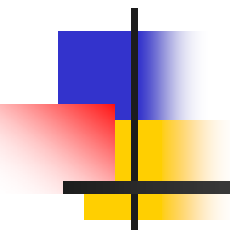


Predictive Data Mining

- Statistical Methods



Wei Jiang
Department of Systems Engineering & Engineering
Management
Stevens Institute of Technology

Predictive Data Mining – Regression Models

- Training data set: several features and outcome
- Build a learner based on training data sets
 - Capture important features of the **relationship between** a (set of) **variable(s) and** one or more **responses**
- Predict the future unseen outcome from seen features of data
- Many models are of the form

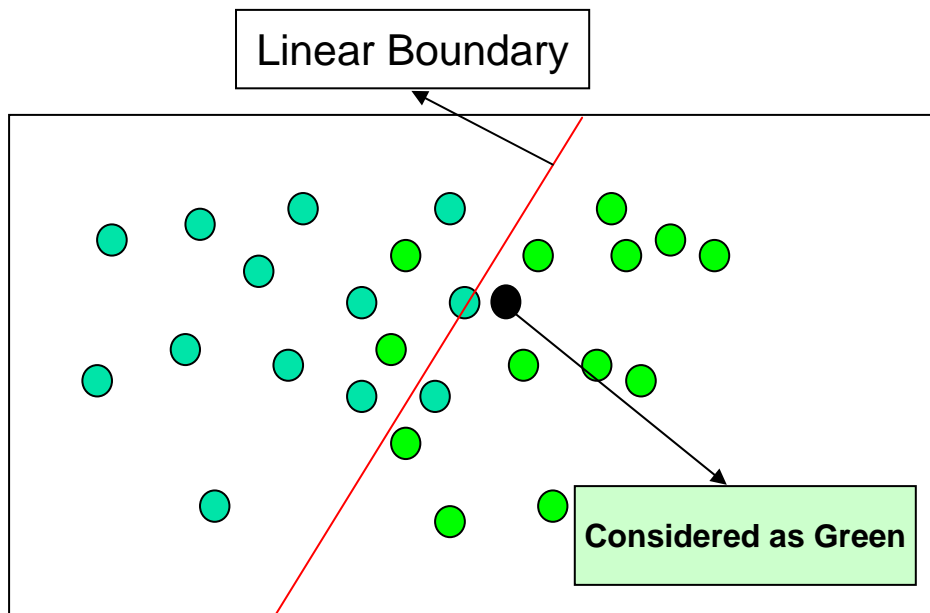
$$g(Y) = f(\underline{x}) + \text{error}$$

- **Differences** in the form of g , f and distributional assumptions about the error term

Two Fundamental Approaches – Global Models

❖ Linear regression models

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



➤ *Linear model*

$$f(x) \approx x^T \beta$$

➤ *Stable, smooth*

➤ *Low variance, high bias*

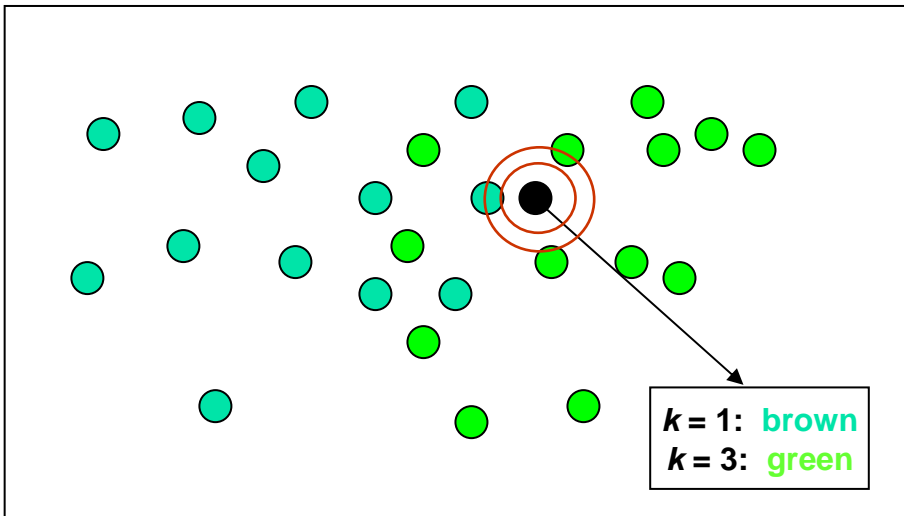
➤ *But, the true function might not be linear!*

