Regression and model fitting

A Tuesday in 2024 Winter

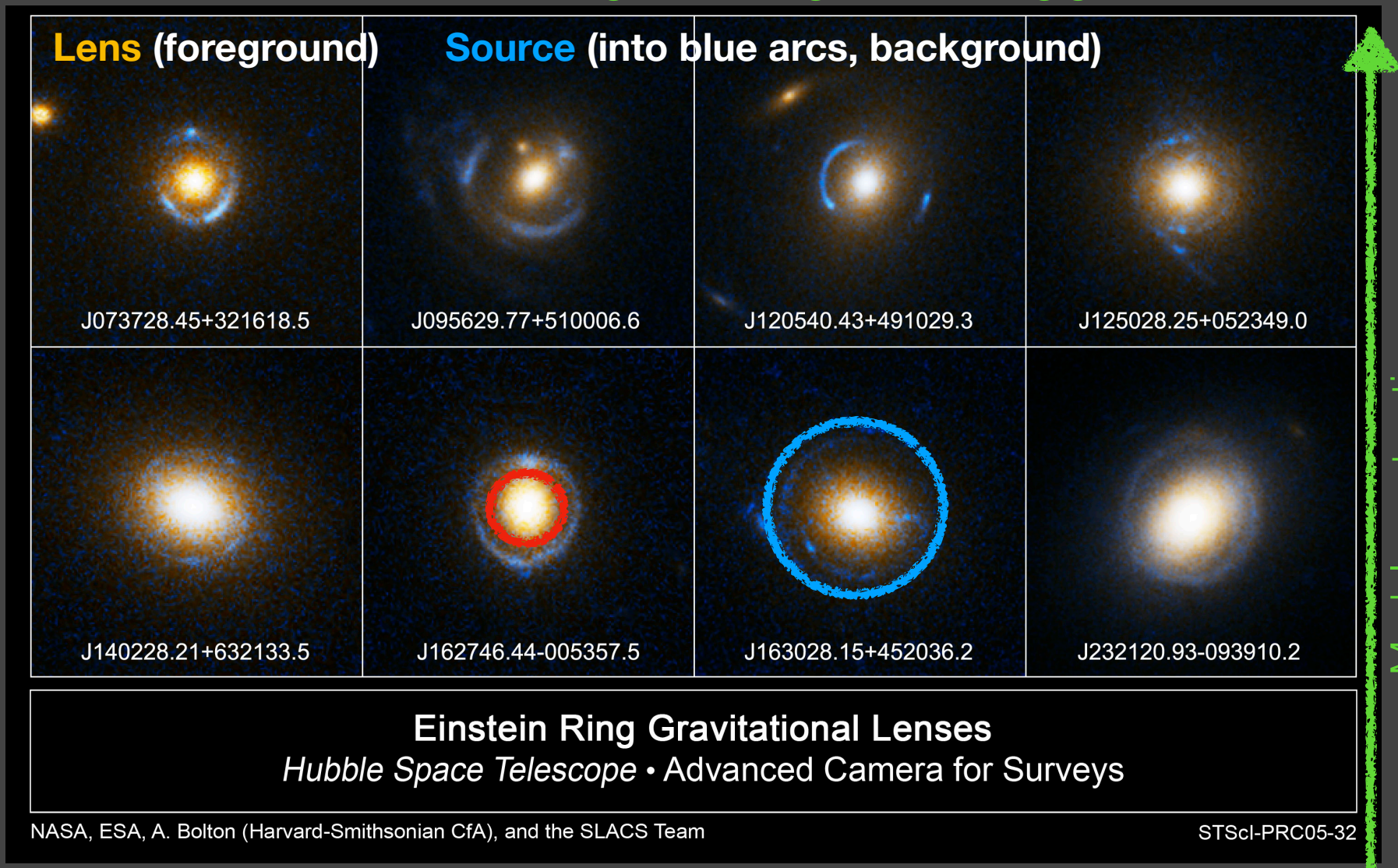
9:50-12:15

清華大學天文系

Department of Astronomy, Tsinghua University

What is model regression?

