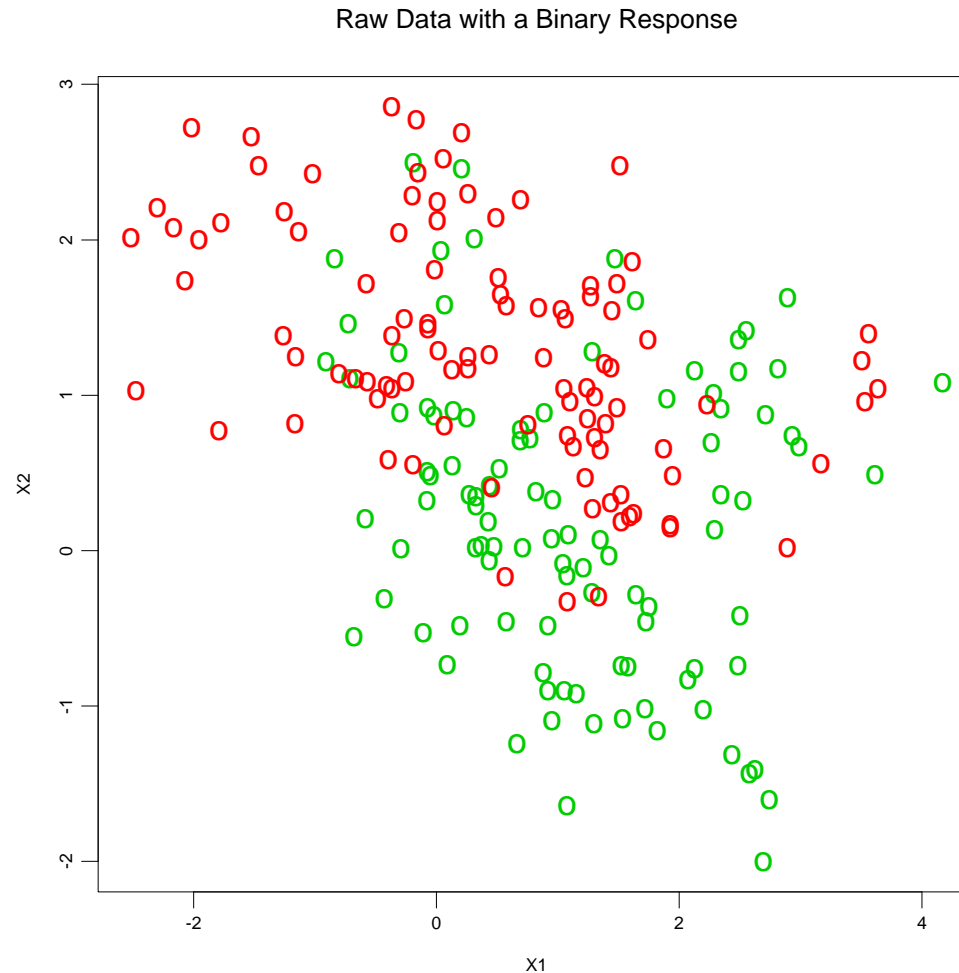


Overview of Supervised Learning

Notation

- X : inputs, feature vector, predictors, *independent variables*.
Generally X will be a vector of p real values. Qualitative features are coded in X using, for example, dummy variables. Sample values of X generally in lower case; x_i is i th of N sample values. Bold \mathbf{X} represents a matrix of feature values, with an x_i in each row.
- Y : output, response, *dependent variable*. Typically a scalar, can be a vector, of real values. Again y_i is a realized value.
- G : a qualitative response, taking values in a discrete set \mathcal{G} ; e.g. $\mathcal{G} = \{\textit{survived}, \textit{died}\}$. We often code G via a binary *indicator* response vector Y .



200 points generated in \mathbb{R}^2 from an unknown distribution; 100 in each of two classes $\mathcal{G} = \{\text{GREEN}, \text{RED}\}$. Can we build a rule to predict the color of future points?

Linear regression

- Code $Y = 1$ if $G = \text{RED}$, else $Y = 0$.
- We model Y as a linear function of X :

$$\hat{Y} = \hat{\beta}_0 + \sum_{j=1}^p X_j \hat{\beta}_j = \mathbf{X}^T \hat{\beta}$$

(Drop β_0 and make first element of X equal to 1)

- Obtain β by *least squares*, by minimizing the quadratic criterion:

$$RSS(\beta) = \sum_{i=1}^N (y_i - x_i^T \beta)^2$$

- Given an $N \times p$ model matrix \mathbf{X} and a response vector \mathbf{y} ,

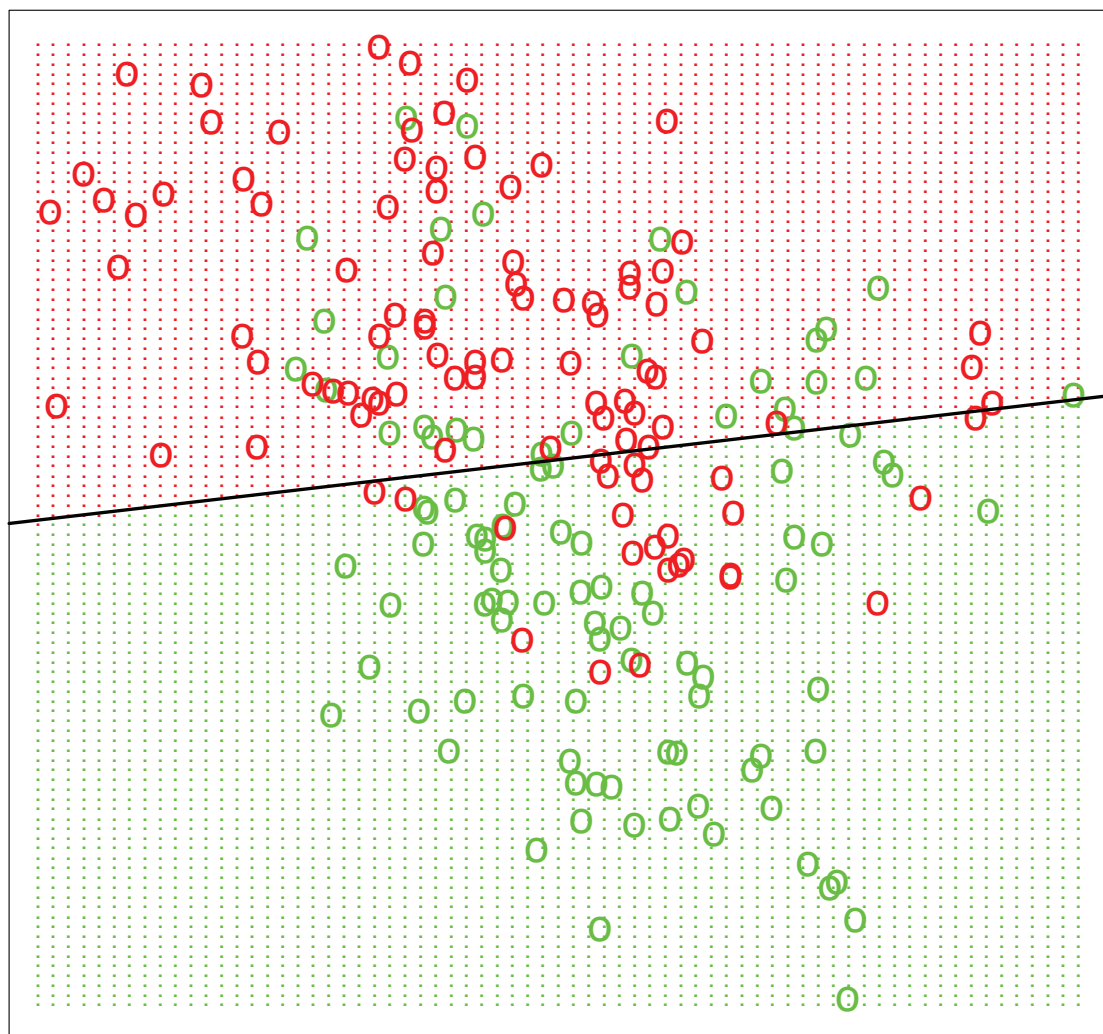
$$\hat{\beta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