Two Fundamental Approaches

– Local Models

- ❖ Nearest neighbor method: majority vote within the k nearest neighbors

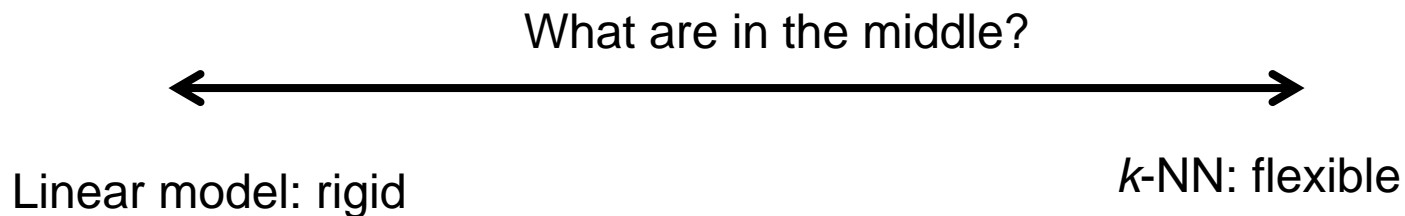
$$\hat{Y}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i$$



- *k-Nearest neighbor*
 $\hat{f}(x) = Ave(y_i | x_i \in N_k(x))$
- *If $N, k \rightarrow \infty, k/N \rightarrow 0$*
 $\hat{f}(x) \rightarrow E(Y | X = x)$
- *High variance, low bias*
- *Insufficient samples!*
- *Curse of dimensionality!*
- *Unstable, wiggly*

Beyond Linear Model and k -NN

- Linear basis expansions
- Roughness penalty and Bayesian methods
- Kernel methods and local regression
- Basis functions and dictionary methods



Non-linear models, linear in parameters

- We can add additional polynomial terms in our equations, e.g., all “2nd order” terms

$$f(\underline{x} ; \underline{\theta}) = \alpha_0 + \sum \alpha_j x_j + \sum \beta_{ij} x_i x_j$$

- Note that it is a non-linear functional form, but it is linear in the parameters (so still referred to as “linear regression”)
 - We can just treat the $x_i x_j$ terms as additional fixed inputs
 - In fact we can add in any non-linear input functions!, e.g.

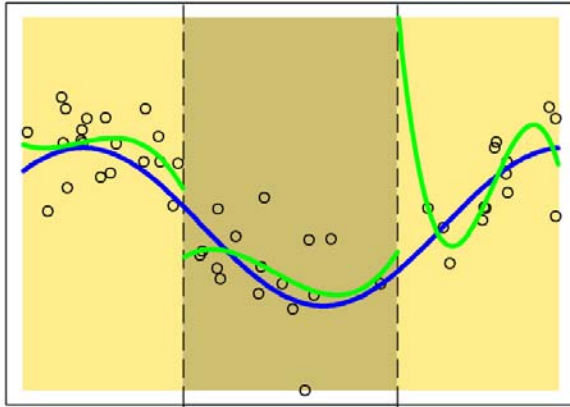
$$f(\underline{x} ; \underline{\theta}) = \alpha_0 + \sum \alpha_j f_j(\underline{x})$$

Comments:

- Exact same linear algebra for optimization (same math)
- size of model space has now exploded -> very large search problem

