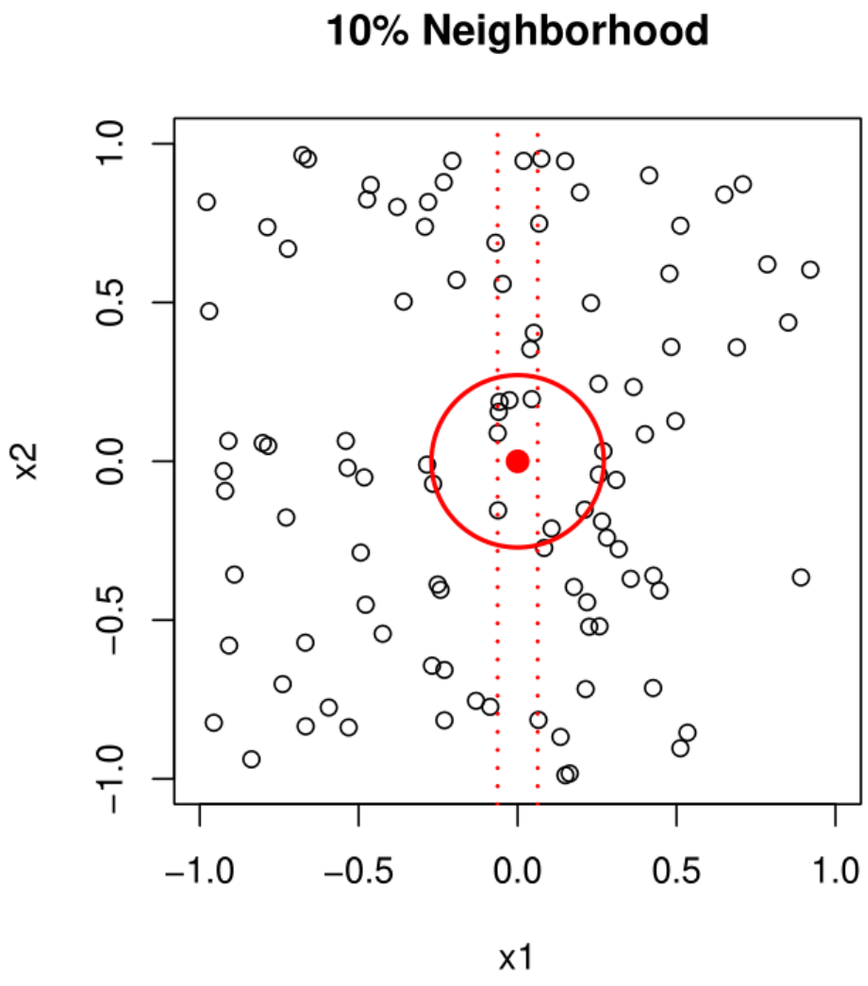
where $\mathcal{N}(x)$ is some *neighborhood* of x.

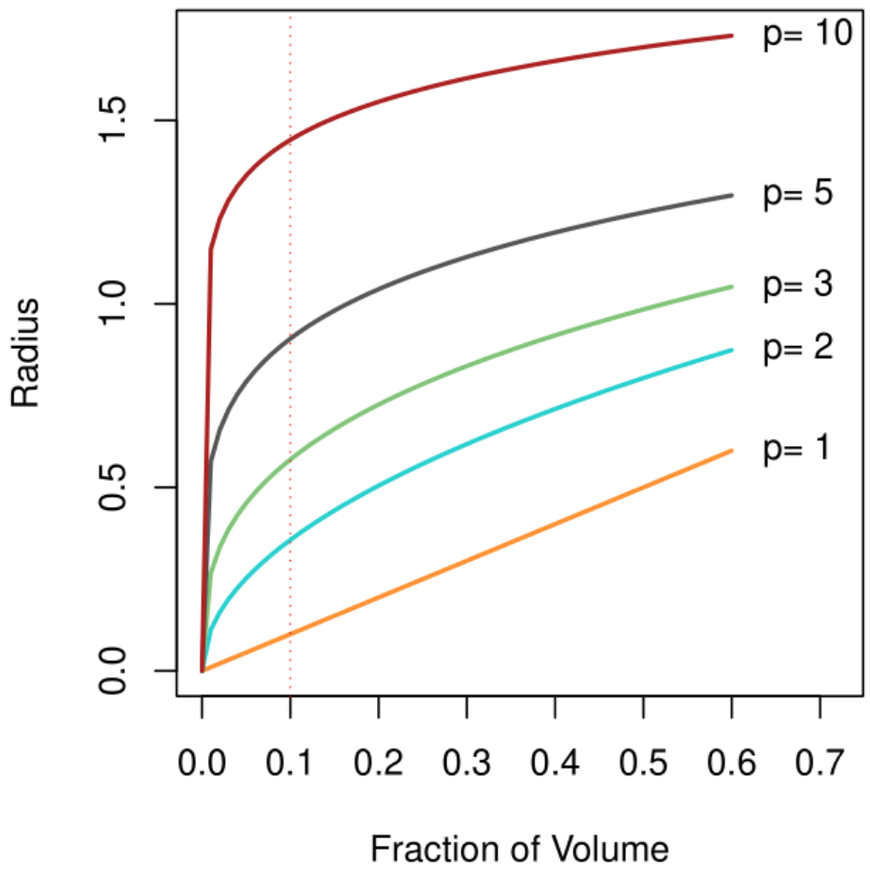


- Nearest neighbor averaging can be pretty good for small p — i.e. $p \leq 4$ and large-ish N.
- We will discuss smoother versions, such as kernel and spline smoothing later in the course.
- Nearest neighbor methods can be *lousy* when p is large. Reason: the *curse of dimensionality*. Nearest neighbors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

  - 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





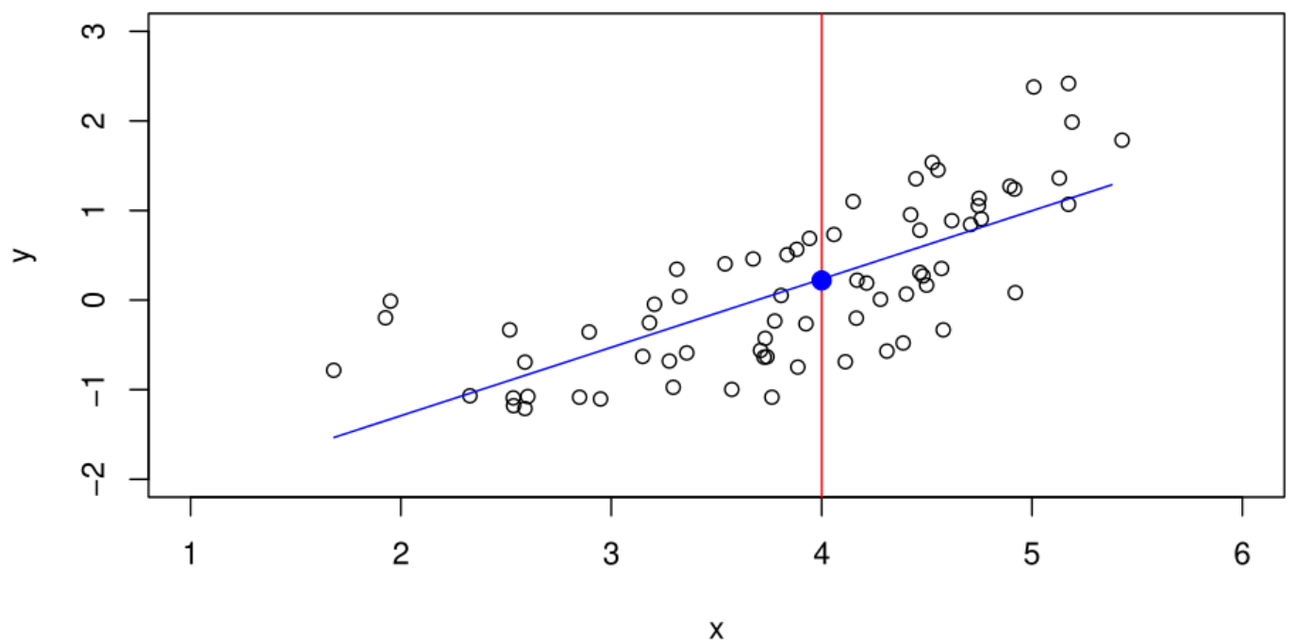
Parametric and structured models

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

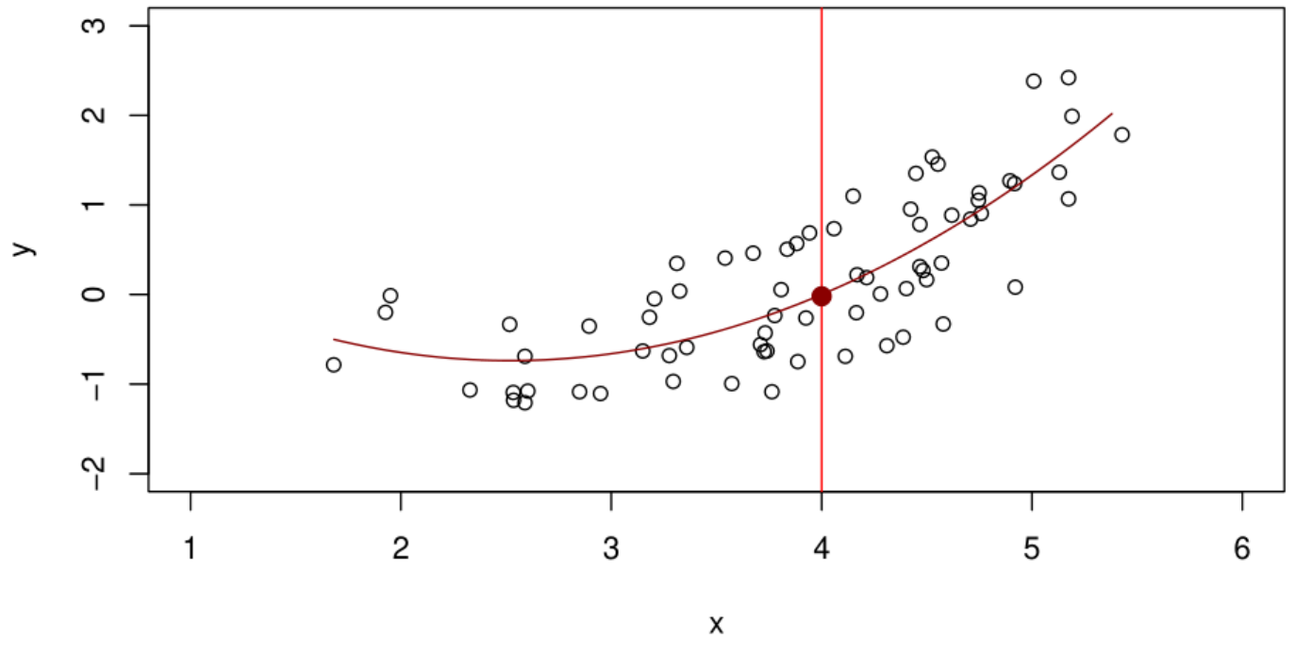
$$f_L(X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots \beta_p X_p.$$