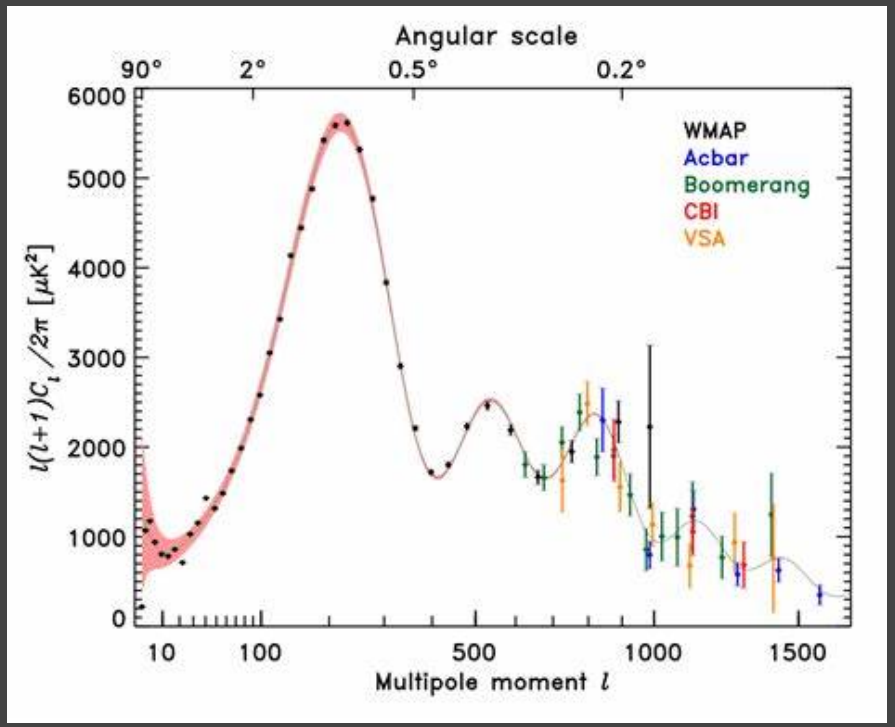
Estimate matter distribution of high redshift galaxies using gravitational lensing



1. Image configuration => *Einstein radius* <=> projected mass enclosed within
2. Spectroscopic measurements => *stellar velocity dispersion* => mass (<R)

What is model regression?

$y = \{y_i\}$



$x = \{x_i\}$

Regression is a statistical method that investigate relationships (*model*) between a dependent variable (also called "*outcome*" or "*response*" variable) and one or more independent variables (also called "*predictors*", "*covariates*","*explanatory variables*" or "*features*") and make predictions for the dependent variable y based on independent variables $\vec{x}$.

Topics

1. Framework - how does regression work?
2. Detailed content - regression models
3. Technique issue - regularization issue
4. Evaluating goodness - cross validation

Nutshell

How does regression work?

The best regression model $y = f(\vec{x} \mid \vec{\theta})$ *can be obtained through a Bayesian inference scheme (previous lectures):*

$$p(\vec{\theta} \mid \{\vec{x}_i, y_i\}, I) \propto p(\{y_i\} \mid \{\vec{x}_i\}, \vec{\theta}, I)\, p(\vec{\theta} \mid I)$$

Joint likelihood function $\propto \Pi_i^N p(y_i \mid \vec{x}_i, \vec{\theta}, I)$ — **Prior**

Data likelihood for each data point, ultimately is an error function:

$$p(y_i \mid \vec{x}_i, \vec{\theta}, I) = e(y_i \mid y = f(\vec{x}_i \mid \vec{\theta}))$$

Error function

The error function describes the probability to observe y_i *for* $\vec{x}_i$ *given regression model* $f(\vec{x} \mid \vec{\theta})$

Regression model/function:

For the i_{th} data point, infer the expectation value of the dependent variable $y(\vec{x}_i)$ based on a given model $f(\vec{x} \mid \vec{\theta})$ for independent variable $\vec{x}_i$, where $\vec{\theta}$ is model parameter, also referred to as regression coefficients.

Nutshell

How does regression work?