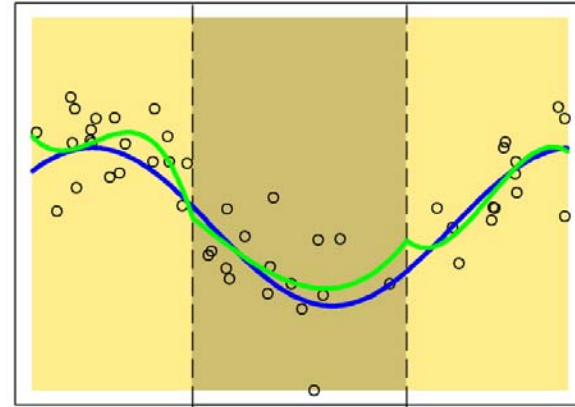
Cubic Spline

Discontinuous



ξ_1 ξ_2

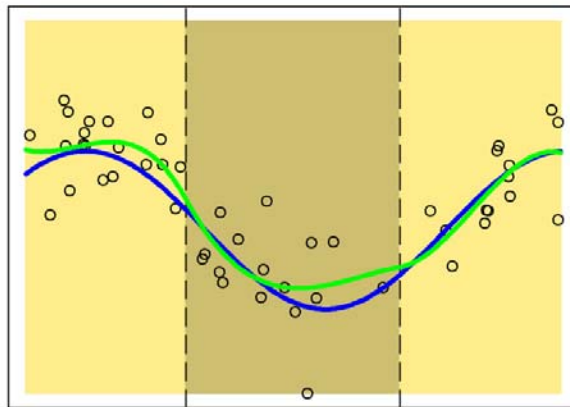
Continuous



ξ_1 ξ_2

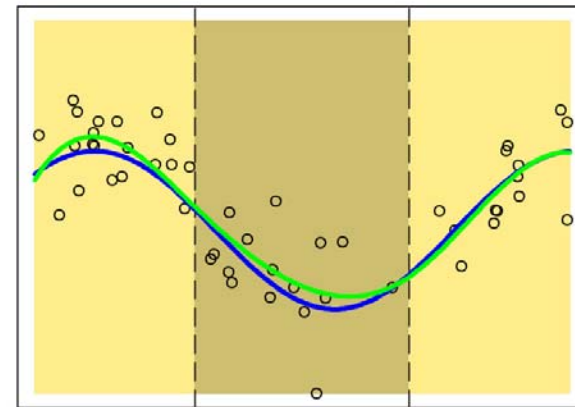
Is it beautiful?

Continuous First Derivative

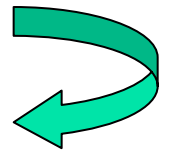


ξ_1 ξ_2

Continuous Second Derivative



ξ_1 ξ_2



Properties of Cubic Spline

Localization – changing the data at a point only changes the interpolant on the two adjacent subintervals

Easy “synthesis” - since the interpolating function is a finite linear combination with the data as the coefficients it is easy to construct

Easy “analysis” - if we are given a cubic splines function we can find the coefficients by evaluating function and its derivative at the nodes

Smoothness – good but not great. We have continuous first derivatives, but generally discontinuous second derivatives.

Easy implementation - it is straight forward to program a computer to set up and evaluate cubic splines efficiently.

Smoothing Spline

- Regression spline: we choose fixed-knots beforehand
- Problem: how can we **automatically** choose these knots to create a good regression spline?
- Smoothing spline comes!
- Complexity of fit is controlled by regularization