- Prediction at a future point x_0 is $\hat{Y}(x_0) = x_0^T \hat{\beta}$. Also

$$\hat{G}(x_0) = \begin{cases} \text{RED} & \text{if } \hat{Y}(x_0) > 0.5, \\ \text{GREEN} & \text{if } \hat{Y}(x_0) \leq 0.5. \end{cases}$$

- The *decision boundary* is $\{x | x^T \hat{\beta} = 0.5\}$ is linear (and seems to make many errors on the training data).

Linear Regression of 0/1 Response



Possible scenarios

Scenario 1: The data in each class are generated from a Gaussian distribution with uncorrelated components, same variances, and different means.

Scenario 2: The data in each class are generated from a mixture of 10 gaussians in each class.

For Scenario 1, the linear regression rule is almost optimal (Chapter 4).

For Scenario 2, it is far too rigid.

K-Nearest Neighbors

A natural way to classify a new point is to have a look at its neighbors, and take a vote:

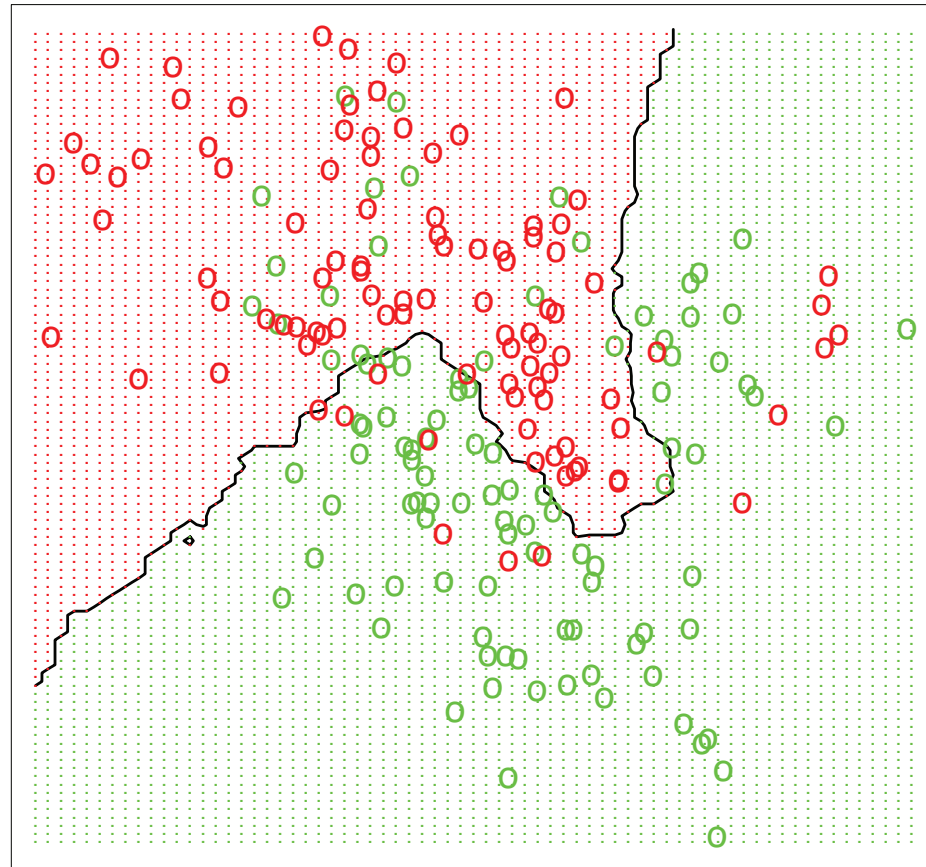
$$\hat{Y}_k(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i,$$

where $N_k(x)$ is a neighborhood of x that contains exactly k neighbors (k -nearest neighborhood).

If there is a clear dominance of one of the classes in the neighborhood of an observation x , then it is likely that the observation itself would belong to that class, too. Thus the classification rule is the majority voting among the members of $N_k(x)$. As before,

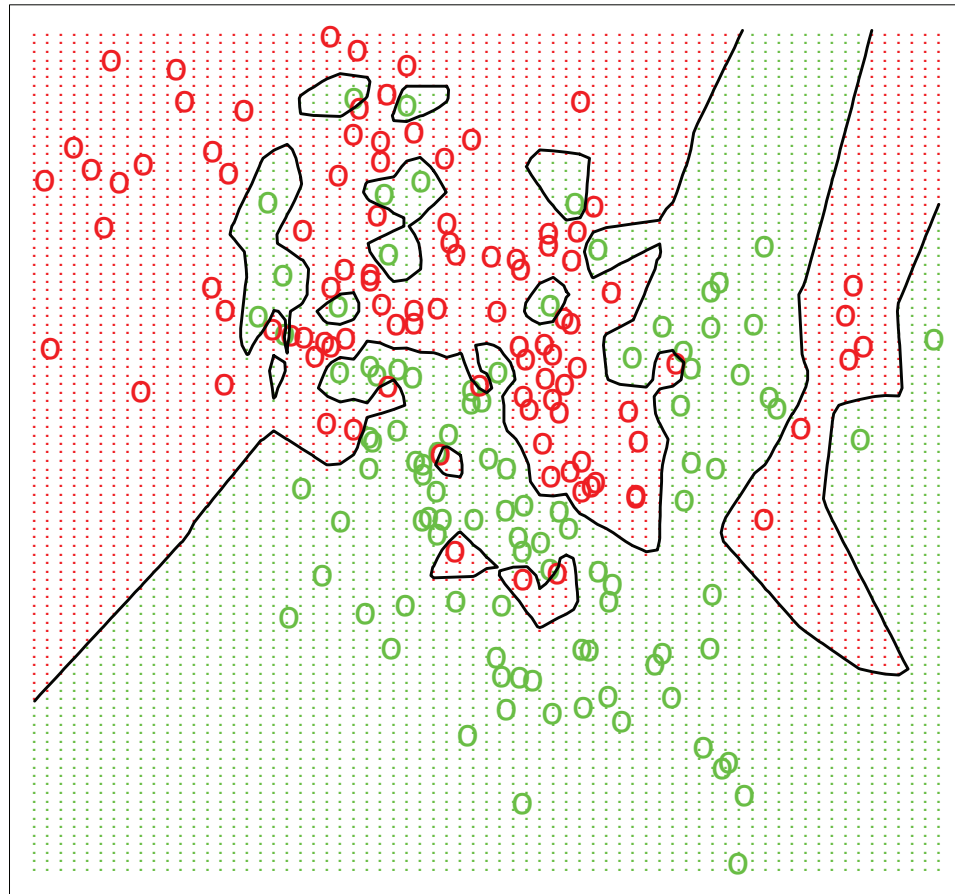
$$\hat{G}_k(x_0) = \begin{cases} \text{RED} & \text{if } \hat{Y}_k(x_0) > 0.5, \\ \text{GREEN} & \text{if } \hat{Y}_k(x_0) \leq 0.5. \end{cases}$$