The best regression model $y = f(\vec{x} \mid \vec{\theta})$ can be obtained through a Bayesian inference scheme (previous lectures):

$$p(\vec{\theta} \mid \{\vec{x}_i, y_i\}, I) \propto p(\{y_i\} \mid \{\vec{x}_i\}, \vec{\theta}, I)\, p(\vec{\theta} \mid I)$$

Joint likelihood function $\propto \Pi_i^N p(y_i \mid \vec{x}_i, \vec{\theta}, I)$ — **Prior**

Data likelihood for each data point, ultimately is an error function:

$$p(y_i \mid \vec{x}_i, \vec{\theta}, I) = e(y_i \mid y = f(\vec{x}_i \mid \vec{\theta}))$$

Error function

Regression model/function

What does the error function look like?!

No matter how complicated your model is, end of the day, the likelihood function is all simple and basic! It is based on the error behavior!

Typical error behaviors for likelihood and the conjugate prior:

1. Gaussian (large number statistics) — Gaussian prior
2. Poisson (small number statistics) — Gamma prior
3. Binomial (success rate, “yes” or “no”) — Beta prior

Error function

Assuming *Gaussian* error $\mathcal{N}(f(x_i), \sigma_i^2)$ for each x_i with *known* σ_i

Data likelihood - error function:

$$p(y_i \mid \vec{x_i}, \vec{\theta}, I) = e(y_i \mid y = f(\vec{x_i} \mid \vec{\theta})) = \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left(\frac{-[y_i - f(\vec{x_i} \mid \vec{\theta})]^2}{2\sigma_i^2}\right)$$

Assuming flat prior for $p(\vec{\theta} \mid I)$:

$$L_p = \ln[p(\vec{\theta} \mid \{\vec{x_i}, y_i\}, I)] = \text{constant} - \Sigma_{i=1}^{N} \frac{[y_i - f(\vec{x_i} \mid \vec{\theta})]^2}{2\sigma_i^2}$$

Maximize Gaussian likelihood

Loss function

L_2 norm (*sensitive to outliers!!*)

Minimize *cost* function -> Least square estimate

Gauss (1777-1855), showed the equivalence between maximizing Gaussian likelihood estimate and linear least square estimate (by *Legendre*) — which is the *minimum variance unbiased estimator* (under CLT, by *Gauss-Markov*); and also applied the method to predict the orbit of Ceres (谷神星), in 1809.

Error function

A note on outliers and loss function

Remember that L_2 norm is more sensitive to outliers, we can also choose error function $e(y_i | y = f(\vec{x_i} | \vec{\theta}))$ with loss function that is less sensitive to outliers, e.g., L_1 norm, for which the minimization is essentially equal to finding the median!

Data likelihood follows an *exponential* error distribution for each x_i with *known* Δ_i:

$$p(y_i | \vec{x_i}, \theta, I) = \frac{1}{2\Delta_i} \exp\left(\frac{-|y_i - f(\vec{x_i} | \vec{\theta})|}{\Delta_i}\right)$$

Assuming flat prior for $p(\vec{\theta} | I)$:

$$L_p = \ln[p(\vec{\theta} | \{\vec{x_i}, y_i\}, I)] = \text{constant} - \Sigma_{i=1}^{N} \frac{|y_i - f(\vec{x_i} | \vec{\theta})|}{\Delta_i}$$

L_1 norm (*better cope with outliers!!*)