$$f = \arg \min_{f \in \text{Sobolev Space}} \text{RSS}(f, \lambda)$$

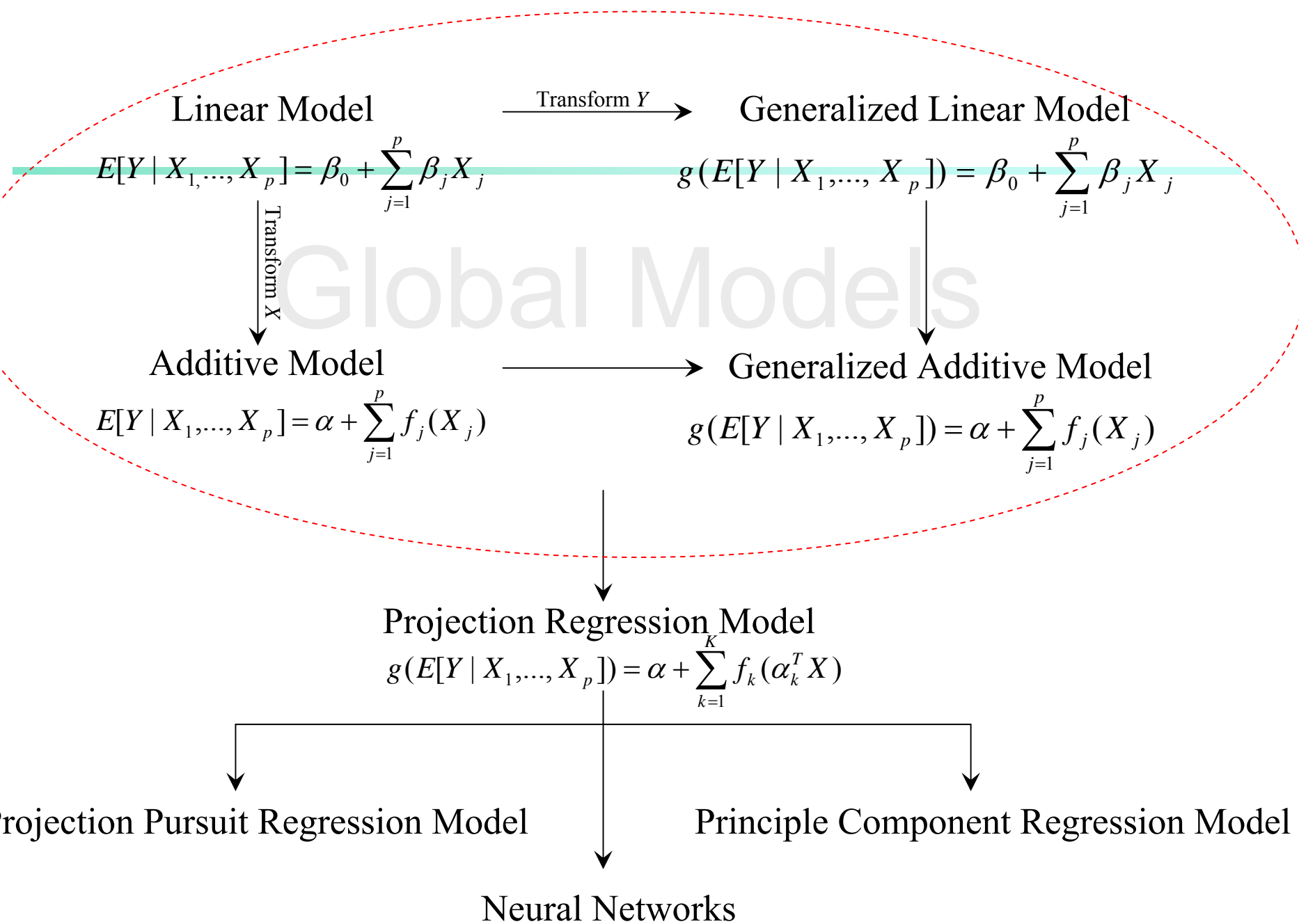
$$= \arg \min_{f \in \text{Sobolev Space}} \sum_{i=1}^N \{y_i - f(x_i)\}^2 + \lambda \int \{f''(t)\}^2 dt$$

$\lambda=0$, f can be any function that interpolates data

$\lambda = \infty$, f is simple least squares line fit

Global Regression Models Overview

- Linear Regression Models
- Generalized Linear Models
 - Logistic Regression
- Nonlinear Regression Models
 - Additive Models
 - Generalized Additive Models
 - Adaptive Models
 - Spline
 - MARS
 - Trees
- Projection Regression Models
 - Projection Pursuit Regression
 - Principle Component Regression & PLS
 - Neural Networks



Other non-linear models

- Memory-based models

$$y' = \sum w_{(x',x)} y, \quad \text{where } y\text{'s are from the training data}$$

$w_{(x',x)}$ = function of distance of x from x'