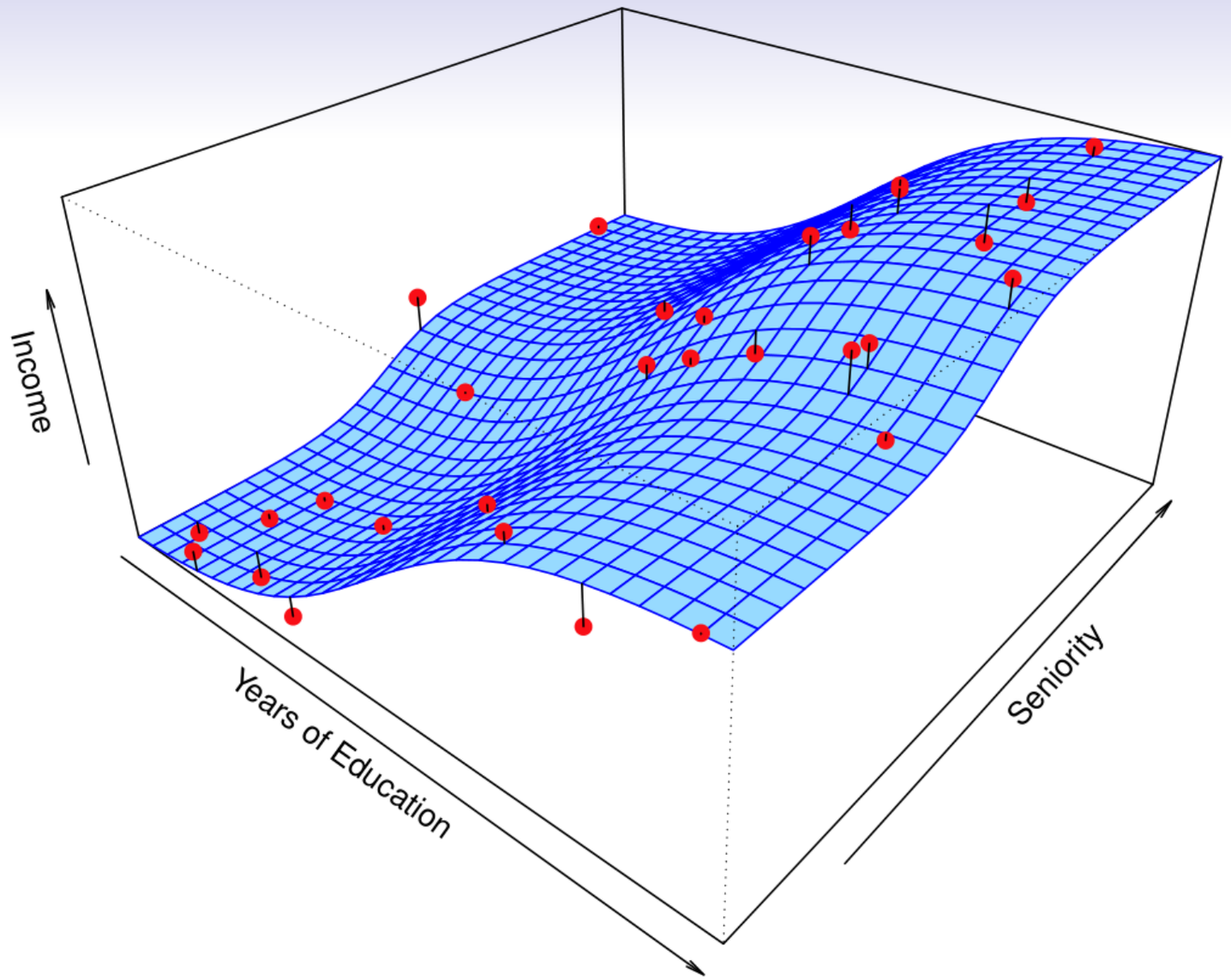
- A linear model is specified in terms of $p + 1$ parameters $\beta_0, \beta_1, \ldots, \beta_p$.
- We estimate the parameters by fitting the model to training data.
- Although it is *almost never correct*, a linear model often serves as a good and interpretable approximation to the unknown true function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

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.

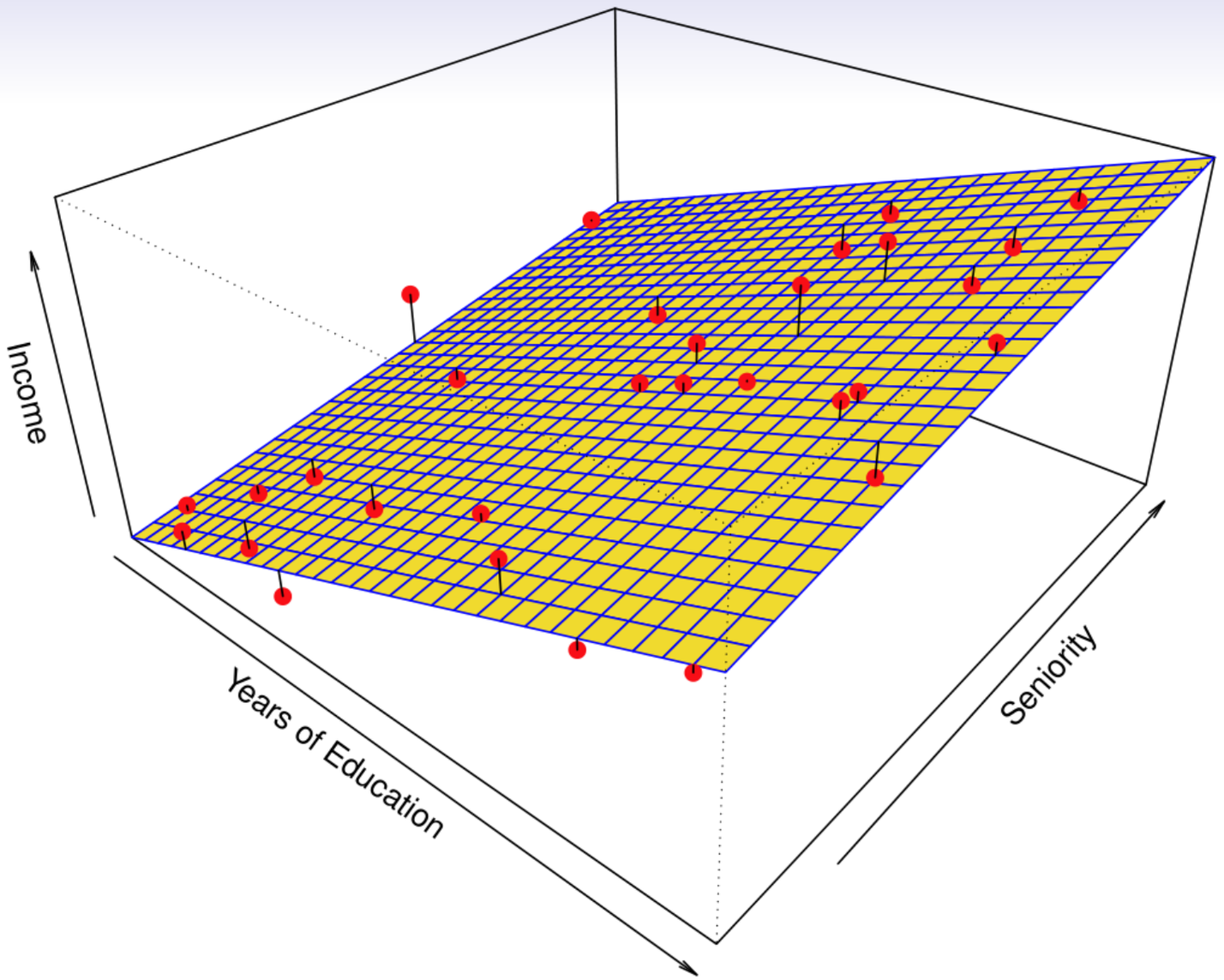




Simulated example. Red points are simulated values for `income` from the model

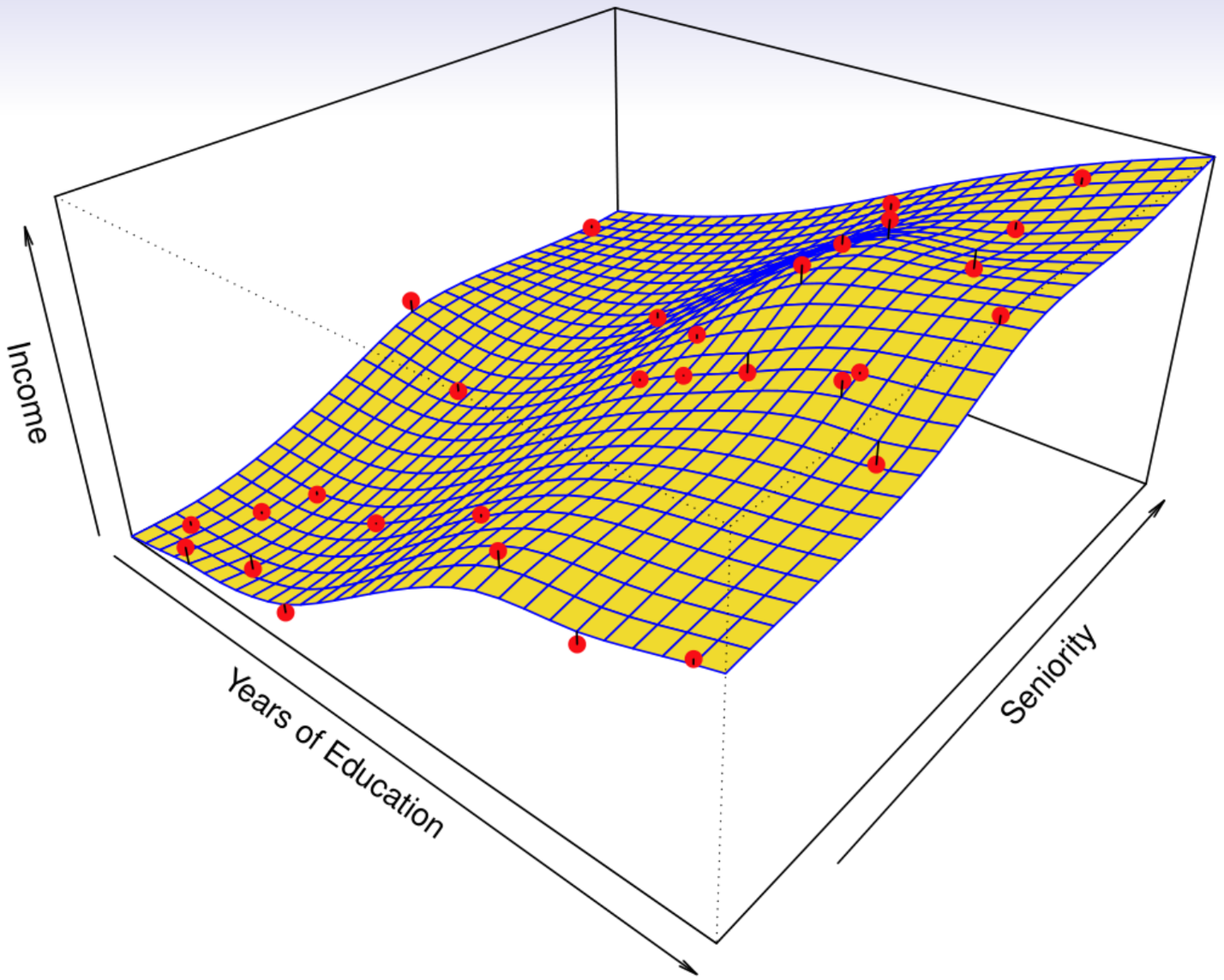
$$\texttt{income} = f(\texttt{education}, \texttt{seniority}) + \epsilon$$

f is the blue surface.