15-Nearest Neighbor Classifier

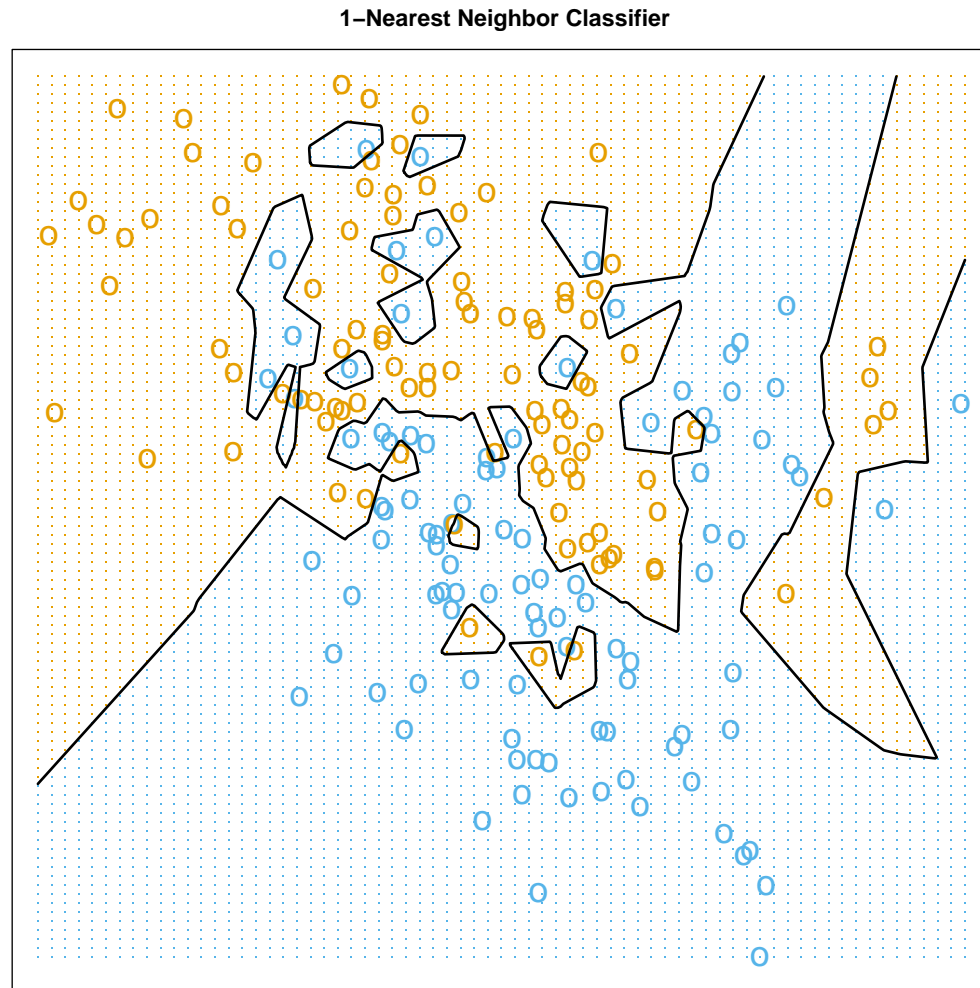


15-nearest neighbor classification. Fewer training data are misclassified, and the decision boundary adapts to the local densities of the classes.

1-Nearest Neighbor Classifier



1-nearest neighbor classification. *None* of the training data are misclassified.



1-nearest neighbor classification figure redrawn (at a finer resolution).
Now the Voronoi regions look better.

Discussion

Linear regression uses 3 parameters to describe its fit. Does K-nearest neighbors use 1 (the value of k here)?

More realistically, k -nearest neighbors uses N/k *effective number of parameters*

Many modern procedures are variants of linear regression and K-nearest neighbors:

- Kernel smoothers
- Local linear regression
- Linear basis expansions
- Projection pursuit and neural networks
- support-vector machines

Linear regression vs k-nearest neighbors?

First we expose the oracle.

The density for each class was an equal mixture of 10 Gaussians.

For the **GREEN** class, its 10 means were generated from a $N((1, 0)^T, \mathbf{I})$ distribution (and considered fixed).

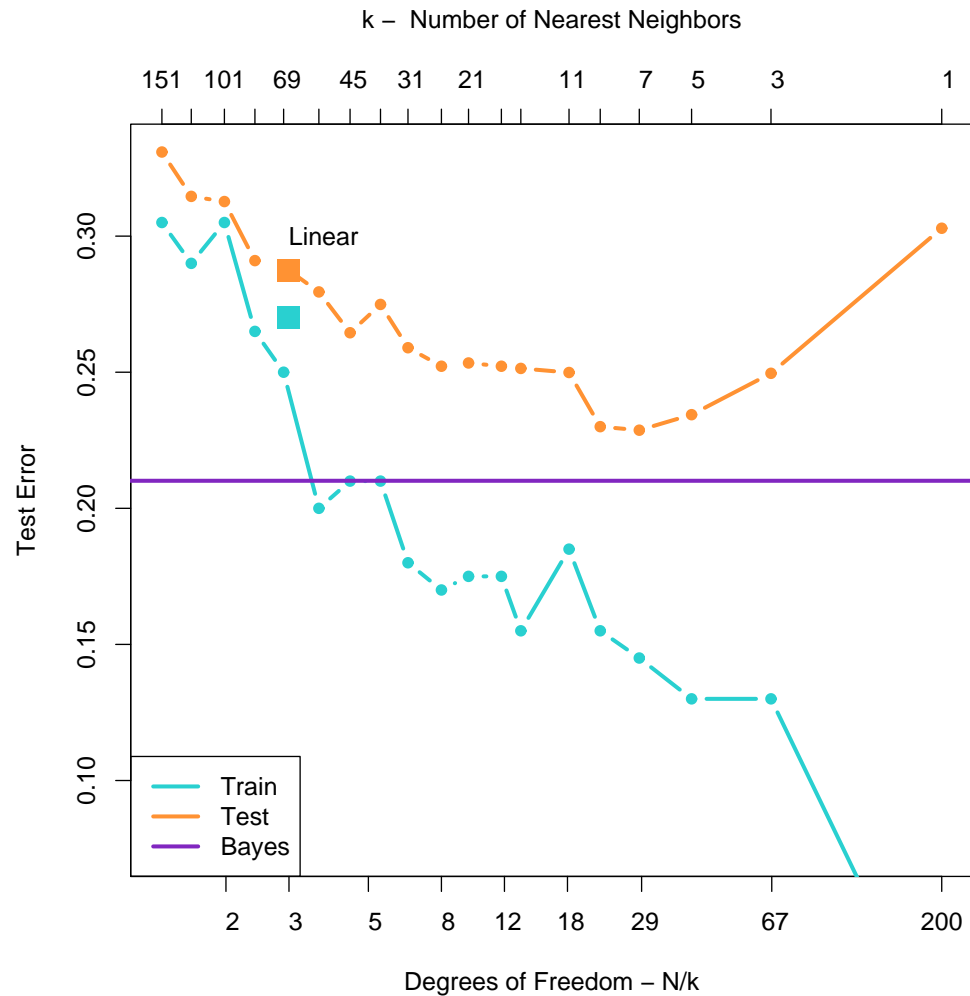
For the **RED** class, the 10 means were generated from a $N((0, 1)^T, \mathbf{I})$. The within cluster variances were $1/5$.