- Local linear regression

$$y' = \alpha_0 + \sum \alpha_j x_j, \quad \text{where the alpha's are fit at prediction time just to the } (y,x) \text{ pairs that are close to } x'$$

- Local likelihood models

$l(y', x^T \beta(x'))$, suitable for generalized linear models and linear covariates

Locally Weighted Regression

- Construct an explicit approximation to f over a local region surrounding query instance x_q
- Locally weighted linear regression:
 - The target function f is approximated near x_q using the linear function:

$$\hat{f}(x) = w_0 + w_1 a_1(x) + \dots + w_n a_n(x)$$

- minimize the squared error: distance-decreasing weight K

$$E(x_q) \equiv \frac{1}{2} \sum_{x \in k_nearest_neighbors_of_x_q} (f(x) - \hat{f}(x))^2 K(d(x_q, x))$$

- the gradient descent training rule:

$$\Delta w_j \equiv \eta \sum_{x \in k_nearest_neighbors_of_x_q} K(d(x_q, x)) ((f(x) - \hat{f}(x)) a_j(x))$$

- In most cases, the target function is approximated by a constant, linear, or quadratic function.

Local Regression Models Overview

- K-nearest neighbor models

- Memory-based models - kernel models

$$y' = \sum w_{(x',x)} y, \quad \text{where } y\text{'s are from the training data}$$

$w_{(x',x)}$ = function of distance of x from x'

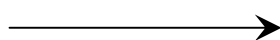
- Local linear regression

$$y' = \alpha_0 + \sum \alpha_j x_j, \quad \text{where the alpha's are fit at prediction time just to the } (y,x) \text{ pairs that are close to } x'$$

- Local likelihood models

$l(y', x^T \beta(x'))$, suitable for generalized linear models and linear covariates

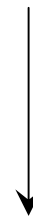
K -Nearest Neighbor



Kernel-Weighted Average

$$E[Y | x] = \text{Ave}(y_i | x_i \in N_k(x))$$

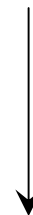
$$E[Y | x] = \frac{\sum_{i=1}^N K_\lambda(x_i, x) y_i}{\sum_{i=1}^N K_\lambda(x_i, x)}$$



Local Linear Regression

$$E[Y | x] = b(x)^T \beta(x)$$

$$\hat{\beta}(x) = \arg \min_{\beta} \sum_{i=1}^N K_\lambda(x, x_i) [y_i - b(x_i)^T \beta]^2$$