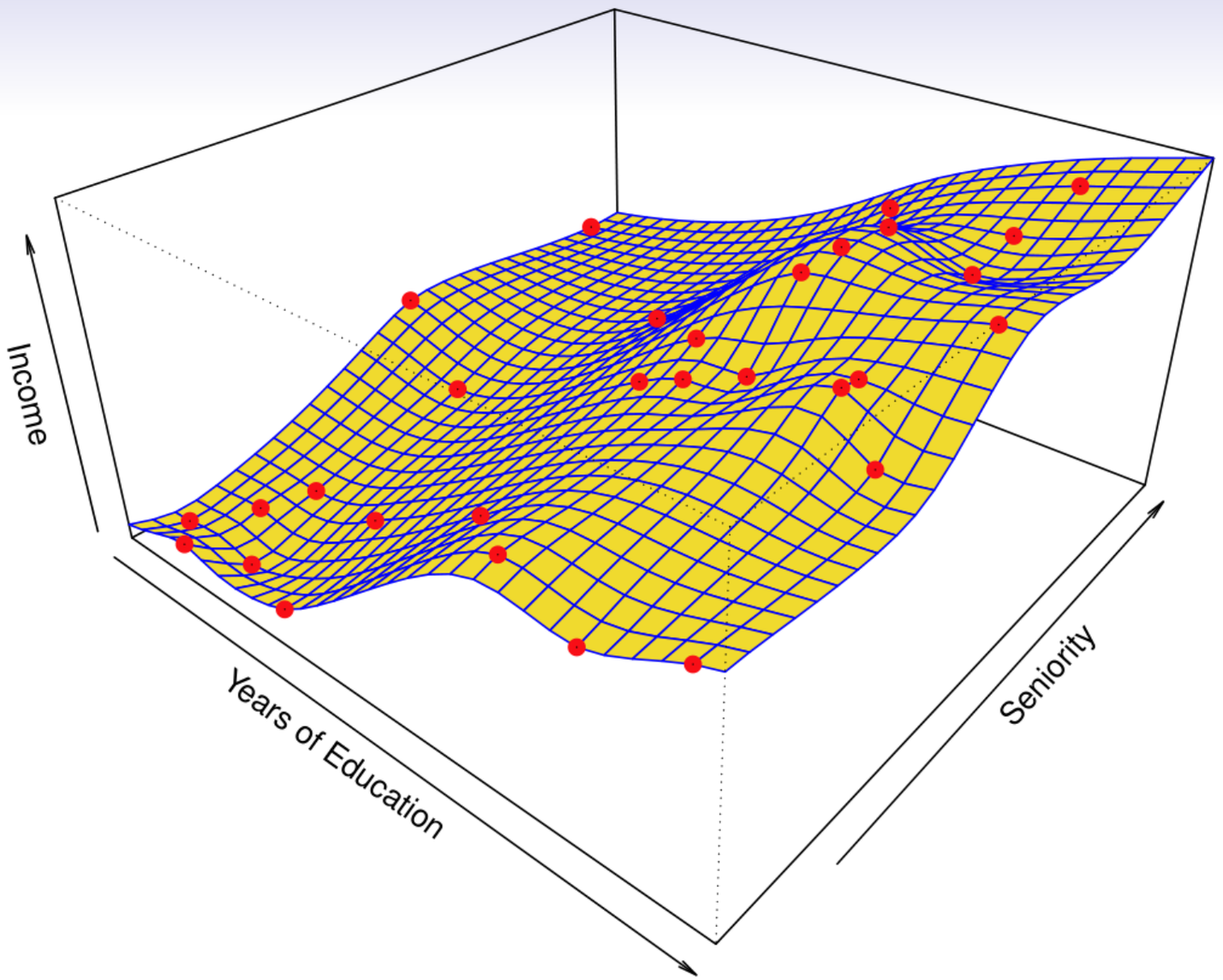


Linear regression model fit to the simulated data.

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



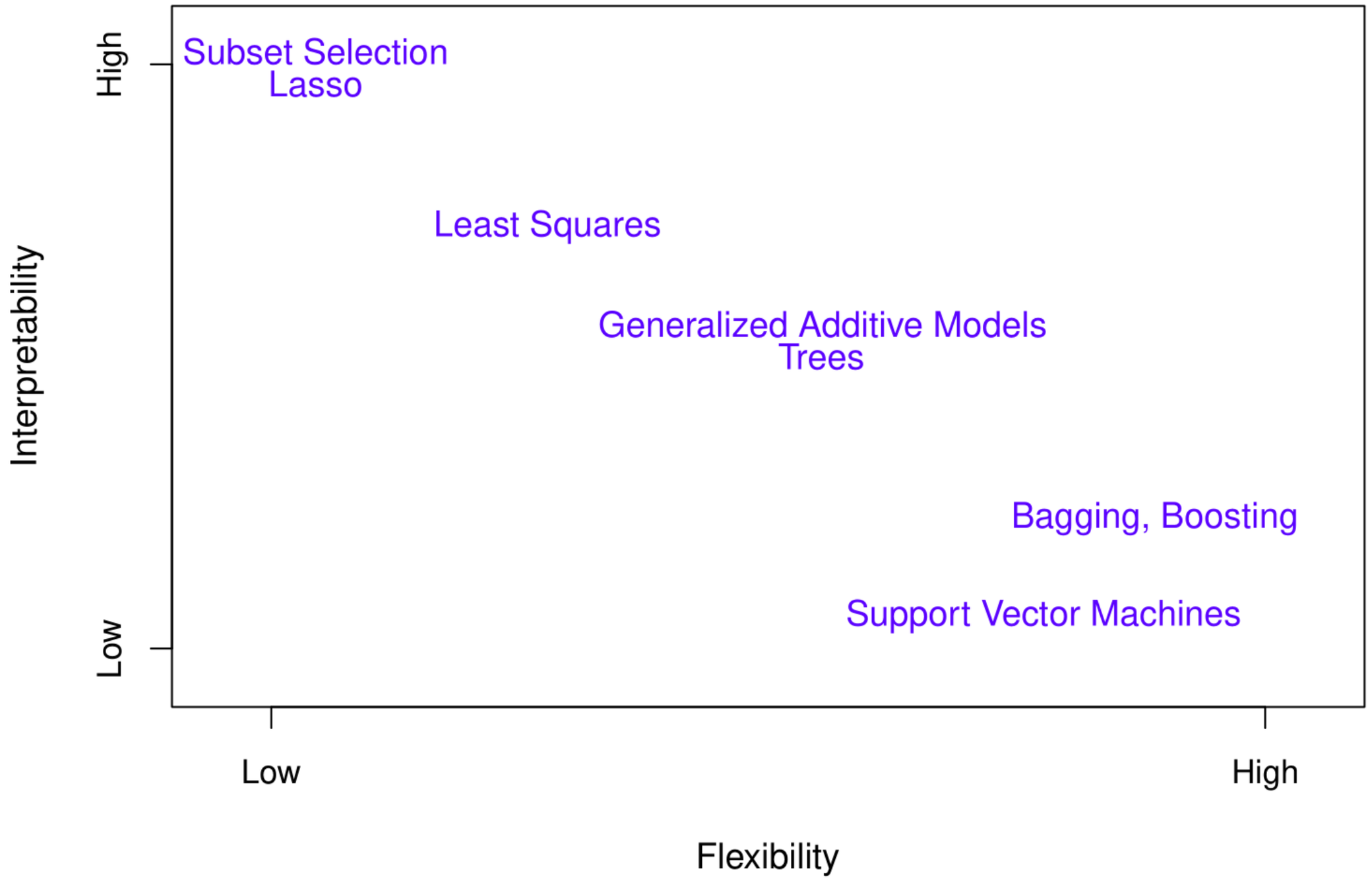
More flexible regression model $\hat{f}_S(\texttt{education}, \texttt{seniority})$ fit to the simulated data. Here we use a technique called a *thin-plate spline* to fit a flexible surface. We control the roughness of the fit (chapter 7).



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*.

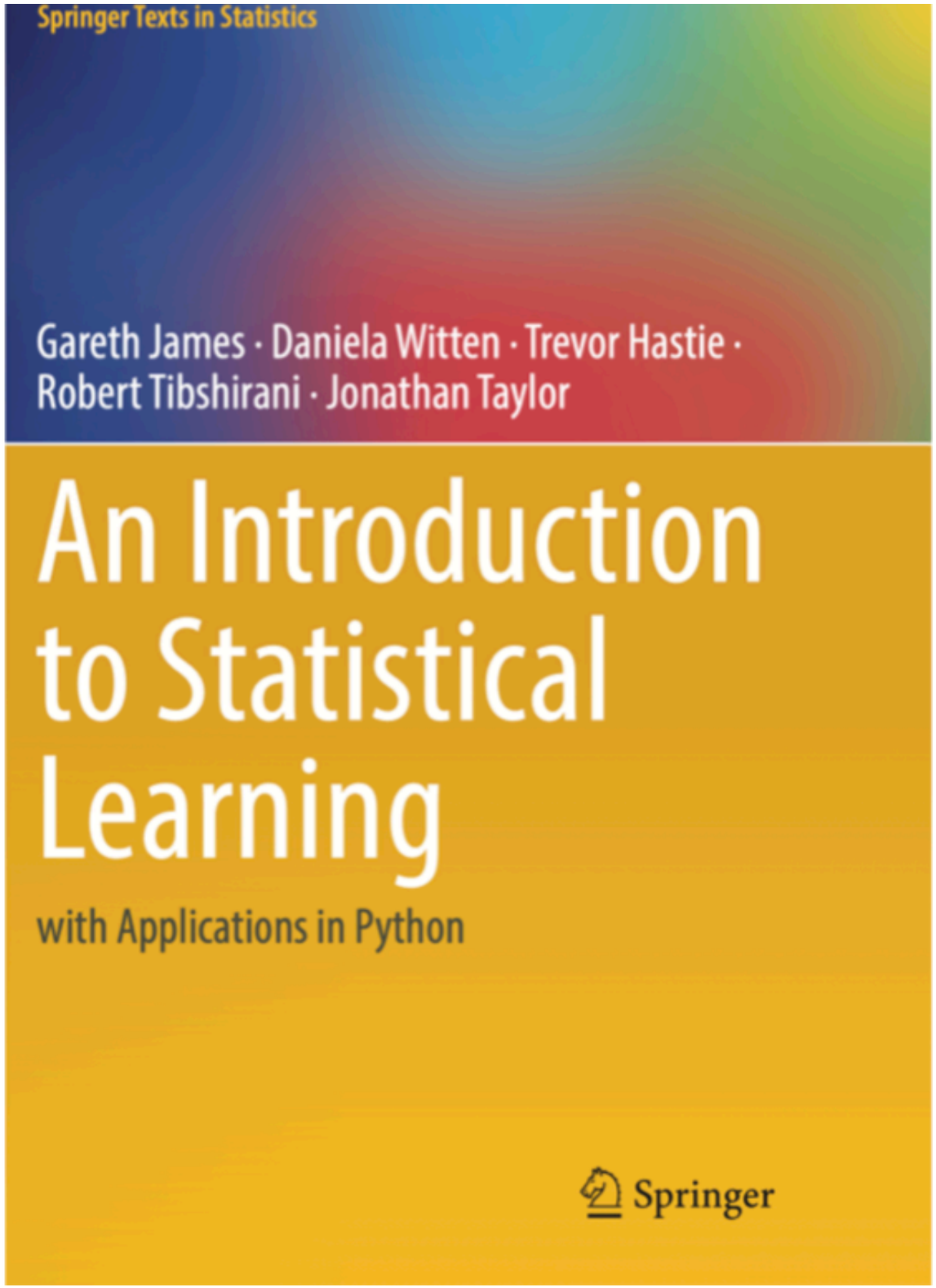
Some trade-offs

- 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

2.2

Assessing Model Accuracy



https://www.statlearning.com/

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.