See see pp 16. for more details, or

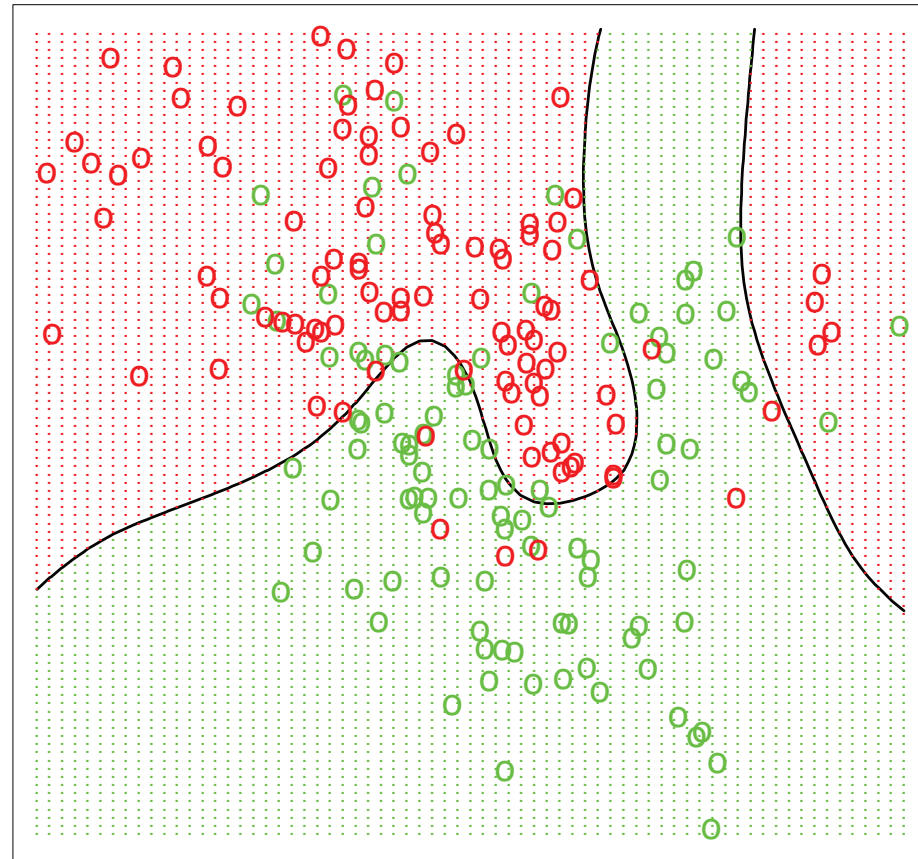
<http://www-stat.stanford.edu/ElemStatLearn>

(book website) for the actual data.



Results of classifying 10,000 test observations generated from this distribution. The *Bayes Error* is the best performance possible

Bayes Optimal Classifier



The optimal Bayes decision boundary for the simulation example. Since the generating density is known for each class, this boundary can be calculated exactly.

Statistical decision theory

Case 1: Quantitative output Y

- Let $X \in R^p$ denote a real valued random input vector
- We have a *Loss function* $L(Y, f(X))$ for penalizing errors in prediction.
- Most common and convenient is *squared error loss*:

$$L(Y, f(X)) = (Y - f(X))^2$$

- This leads us to a criterion for choosing f ,

$$EPE(f) = E_{XY}(Y - f(X))^2$$

the Expected (squared) Prediction Error.

-

$$\begin{aligned} EPE(f) &= E_X[E_{Y|x}(Y - E(Y|x) + E(Y|x) - f(x))]^2 \\ &= E_X(E_{Y|x}(Y - E(Y|x))^2 + E_X(E(Y|x) - f(x))^2) \\ &= \text{Irreducible Error} + MSE \end{aligned}$$

- Minimizing $EPE(f)$ leads to a solution $f(x) = E(Y|X = x)$, the conditional expectation, also known as the *regression* function.
- Irreducible error also referred to as *Bayes Error*.

Case 2: Qualitative output G

- Suppose our prediction rule is $\hat{G}(X)$, and G and $\hat{G}(X)$ take values in \mathcal{G} , with $\text{card}(\mathcal{G}) = K$.
- We have a different loss function for penalizing prediction errors. $L(k, \ell)$ is the price paid for classifying an observation belonging to class \mathcal{G}_k as \mathcal{G}_ℓ .
- Most often we use the 0-1 loss function where all misclassifications are charged a single unit.
- The expected prediction error is

$$EPE = E[L(G, \hat{G}(X))]$$

- Solution is

$$\hat{G}(x) = \operatorname{argmin}_{g \in \mathcal{G}} \sum_{k=1}^K L(\mathcal{G}_k, g) P(\mathcal{G}_k | X = x)$$

With the 0-1 loss function this simplifies to

$$\hat{G}(x) = \mathcal{G}_k \text{ if } P(\mathcal{G}_k|x) = \max_{g \in \mathcal{G}} P(g|X)$$

This is known as the *Bayes classifier*. It just says that we should pick the class having maximum probability at the input x .

Question: how did we construct the Bayes classifier for our simulation example?

- K-nn tries to implement conditional expectations directly, by
 - Approximating expectations by sample averages
 - Relaxing the notion of conditioning at a point, to conditioning in a region about a point.
- As $N, k \rightarrow \infty$, such that $k/N \rightarrow 0$, the K-nearest neighbor estimate $\hat{f}(x) \rightarrow E(Y|X = x)$ — it is *consistent*.
- Linear regression assumes a (linear) structural form for $f(x) = x^T \beta$, and minimizes sample version of EPE directly.
- As sample size grows, our estimate of linear coefficients $\hat{\beta}$ converges to the optimal $\beta_{opt} = E(XX^T)^{-1}E(XY)$.
- Model is limited by the linearity assumption

Why not always use k-nearest neighbors?