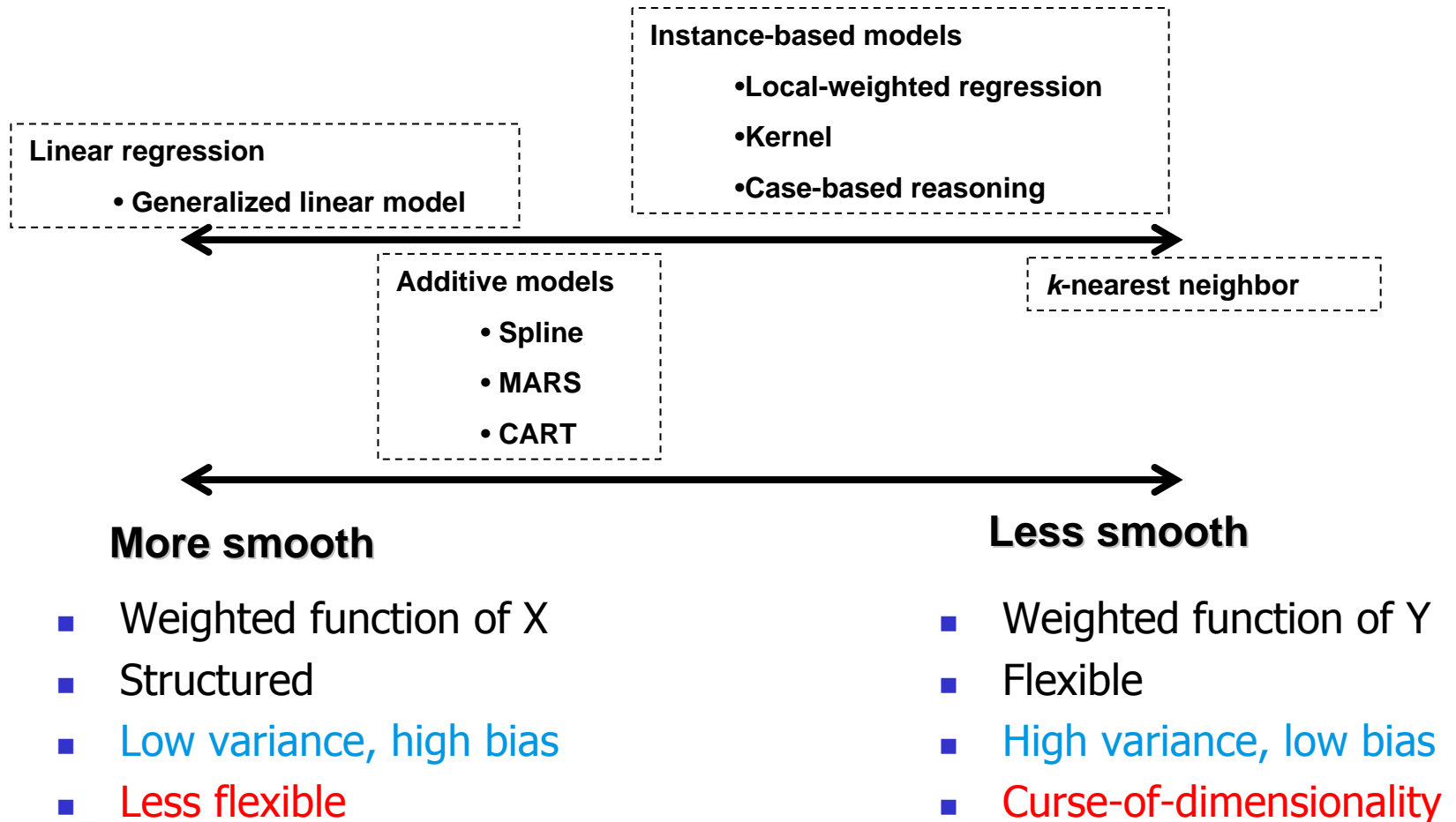


Local Likelihood

$$\hat{\beta} = \arg \max_{\beta} \sum_{i=1}^N K_\lambda(x, x_i) l(y_i, x_i^T \beta)^2$$

Local Models

Roadmaps



Do we have a standard to judge the models?