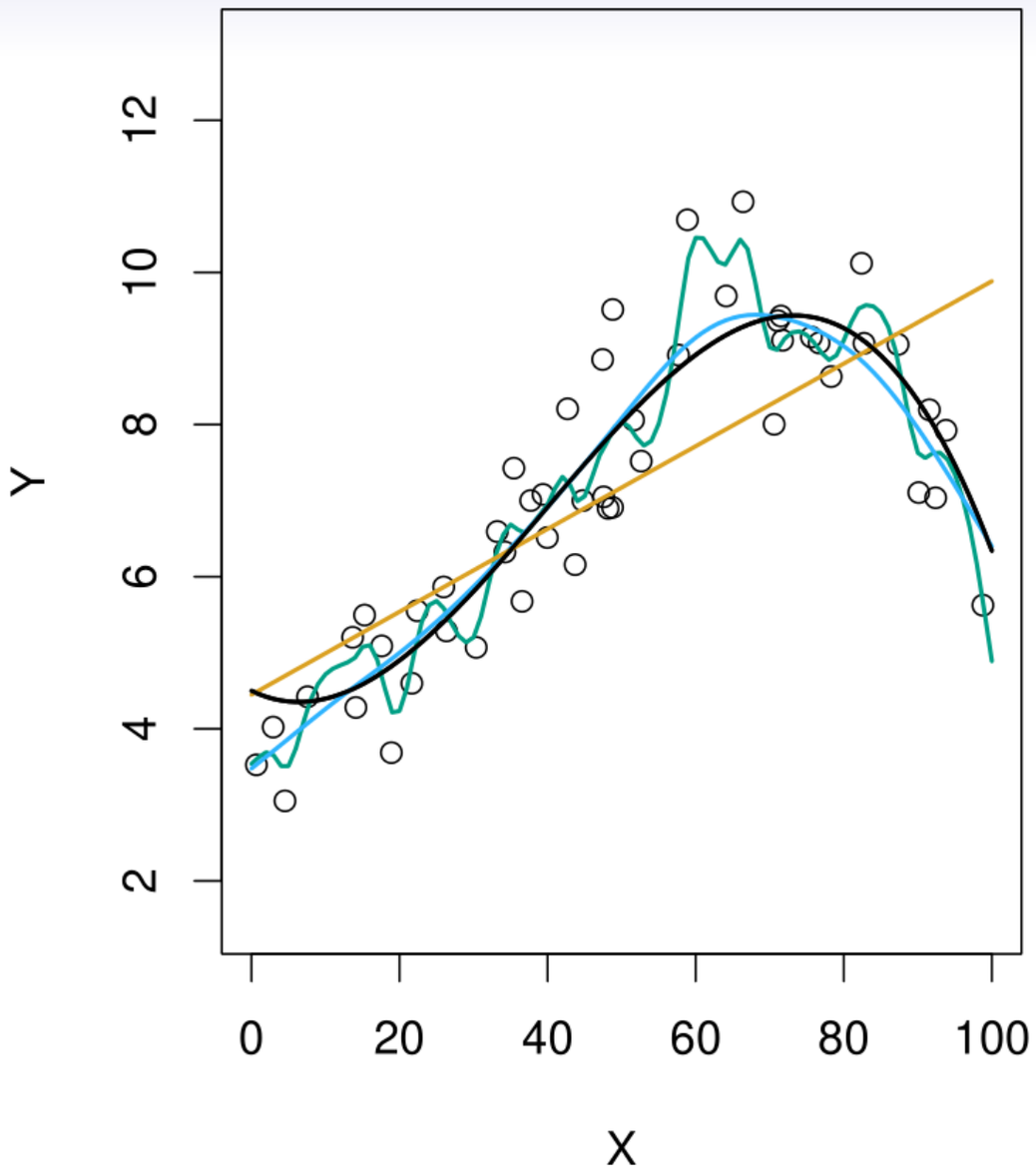
- We could compute the average squared prediction error over Tr:

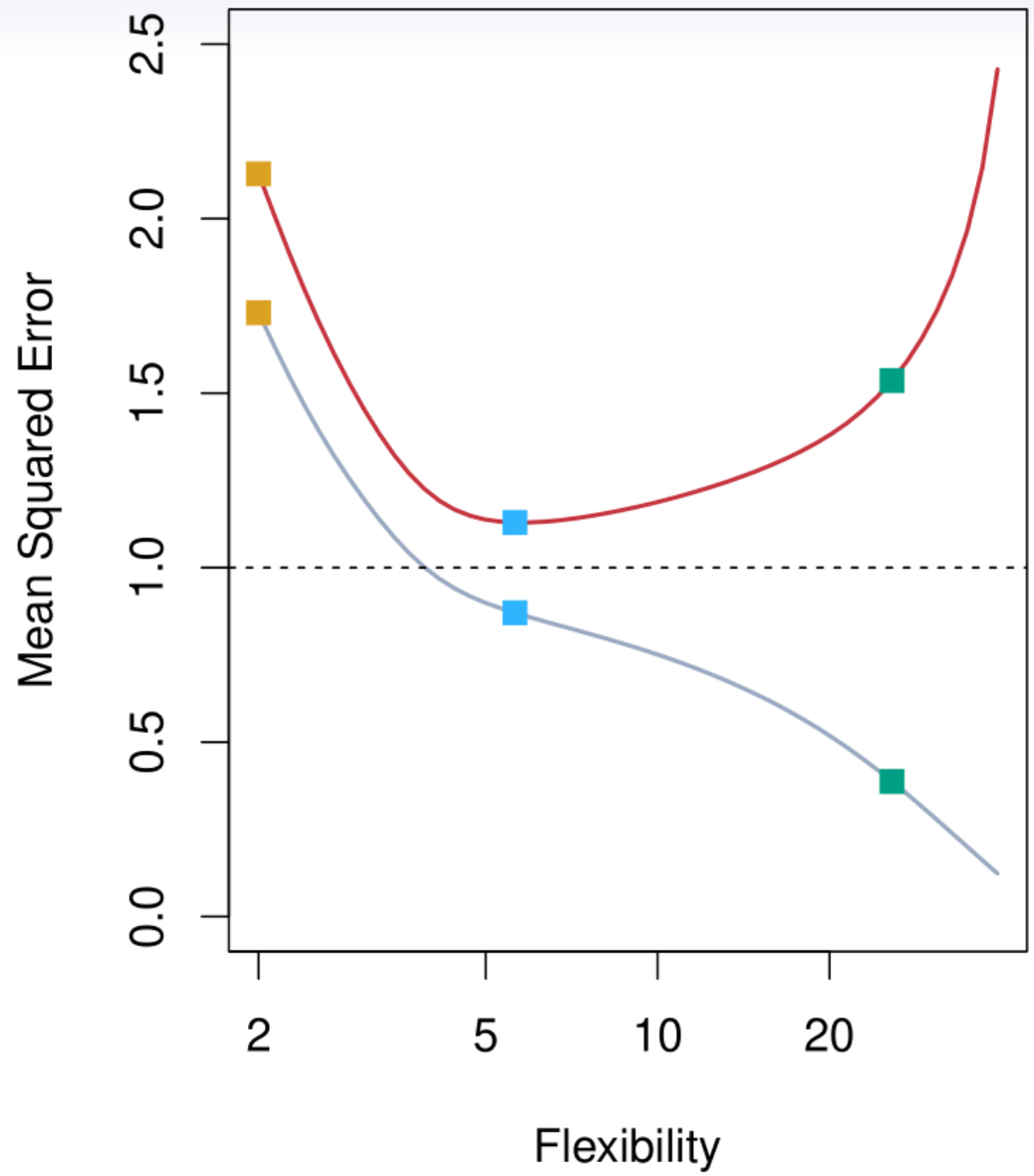
$$\text{MSE}_{\mathsf{Tr}} = \text{Ave}_{i \in \mathsf{Tr}}[y_i - \hat{f}(x_i)]^2$$

This may be biased toward more overfit models.

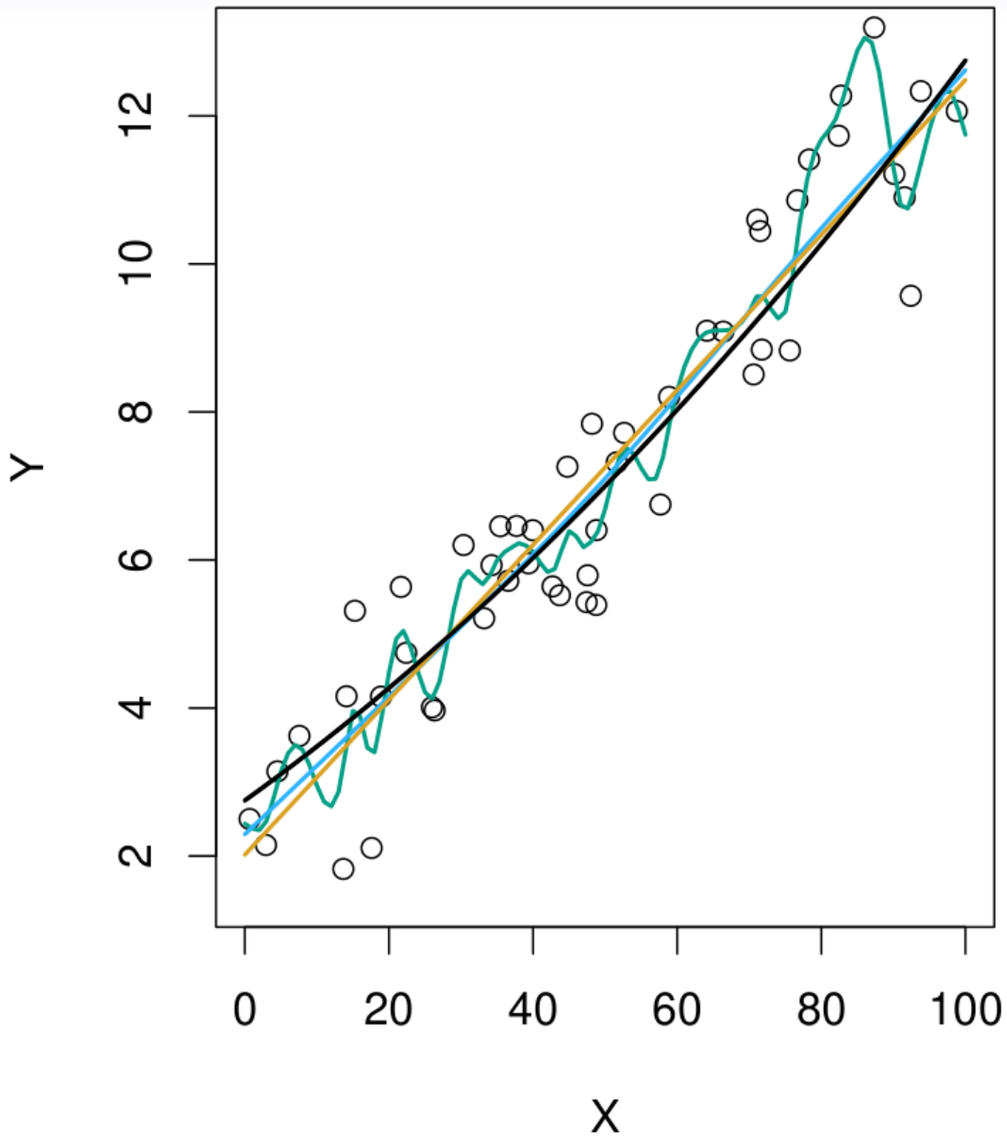
- Instead we should, if possible, compute it using fresh *test* data $\mathsf{Te} = \{x_i, y_i\}_1^M$:

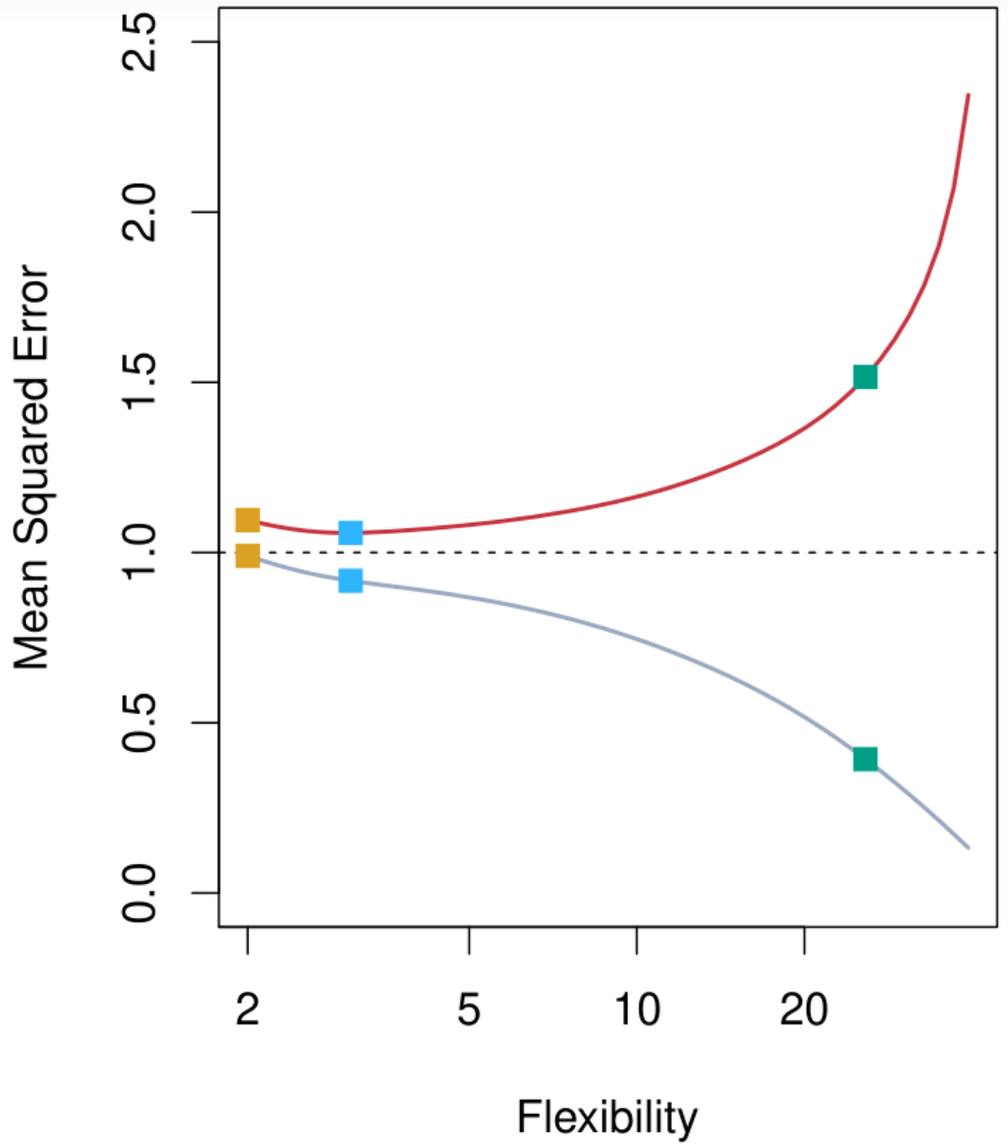
$$\text{MSE}_{\mathsf{Te}} = \text{Ave}_{i \in \mathsf{Te}}[y_i - \hat{f}(x_i)]^2$$



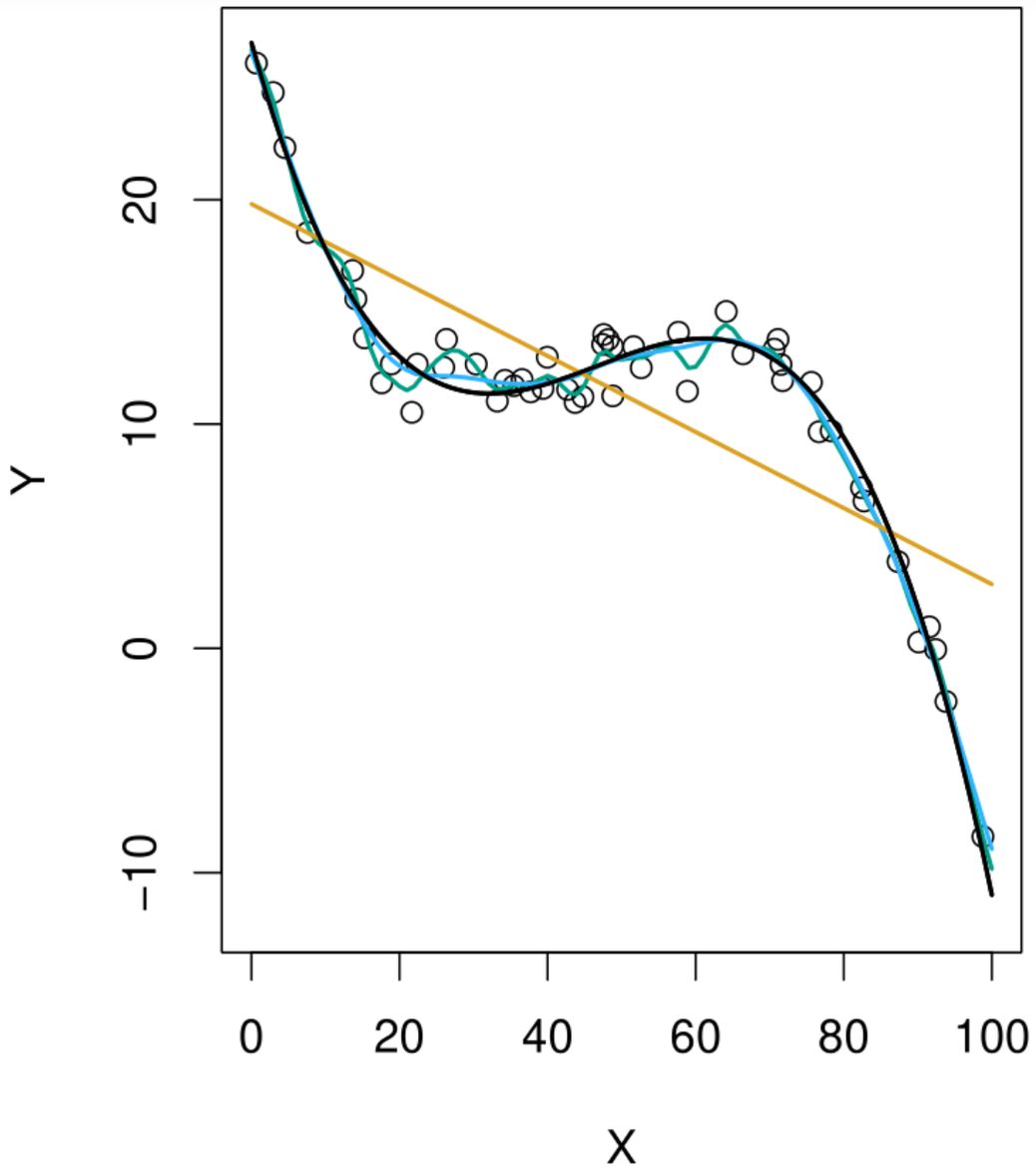


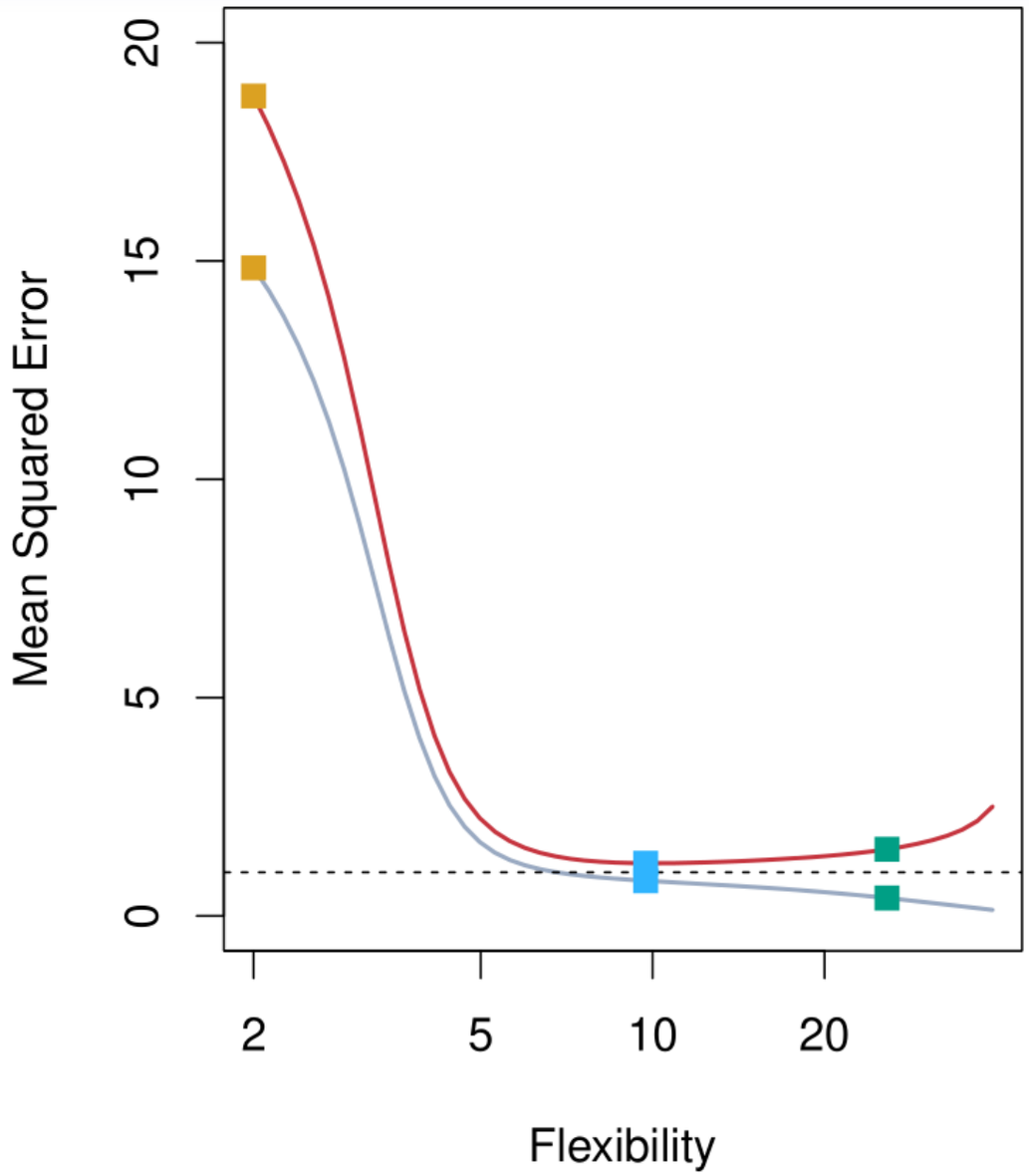
Black curve is truth. Red curve on right is MSE_{Te}, grey curve is MSE_{Tr}. Orange, blue and green curves/squares correspond to fits of different flexibility.