Bias of $\hat{f}(x_0)$:

$$f(x_0) - E\hat{f}(x_0)$$

Variance of $\hat{f}(x_0)$:

$$E[\hat{f}(x_0) - E\hat{f}(x_0)]^2$$

EPE (Expected Prediction Error) of $\hat{f}(x_0)$:

$$E[Y - \hat{f}(x_0)]^2$$

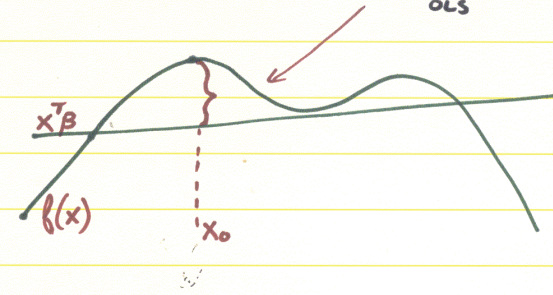
where the expectation is with respect to *everything* that is random.

PERFORMANCE OF OLS VS KNN

$$\text{OLS: } \text{EPE}(x_0) \propto \sigma^2 + \frac{p}{N} \sigma^2 + \text{bias}_{\text{OLS}}^2(x_0)$$

$$p = \text{DIM}(x)$$

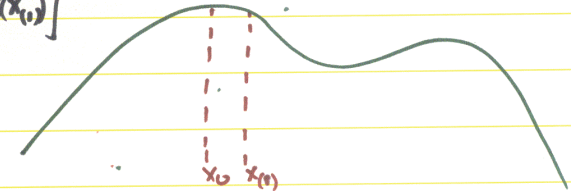
$N = \text{TRAINING SAMPLE SIZE}$



$$\text{KNN: } (k=1)$$

$$\text{EPE}(x_0) \propto \sigma^2 + \sigma^2 + \text{bias}_{1\text{-NN}}^2(x_0)$$

$$\text{bias}_{1\text{-NN}}^2 = [f(x_0) - E f(x_{(n)})]^2$$



$$\bullet \sigma^2 + \frac{p}{N} \sigma^2 \propto \sigma^2 < 2\sigma^2$$

$\bullet f(x)$ DETERMINES THE BIAS

$\bullet \text{FOR LARGE } N, \text{ K-NN IS OPTIMAL?}$

MSE=Variance + Squared Bias

$$\begin{aligned} EPE(x_0) &= E_{\mathcal{T}} E_{Y|x_0} (Y - \hat{f}(x_0))^2 \\ &= \sigma^2(x_0) + E_{\mathcal{T}} (\hat{f}(x_0) - f(x_0))^2 \\ &= \text{Irreducible Error}(x_0) + \text{MSE}(x_0) \end{aligned}$$

where $\sigma^2(x_0) = \text{Var}(Y|x_0)$, $f(x_0) = E(Y|x_0)$, and \mathcal{T} =*Training Data*.

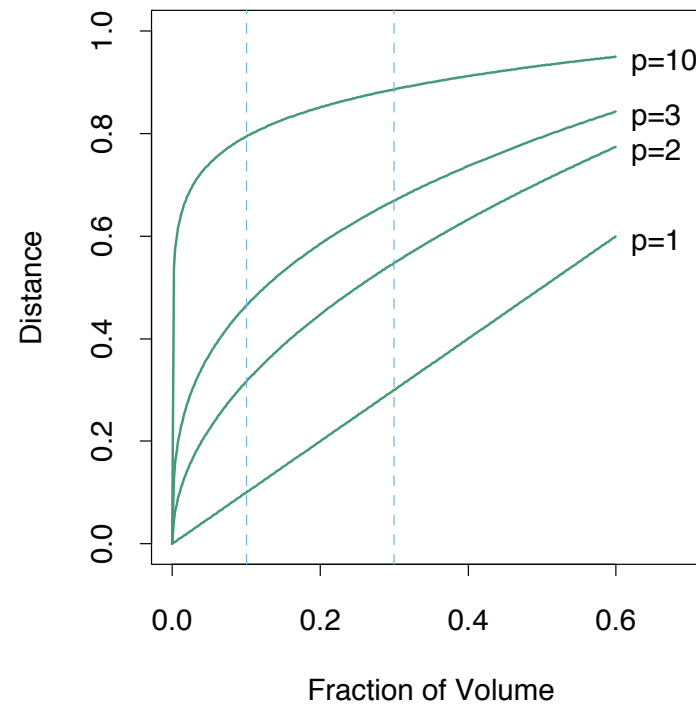
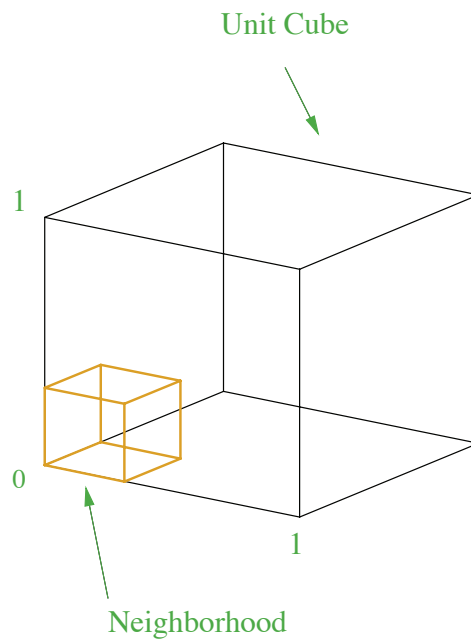
$$\begin{aligned} MSE(x_0) &= E_{\mathcal{T}} \left(\hat{f}(x_0) - E_{\mathcal{T}} \hat{f}(x_0) + E_{\mathcal{T}} \hat{f}(x_0) - f(x_0) \right)^2 \\ &= E_{\mathcal{T}} (\hat{f}(x_0) - E_{\mathcal{T}} \hat{f}(x_0))^2 + (E_{\mathcal{T}} \hat{f}(x_0) - f(x_0))^2 \\ &= \text{Variance} + \text{Squared Bias} \end{aligned}$$

Often prediction error (EPE) is a vehicle for getting at MSE, which tells how well we are estimating $f(x)$, our real interest.

Curse of dimensionality

K-nearest neighbors can fail in high dimensions, because it becomes difficult to gather K observations close to a target point x_0 :

- near neighborhoods tend to be spatially large, and estimates are biased.
- reducing the spatial size of the neighborhood means reducing K , and the variance of the estimate increases.