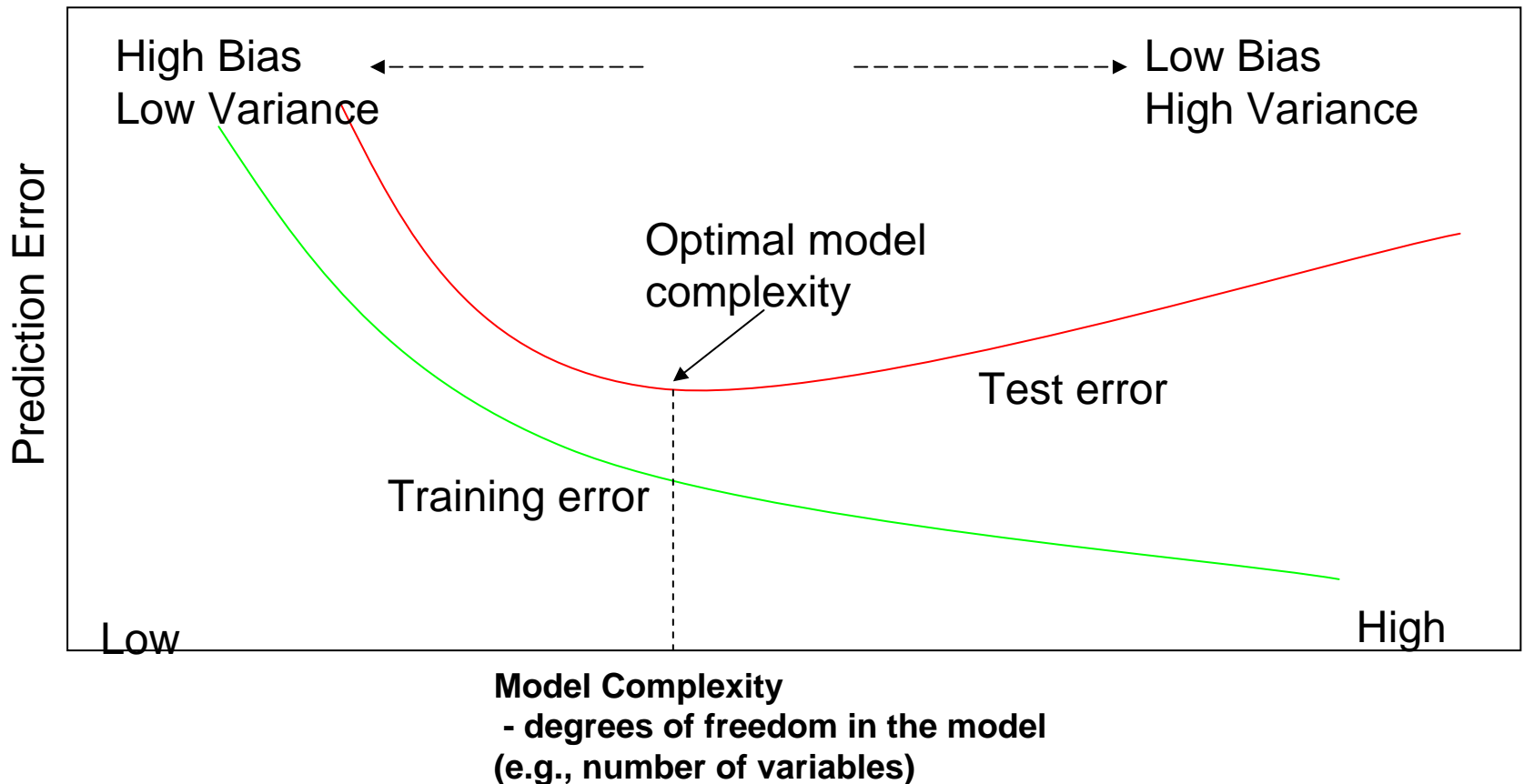
- Two different (but related) issues here
 - Finding the function f that minimizes $S(\underline{\theta})$ for future data
 - Model complexity
 - Getting a good estimate of $S(\underline{\theta})$, using the chosen function, on future data,
 - e.g., we might have selected the best function f , but our estimate of its performance will be optimistically biased if our estimate of the score uses any of the same data used to fit and select the model.

Defining what “best” means

- How do we measure “best”?
 - Best performance on the training data?
- Note:
 - Performance on the training data will in general be optimistic
- Alternatives:
 - Measure performance on a single validation set
 - Measure performance using multiple validation sets
 - Cross-validation
 - Add a penalty term to the score function that “corrects” for optimism
 - e.g., BIC

Curse of dimensionality and complexity

- **Exponential** growth of hyper-volume as a function of dimensionality
 - In high dimensions, sample size cannot satisfy the requirement
- Complexity