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

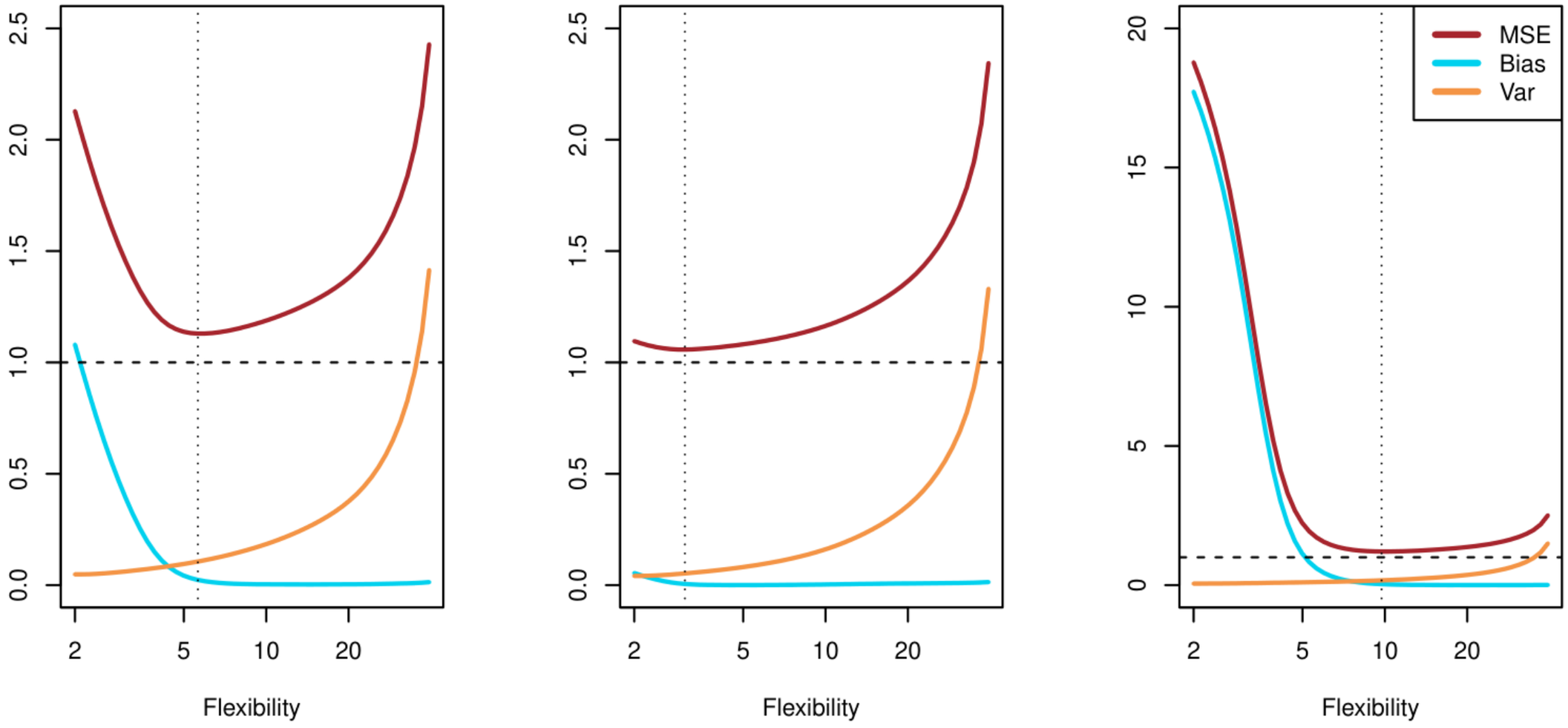
Suppose we have fit a model $\hat{f}(x)$ to some training data Tr, and let (x_0, y_0) be a test observation drawn from the population. If the true model is $Y = f(X) + \epsilon$ (with $f(x) = E(Y|X = x)$), then

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

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.*

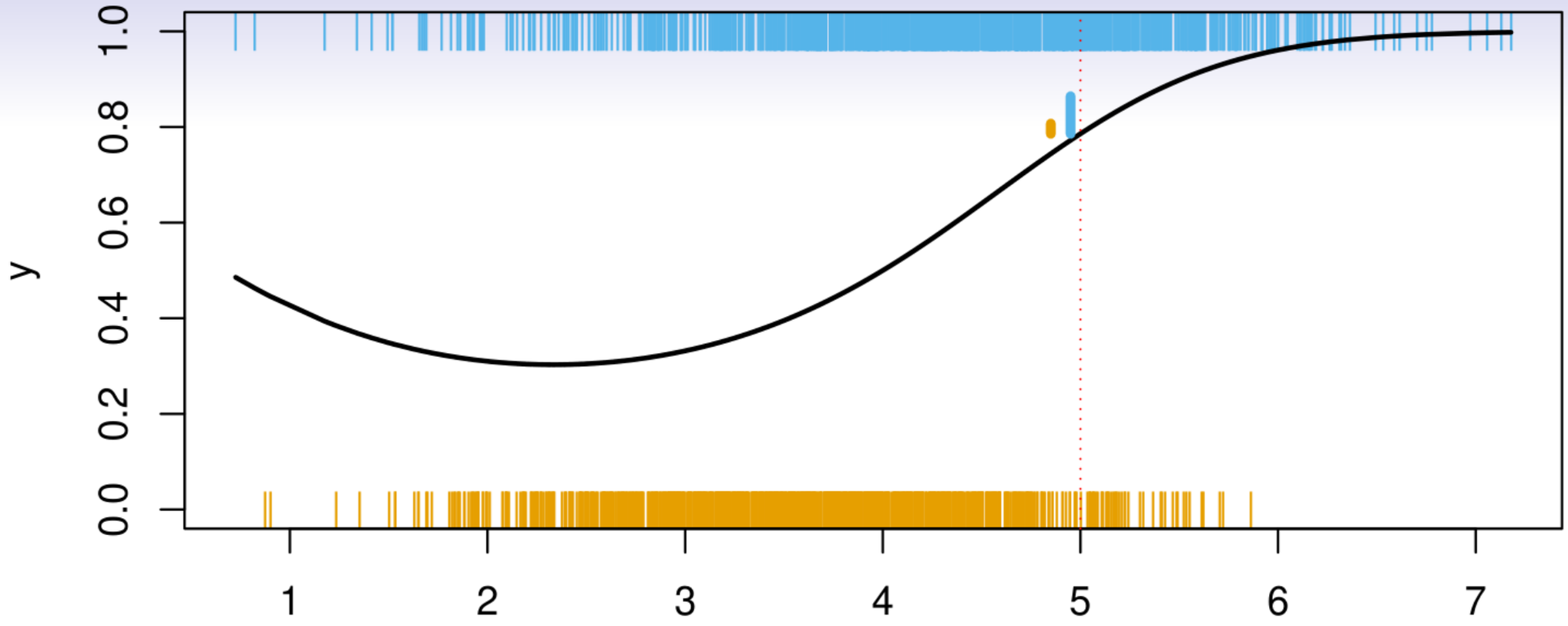
Bias-variance trade-off for the three examples



Classification Problems

Here the response variable Y is *qualitative* — e.g. email is one of $\mathcal{C} = (\texttt{spam}, \texttt{ham})$ (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 $X = (X_1, X_2, \ldots, X_p)$.

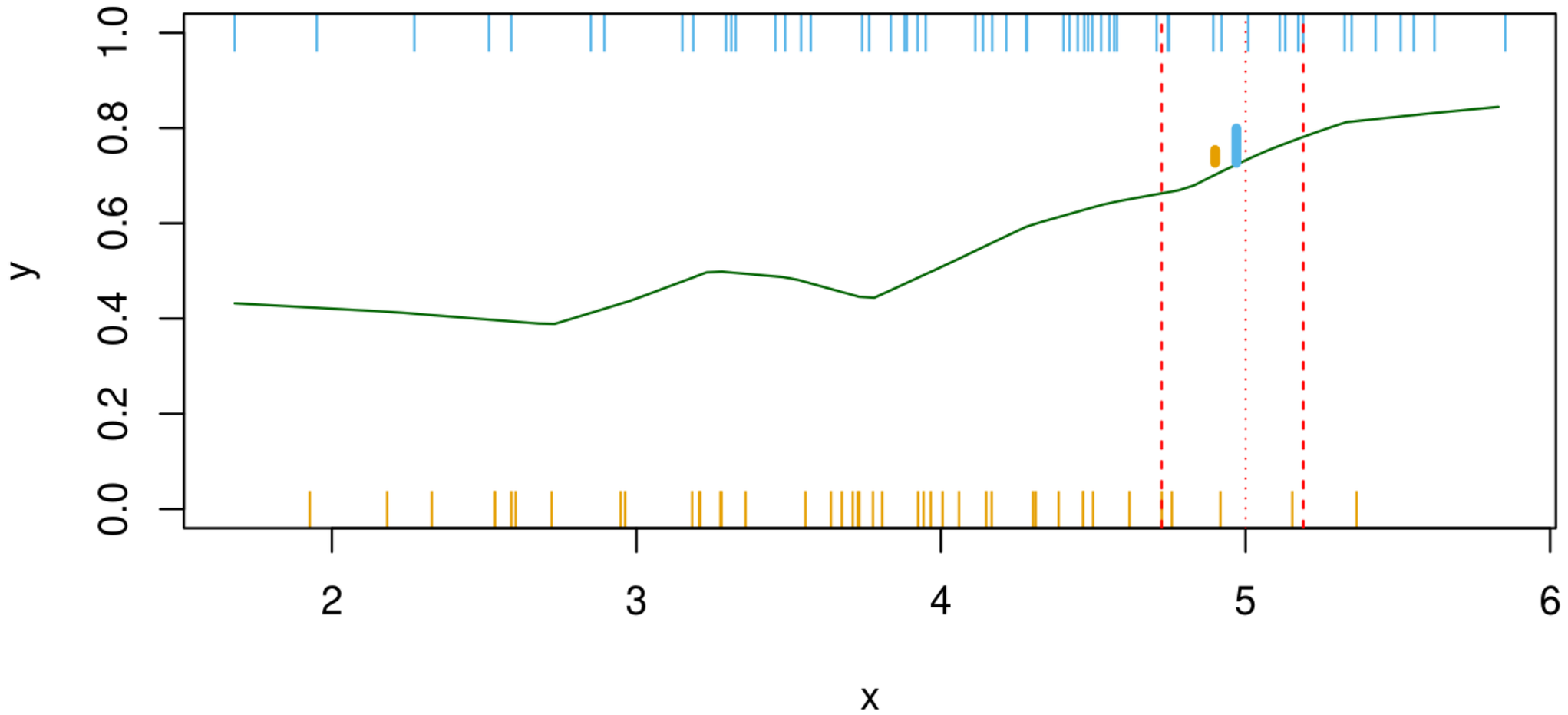


Is there an ideal $C(X)$? Suppose the K elements in $\mathcal{C}$ are numbered $1, 2, \ldots, K$. Let

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

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

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



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 $\hat{p}_k(x),\ k = 1, \ldots, K$.

Classification: some details

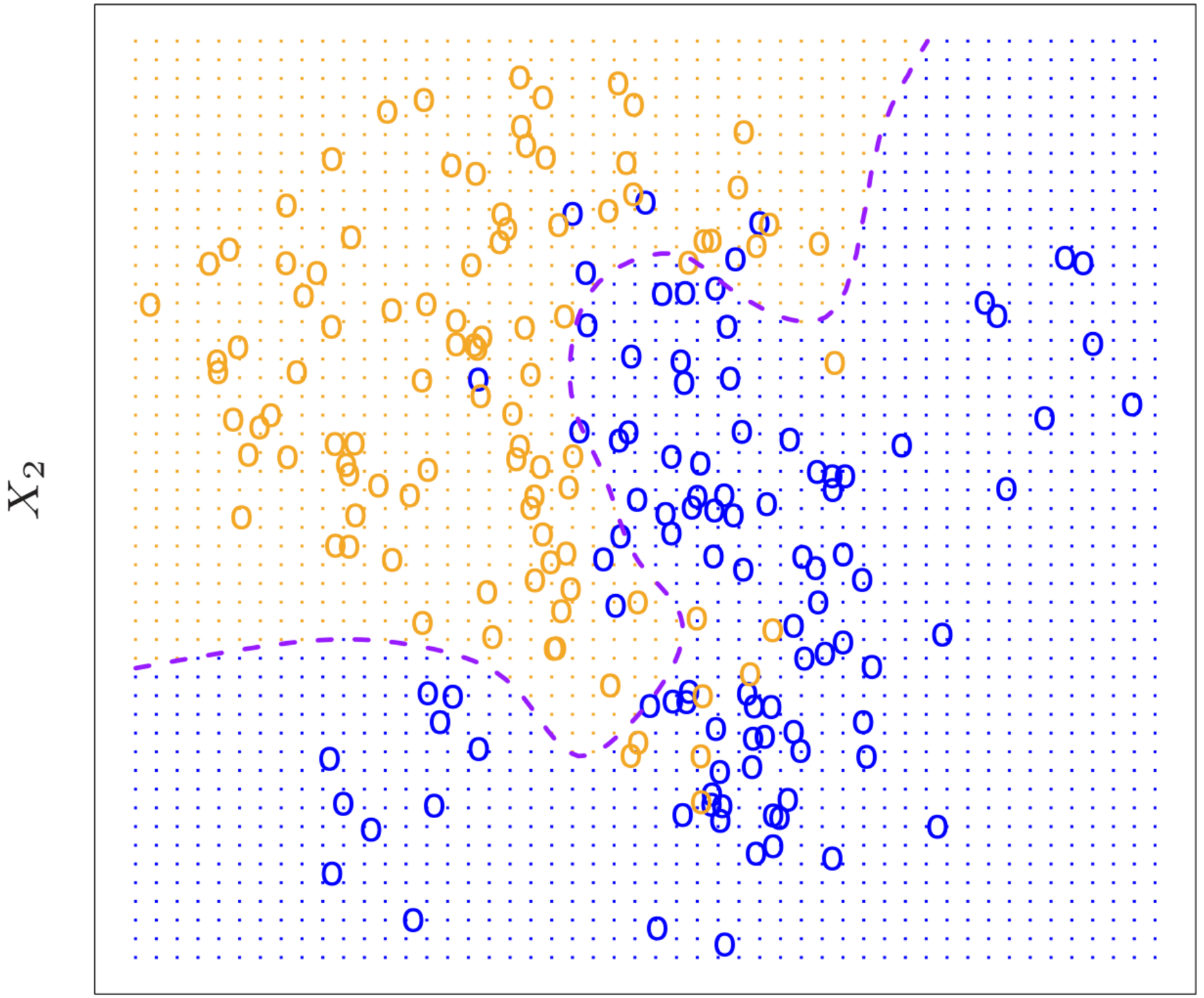
- Typically we measure the performance of $\hat{C}(x)$ using the misclassification error rate:

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

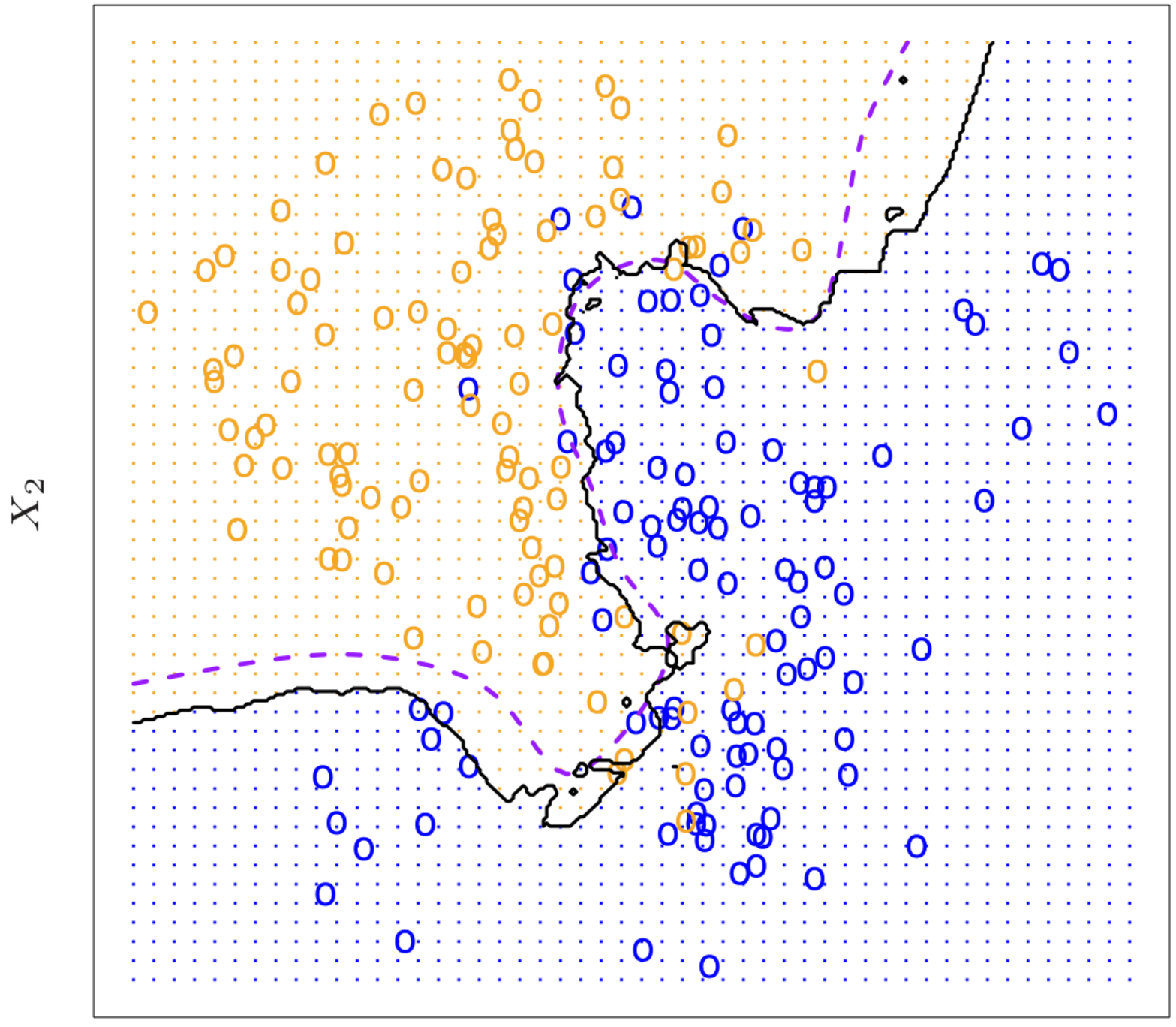
- The Bayes classifier (using the true $p_k(x)$) has smallest error (in the population).

- 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.

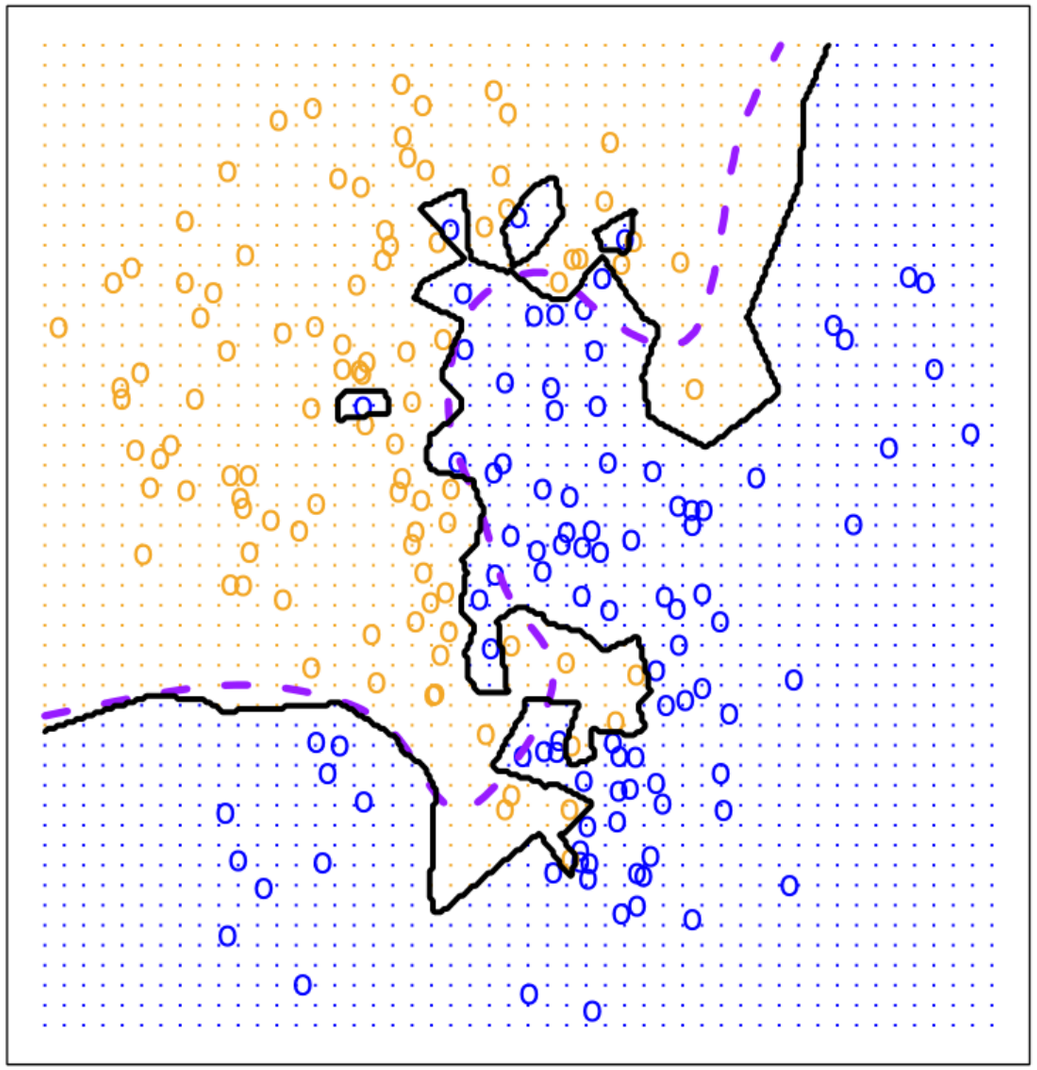
Example: K-nearest neighbors in two dimensions