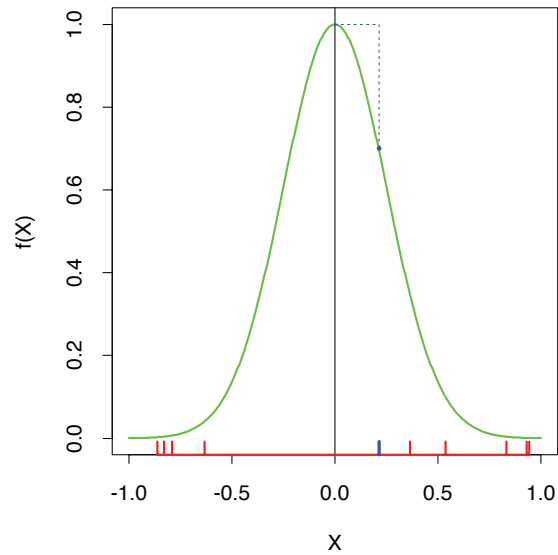
- Most points are at the boundary
- Sampling density is proportional to $N^{1/p}$; if 100 points are sufficient to estimate a function in \mathbb{R}^1 , 100^{10} are needed to achieve similar accuracy in \mathbb{R}^{10}

Example 1

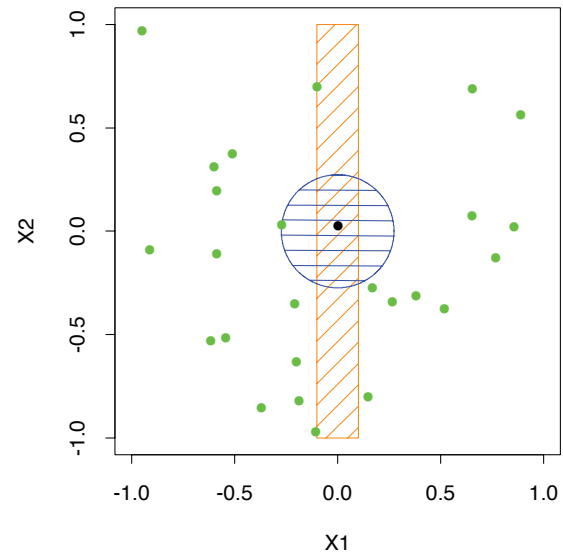
- 1000 training examples x_i generated uniformly on $[-1, 1]^p$.
- $Y = f(X) = e^{-8\|X\|^2}$ (no measurement error).
- use the 1-nearest-neighbor rule to predict y_0 at the test-point $x_0 = 0$.

$$\begin{aligned}\text{MSE}(x_0) &= \mathbb{E}_{\mathcal{T}}[f(x_0) - \hat{y}_0]^2 \\ &= \mathbb{E}_{\mathcal{T}}[\hat{y}_0 - \mathbb{E}_{\mathcal{T}}(\hat{y}_0)]^2 + [\mathbb{E}_{\mathcal{T}}(\hat{y}_0) - f(x_0)]^2 \\ &= \text{Var}_{\mathcal{T}}(\hat{y}_0) + \text{Bias}^2(\hat{y}_0).\end{aligned}$$

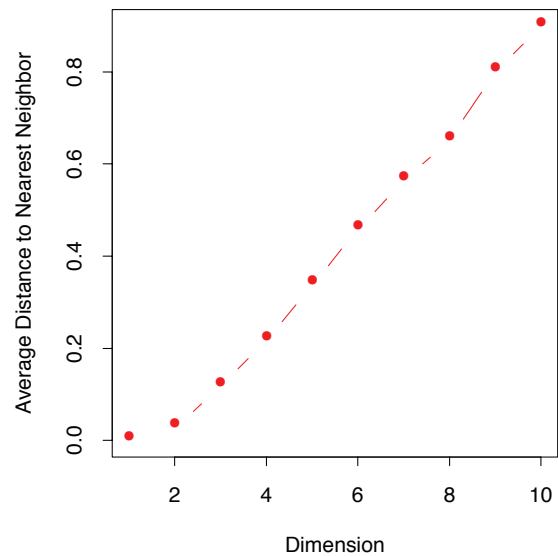
1-NN in One Dimension



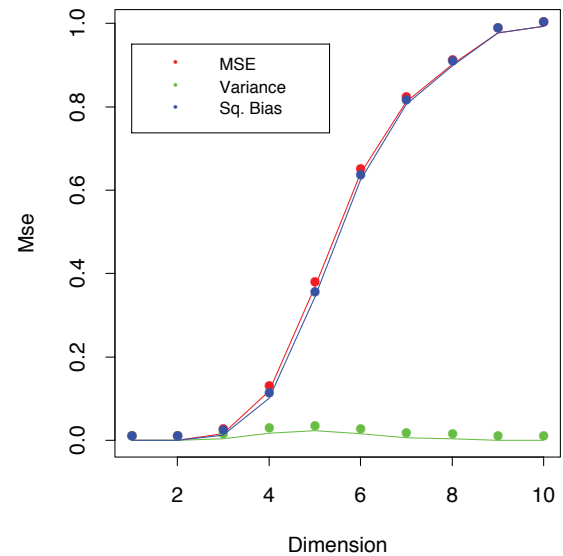
1-NN in One vs. Two Dimensions

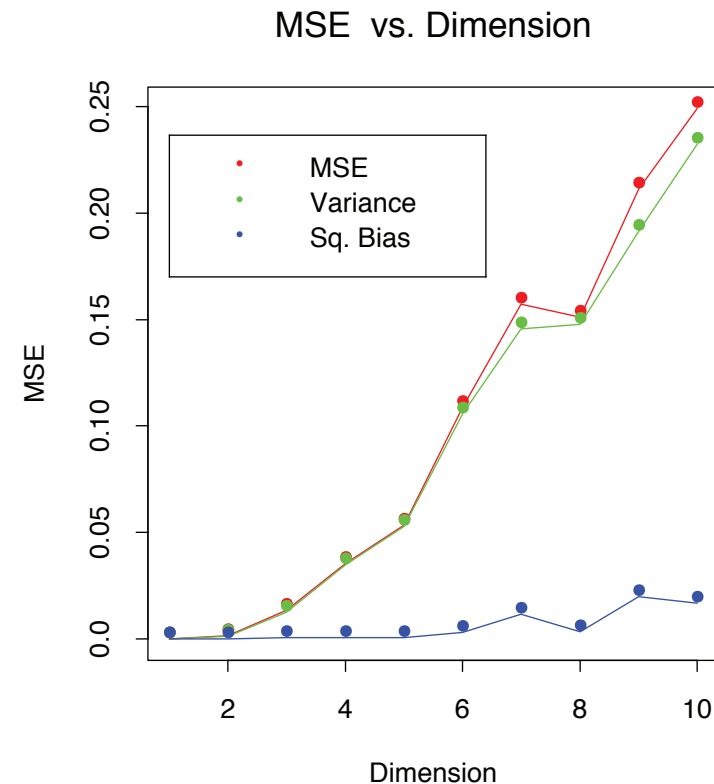
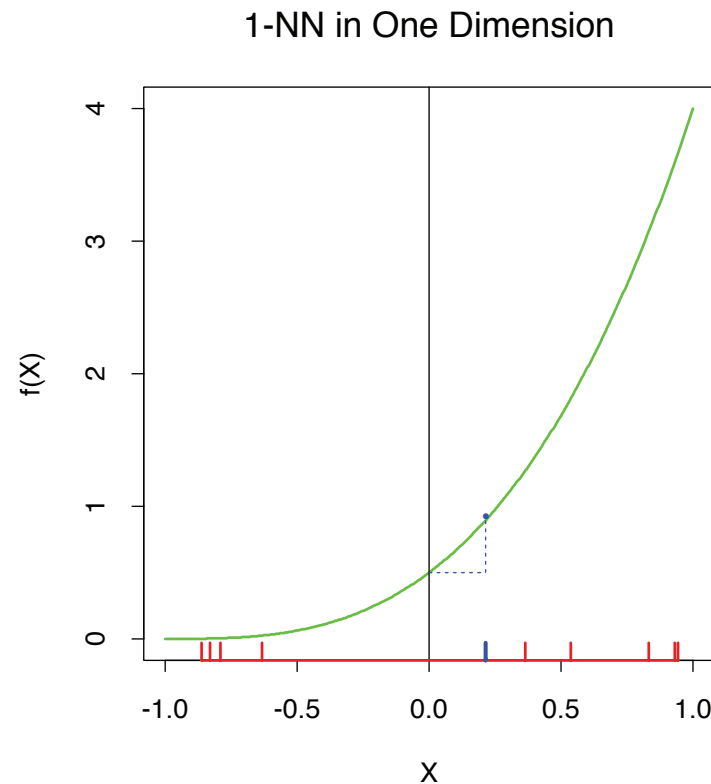