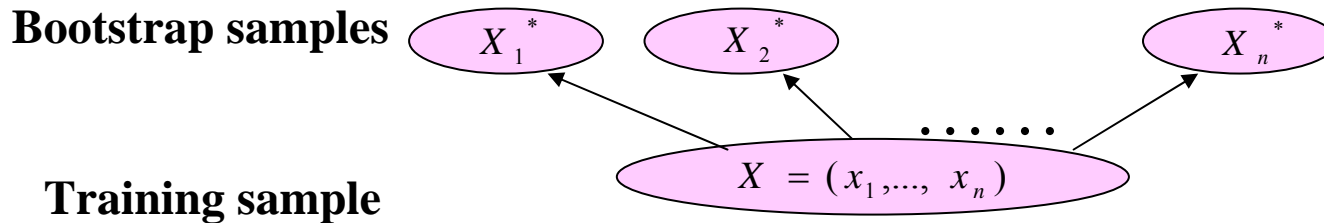


Estimating Prediction Error

- Data-rich
 - Partition: Train-Validation-Test “50%-25%-25%”
- Data-insufficient:
 - Analytical approaches
 - AIC, BIC, MDL, SRM
 - Efficient sample re-use approaches
 - Cross-validation
 - divide the data set into k subsamples
 - use $k-1$ subsamples as training data and one sub-sample as test data— k -fold cross-validation
 - for data set with moderate size
 - Bootstrapping (leave-one-out)
 - for small size data

Bootstrap

- Basic idea:
 - Randomly draw datasets with replacement from the training data
 - Each sample has the same size as the original training set

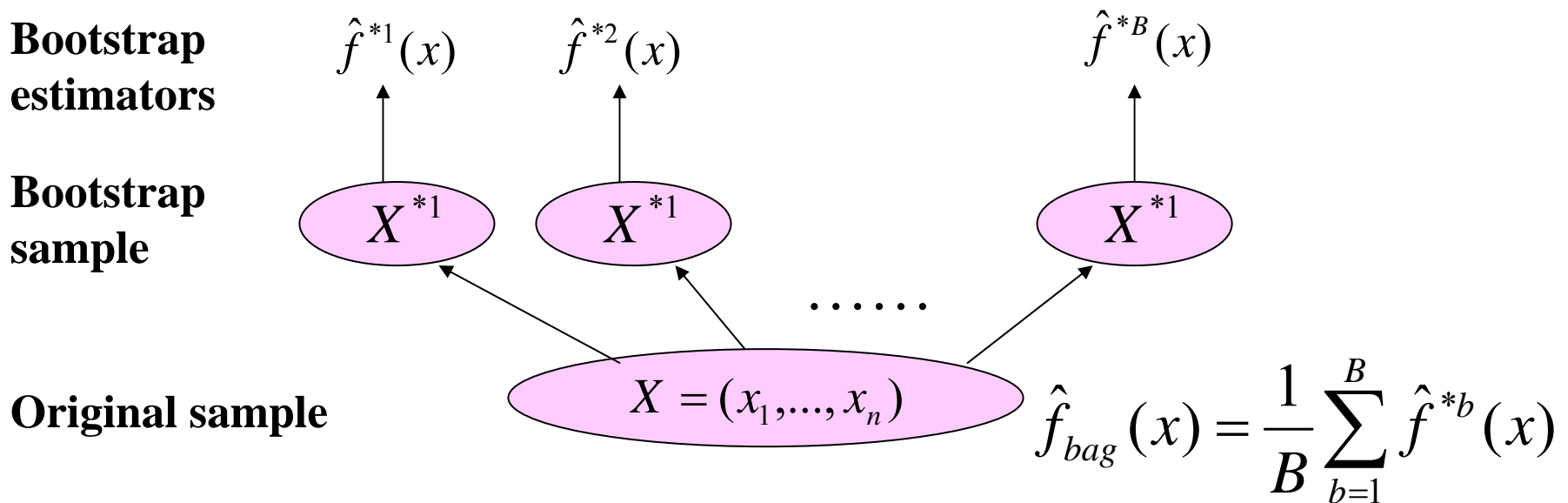


Bootstrap Variance Estimation

1. Draw $X_1^*, \dots, X_n^* \sim \hat{F}_n$
2. Compute $T_n^* = g(X_1^*, \dots, X_n^*)$
3. Repeat steps 1 and 2, B times, to get
4. Let
$$v_{boot} = \frac{1}{B} \sum_{b=1}^B (T_{n,b}^* - \frac{1}{B} \sum_{r=1}^B T_{n,r}^*)^2 \quad T_{n,1}^*, \dots, T_{n,B}^*$$

Bagging

- Bootstrap:
 - A way of assessing the accuracy of a parameter estimate or a prediction
- Bagging (Bootstrap Aggregating)
 - Use bootstrap samples to predict data classifiers



Classification becomes majority voting