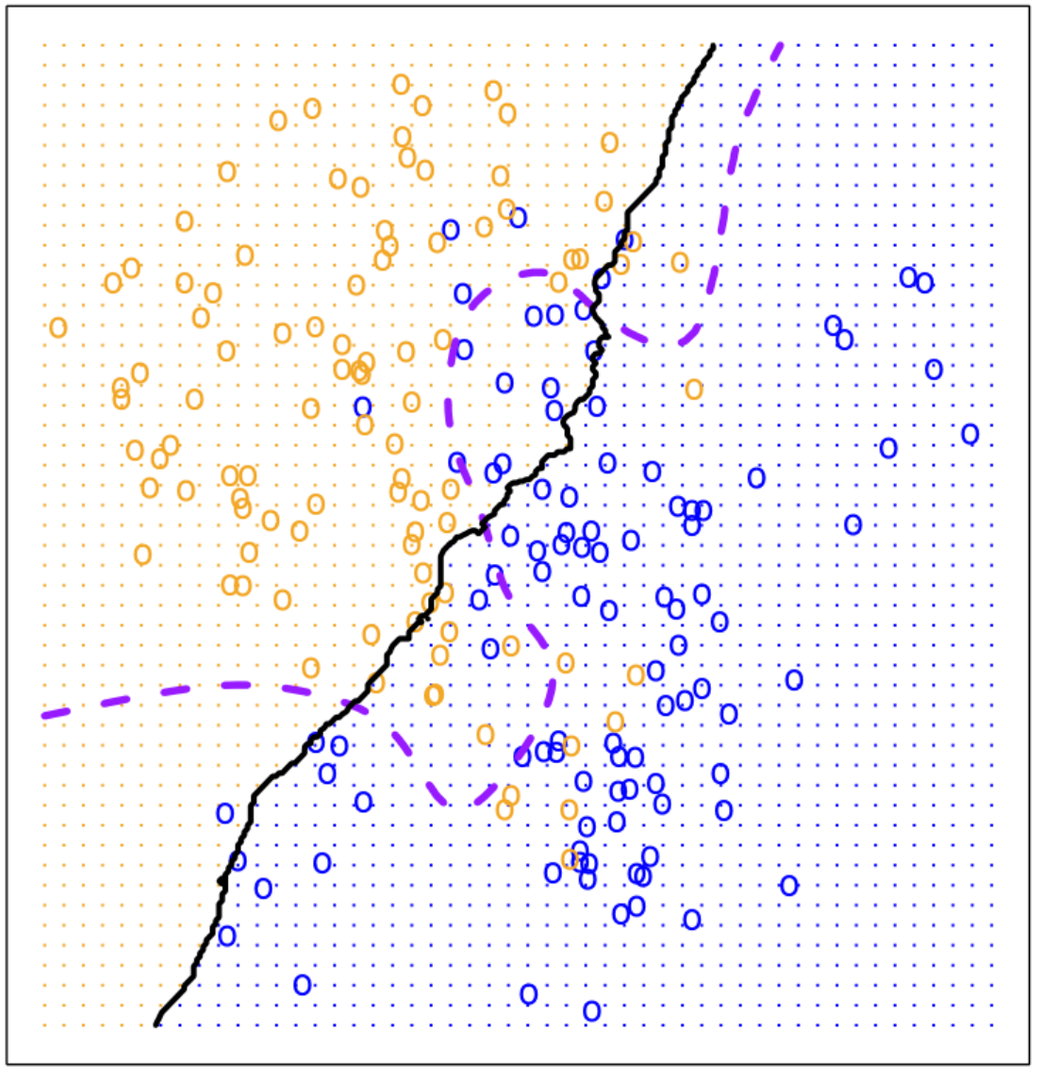
KNN: K=10



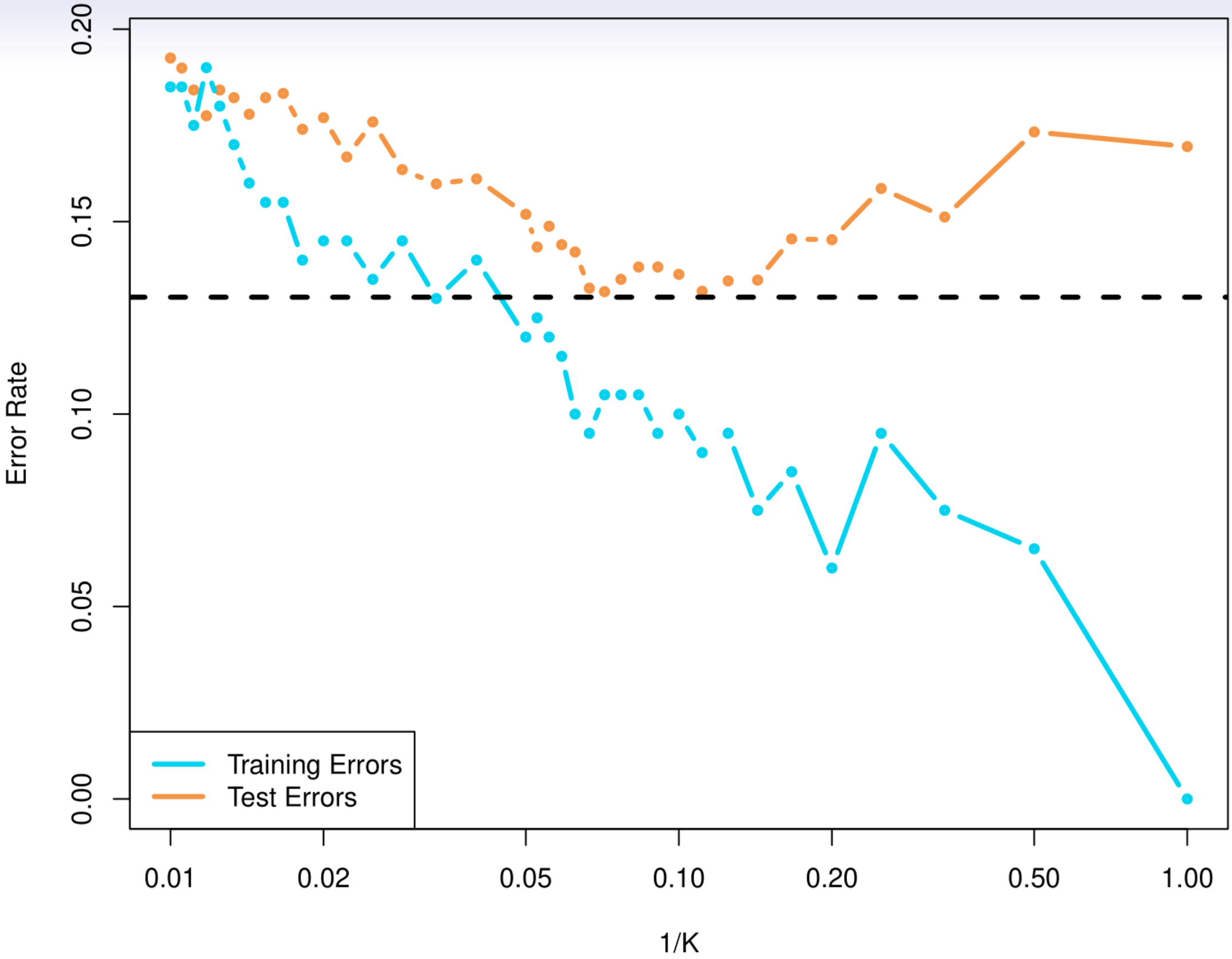
KNN: K=1



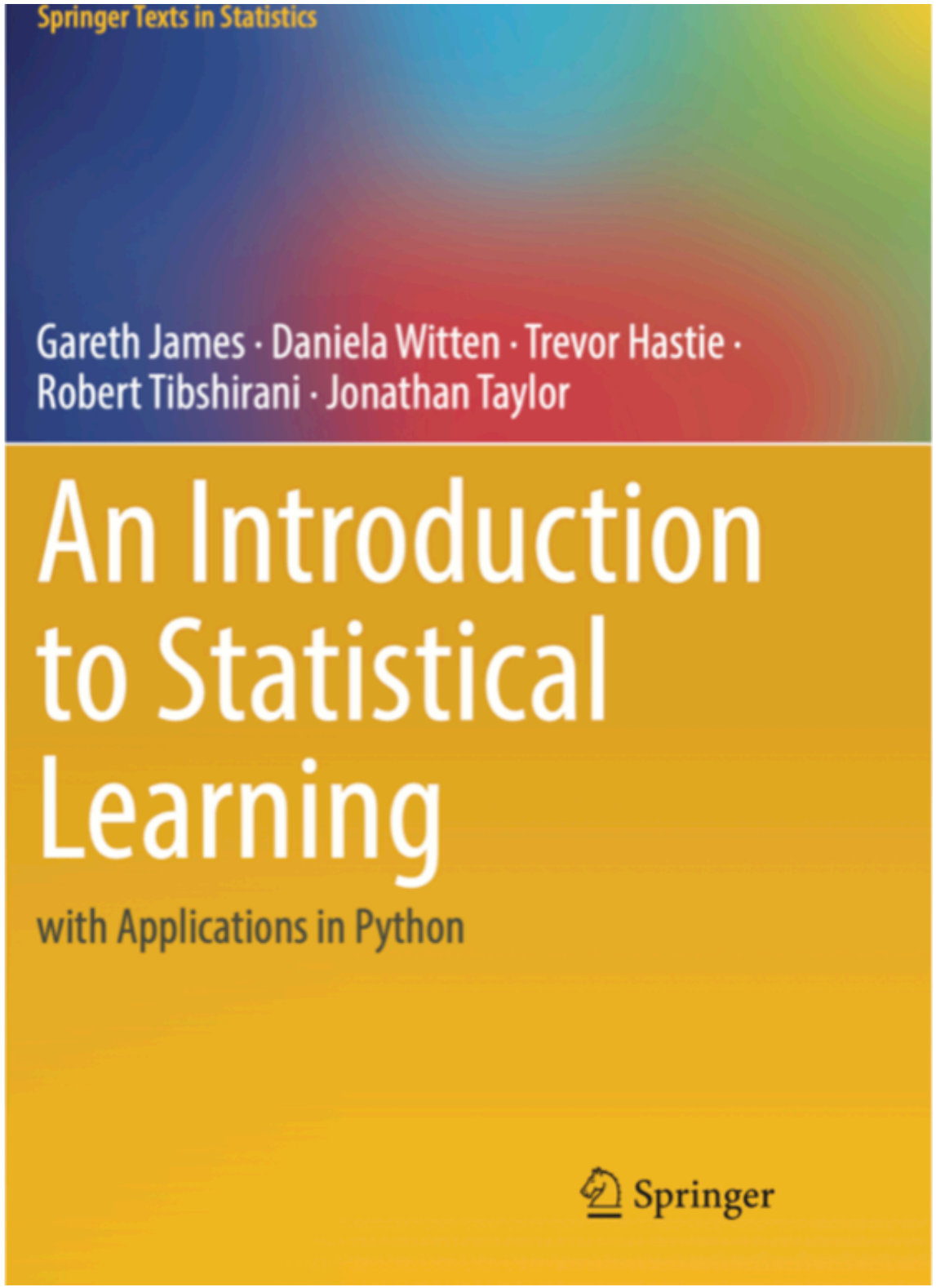
KNN: K=100



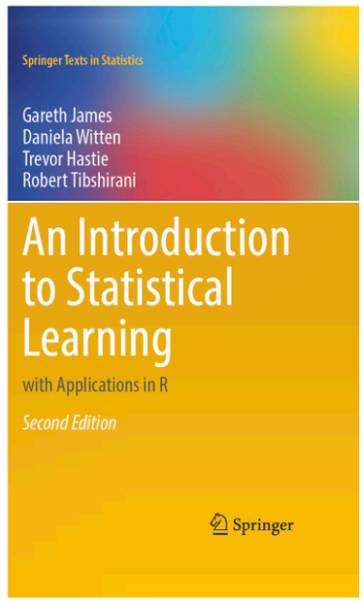
33

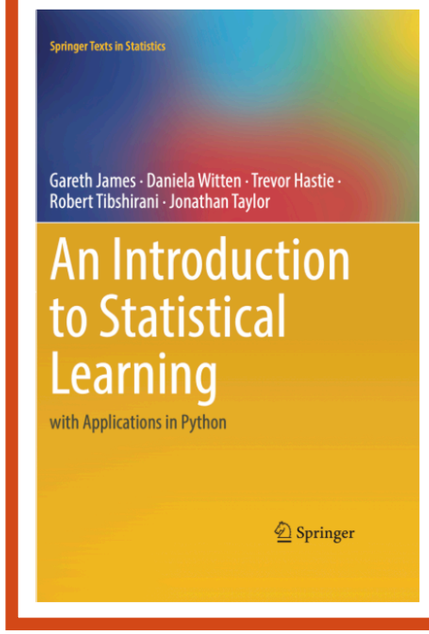

Error Rate
0.00
0.05
0.10
0.15
0.20
0.01
0.02
0.05
0.10
0.20
0.50
1.00
1/K
Training Errors
Test Errors

2.3 Lab: Introduction to python



https://www.statlearning.com/

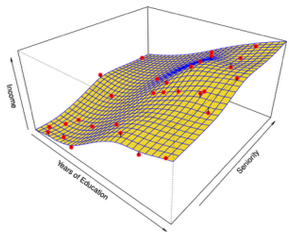




https://www.statlearning.com/

https://www.statlearning.com/resources-python

You will need to install the `ISLP` package, which provides access to the datasets and custom-built functions that we provide. Inside a macOS or Linux terminal type `pip install ISLP`; this also installs most other packages needed in the labs. The `Python` resources page has a link to the `ISLP` documentation website.



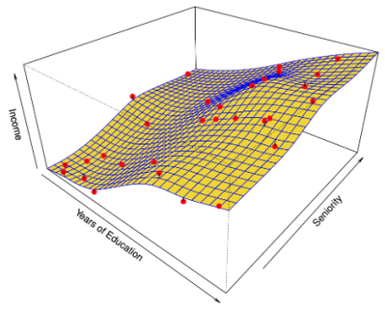
Welcome to ISLP documentation!

ISLP is a Python library to accompany *Introduction to Statistical Learning with applications in Python*. See the statistical learning homepage for more details.

See the ISLP reference

Contents

https://intro-stat-learning.github.io/ISLP/



Labs

launch binder

Download!

The current version of the labs for `ISLP` are included here.

Package versions

Attention

Python packages change frequently. The labs here are built with ISLP_labs/v2.1.2. Visit the lab git repo for specific instructions to install the frozen environment.

Warning

The version of the `ISLP` library used to build these labs may differ slightly from the one documented here. The labs are built with ISLP/v0.3.19.

The Binder link above will run ISLP_labs/v2.1.2 with library version ISLP/v0.3.19.