Distance to 1-NN vs. Dimension



MSE vs. Dimension





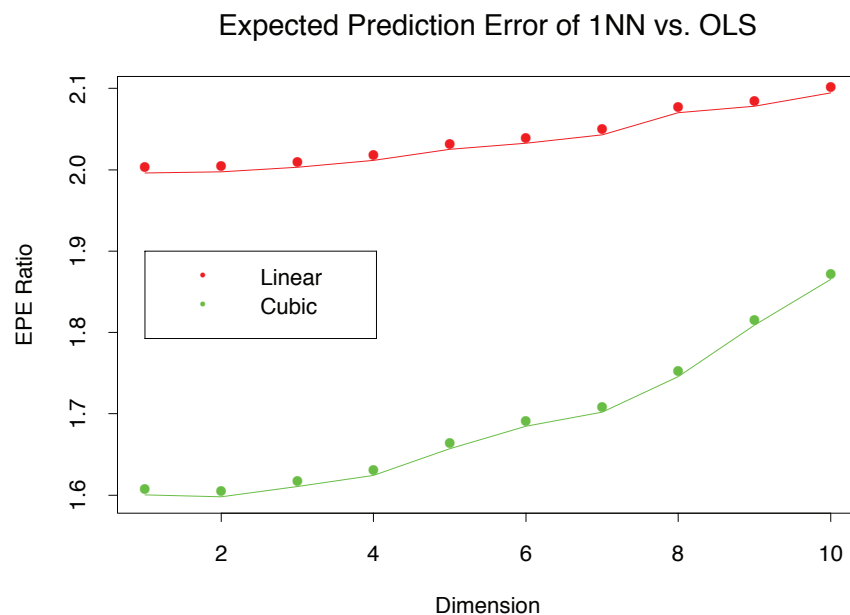
Same scenario, except here $f(X) = \frac{1}{2}(X_1 + 1)^3$. Here we have

- $E\hat{f}(x_0) = Ef(X_{(1)}) \approx f(EX_{(1)}) = f(x_0)$ for all dimensions
- $\text{Var}\hat{f}(x_0) = \text{Var}f(X_{(1)}) \uparrow$ with dimension

Example 2

If the linear model is correct, or almost correct, K-nearest neighbors will do much worse than linear regression.

In cases like this (and of course, assuming we know this is the case), simple linear regression methods are not affected by the dimension.



The curves show the expected prediction error (at $x_0 = 0$) for 1-nearest neighbor relative to least-squares for the model $Y = f(x) + \varepsilon$. For the **RED** curve, $f(x) = x_1$, while for the **GREEN** curve, $f(x) = \frac{1}{2}(x_1 + 1)^3$.

Statistical Models

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

with $E(\varepsilon) = 0$ and X and ε independent.

- $E(Y|X) = f(X)$
- $\Pr(Y|X)$ depends on X only through $f(X)$.
- Useful approximation to the truth — all unmeasured variables captured by ε
- N realizations $y_i = f(x_i) + \varepsilon_i$, $i = 1, \dots, N$
- Assume ε_i and ε_j are independent.

More generally can have, for example, $\text{Var}(Y|X) = \sigma^2(X)$.

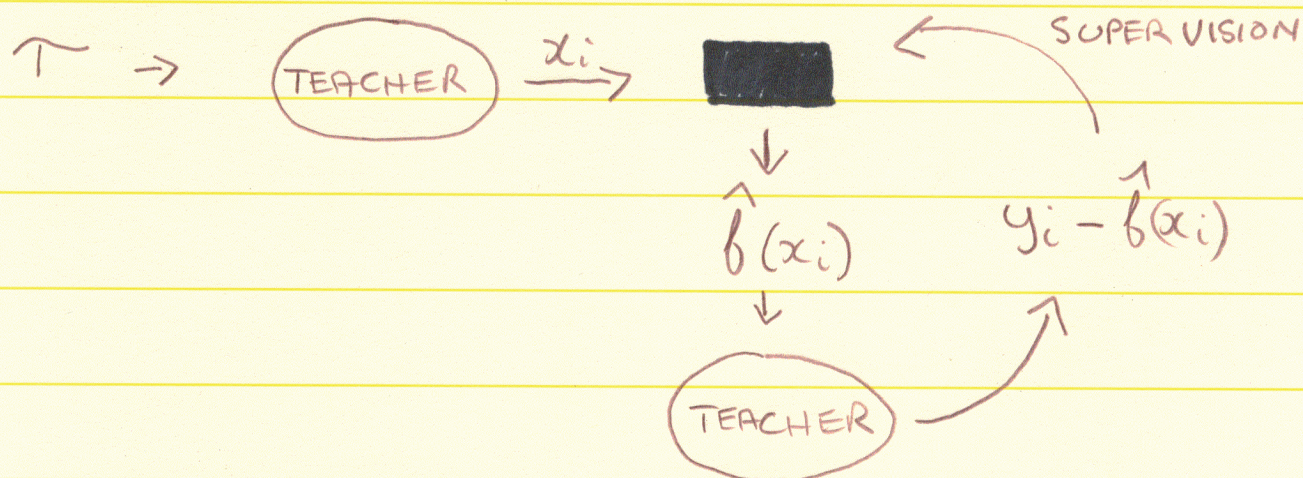
For qualitative outcomes $\{\Pr(G = \mathcal{G}_k|X)\}_1^K = p(X)$ which we model directly.

ALGORITHMIC APPROACH

$$\mathcal{T} = \{x_i, y_i\}_1^n$$

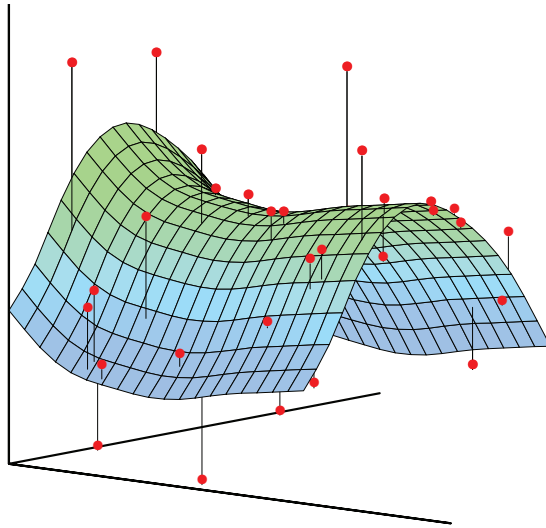
$$\mathcal{T} \rightarrow \text{ALGORITHM} \rightarrow \hat{f}$$

SUPERVISED LEARNING ALGORITHM



"LEARNING BY EXAMPLE"

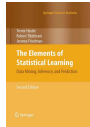
Function Approximation



$$\text{RSS}(\theta) = \sum_{i=1}^N (y_i - f_{\theta}(x_i))^2$$

Assumes

- x_i, y_i are points in, say \mathbb{R}^{p+1} .
- A (parametric or non-parametric) form for $f(X)$
- A loss function for measuring the quality of the approximation.

More generally, *Maximum Likelihood Estimation* (R.A. Fisher, see  pp 514.) provides a natural basis for estimation. We will see examples such as logistic regression via the binomial likelihood.

$$\begin{aligned}\Pr(G = k|X = x) &= p_{k,\theta}(x) \\ \ell(\theta) &= \sum_{i=1}^N \log p_{g_i,\theta}(x_i)\end{aligned}$$

Structured Regression Models

$$\text{RSS}(f) = \sum_{i=1}^N (y_i - f(x_i))^2$$

- Any function passing through (x_i, y_i) , $i = 1, \dots, N$ has $\text{RSS} = 0$
- Need to restrict the class
- Usually restrictions impose local behavior — see equivalent kernels in chapters 5 and 6
- Any method that attempts to approximate locally varying functions is “cursed”
- Alternatively, any method that “overcomes” the curse, assumes an implicit metric that does not allow neighborhoods to be simultaneously small in all directions.

Classes of Restricted Estimators

Some of the classes of restricted methods that we cover are

- Roughness Penalty and Bayesian Methods (chap 5)

$$\text{Penalized RSS}(f, \lambda) = \text{RSS}(f) + \lambda J(f)$$

- Basis functions and dictionary methods (chap 5)

$$f_{\theta}(x) = \sum_{m=1}^M \theta_m h_m(x)$$

- Kernel Methods and Local Regression (chap 6)

$$\text{RSS}(f_{\theta}, x_0) = \sum_{i=1}^N K_{\lambda}(x_0, x_i) (y_i - f_{\theta}(x_i))^2$$

Model Selection and the Bias-Variance Tradeoff

Many of the flexible methods have a *complexity parameter*:

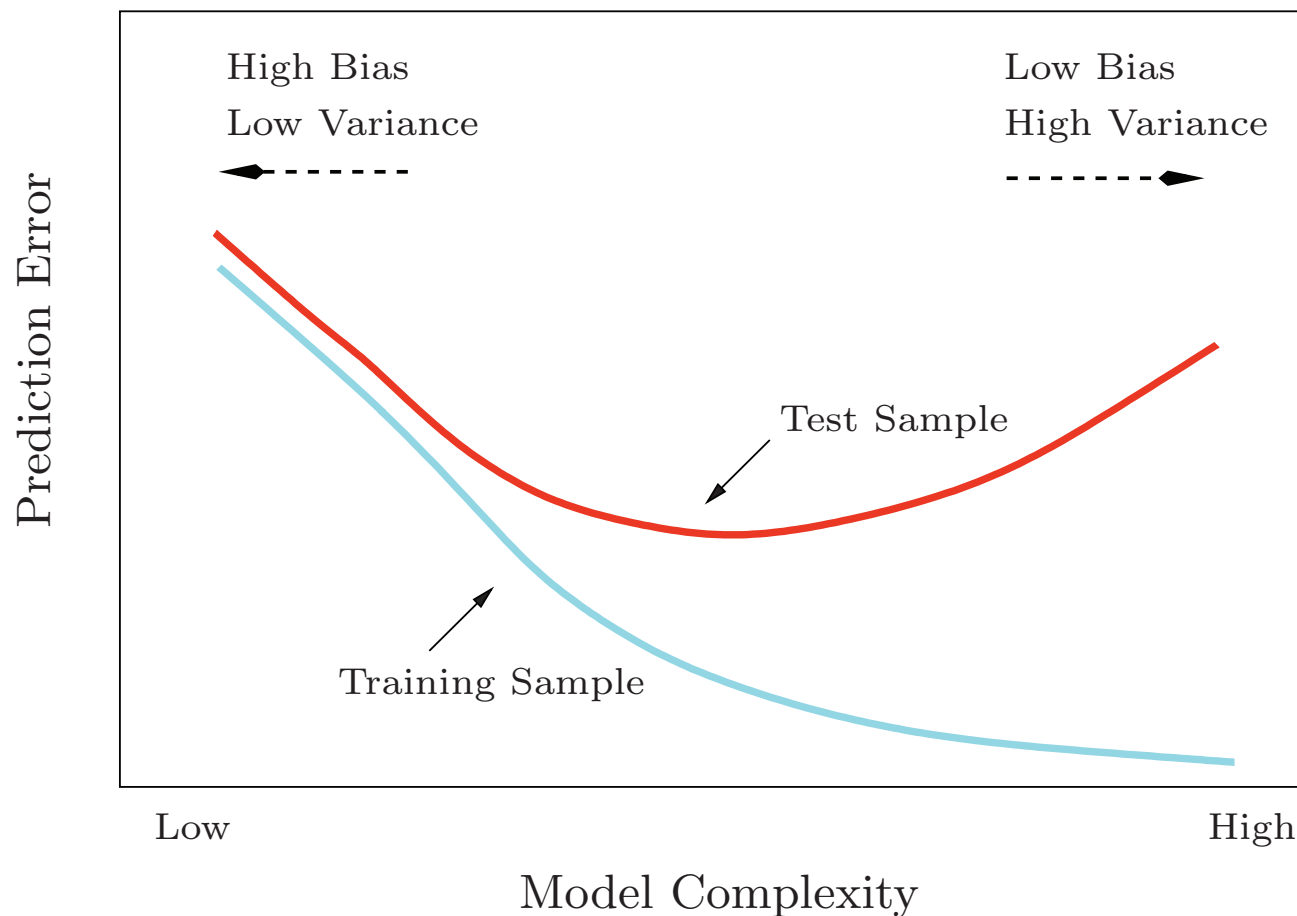
- the multiplier of the penalty term
- the width of the kernel
- the number of basis functions

Cannot use RSS to determine this parameter — why?

Can use *Prediction error* on unseen test cases to guide us

E.g. $Y = f(X) + \varepsilon$, K-nn (and assume the sample x_i are fixed):

$$\begin{aligned} \mathbb{E}[(Y - \hat{f}_k(x_0))^2 | X = x_0] &= \sigma^2 + \text{Bias}^2(\hat{f}_k(x_0)) + \text{Var}_{\mathcal{T}}(\hat{f}_k(x_0)) \\ &= \sigma^2 + \left[f(x_0) - \frac{1}{k} \sum_{\ell=1}^k f(x_{(\ell)}) \right]^2 + \frac{\sigma^2}{k} \end{aligned}$$



Selecting k is a *bias-variance* tradeoff: decreasing k increases the variance, but decreases the bias . . . and vice-versa.