Loss function

Maximize likelihood estimate

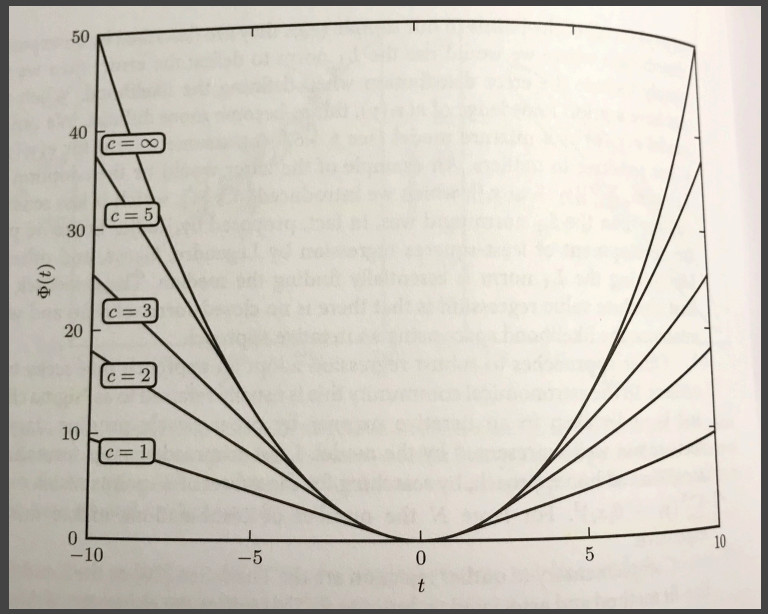
Minimize cost function - the L_1 norm

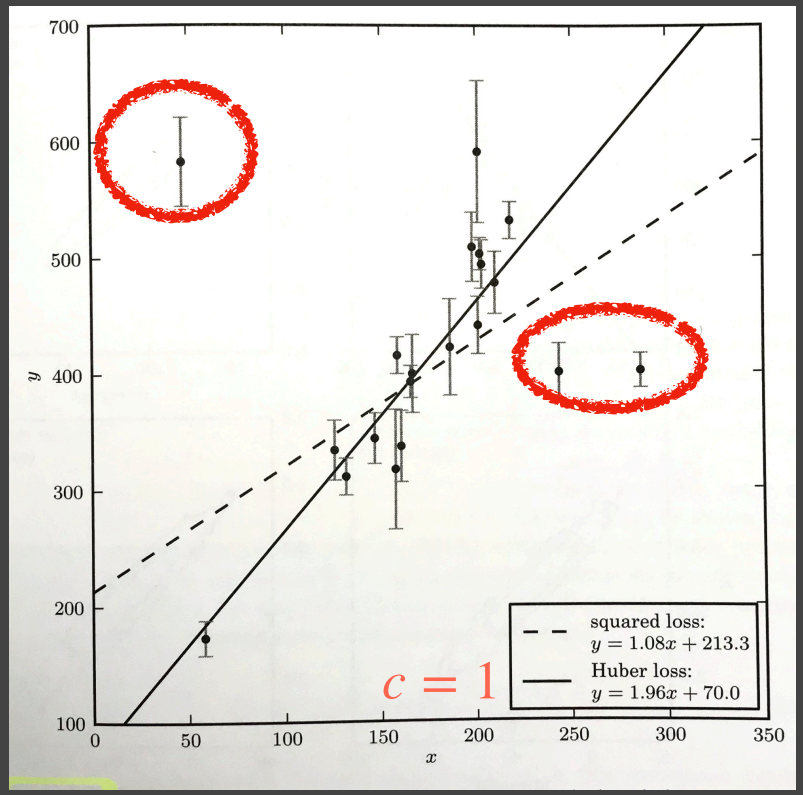
Alternatively:

- Outlier rejection algorithm: "sigma clipping" - pruning data points (in an iterative fashion) that are not well represented by the model.

- *Other M (maximum-likelihood like) estimator: Huber loss function*

Huber loss function





Huber estimator minimize the *cost* function $\Sigma_{i=1}^{N}\phi(t_i)$, where $\phi(t_i)$ is the *Huber loss* function:

$$\phi(t_i) = 1/2\, t_i^2 \quad \text{if } |t_i| \leqslant c$$

L_2 norm ~ Gaussian-like

$$\phi(t_i) = c\,|t_i| - 1/2\, c^2 \quad \text{if } |t_i| \geqslant c$$

L_1 norm ~ Exponential-like

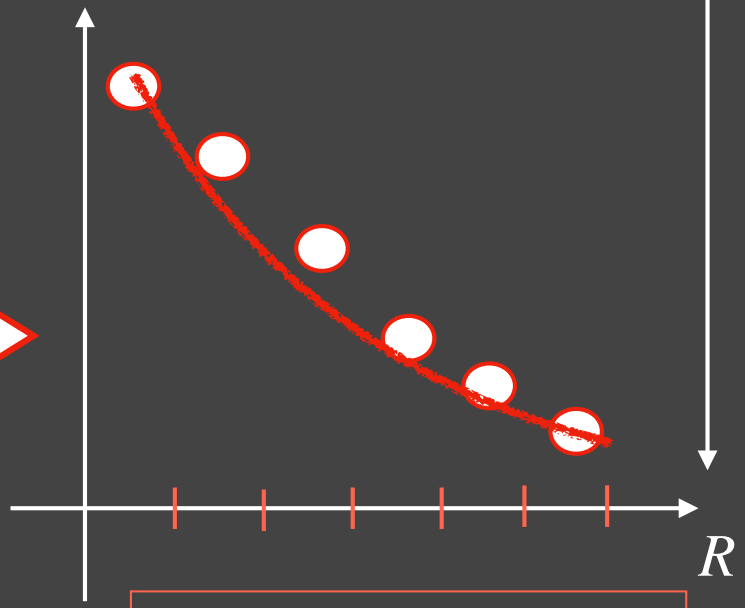
Where $t_i = [y_i - f(x_i|\,\vec{\theta})]/\sigma_i$. Note: the function is continues and differentiable!

Note #1: outliers can also be identified (with a probability) via Bayesian inference, assuming a distribution on outliers and probability of being an outlier for each data point.

Note #2: How to decide a proper c ? — cross-validation (see later)

A special case where $\{y_i\}$ are number counts in bins at $\{x_i\}$

$\Sigma(R) = N(R)/S(R)$



Each value in bin comes from number counting

How do we deciding the error function for number counts in each bin?

Error function

A special case where $\{y_i\}$ are number counts in bins at $\{x_i\}$

★ With *big number counts* ~ Gaussian distribution $y_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$,
where mean $\mu_i = f(x_i|\vec{\theta})$, std $\sigma_i = \mu_i^{1/2}$ for each x_i:

$$p(y_i|x_i, \vec{\theta}, I) = \frac{1}{\sqrt{2\pi f(x_i|\vec{\theta})}} \exp\left(\frac{-[y_i - f(x_i|\vec{\theta})]^2}{2f(x_i|\vec{\theta})}\right)$$

Data likelihood

Gaussian distribution!

★ With *small number counts* ~ Poisson distribution
$y_i \sim p(k|\mu_i)$, where $k = y_i$, $\mu_i = f(x_i|\vec{\theta})$ for each x_i:

$$p(y_i|x_i, \vec{\theta}, I) = \frac{f(x_i|\vec{\theta_i})^{y_i} \exp(-f(x_i|\vec{\theta_i})}{y_i!}$$

Data likelihood

Poisson distribution!

$$p(k|\mu) = \frac{\mu^k \exp(-\mu)}{k!}$$

***the importance of the error distribution*!**

The essence of Bayesian model inference and Bayesian regression lies in the *error* behavior! Without (assuming) a certain *error* distribution, one cannot perform any likelihood based estimates (either classic or Bayesian).

Model likelihood and uncertainty all come from the *errors*!

Nutshell

How does regression work?

The best regression model $y = f(\vec{x} \mid \vec{\theta})$ can be obtained through a Bayesian inference scheme (previous lectures):

$$p(\vec{\theta} \mid \{\vec{x}_i, y_i\}, I) \propto p(\{y_i\} \mid \{\vec{x}_i\}, \vec{\theta}, I)\, p(\vec{\theta} \mid I)$$

Likelihood function (error function) $\propto \Pi_i^N p(y_i \mid \vec{x}_i, \vec{\theta}, I)$ — **Prior**

Data likelihood for each data point:

$$p(y_i \mid \vec{x}_i, \vec{\theta}, I) = e(y_i \mid y = f(\vec{x}_i \mid \vec{\theta}))$$

Error function

The error function describes the probability to observe y_i for x_i given regression model $f(\vec{x} \mid \vec{\theta})$, what does the error function look like?!

No matter how complicated your model is, end of the day, the likelihood function is all simple and basic! It is based on the error behavior!

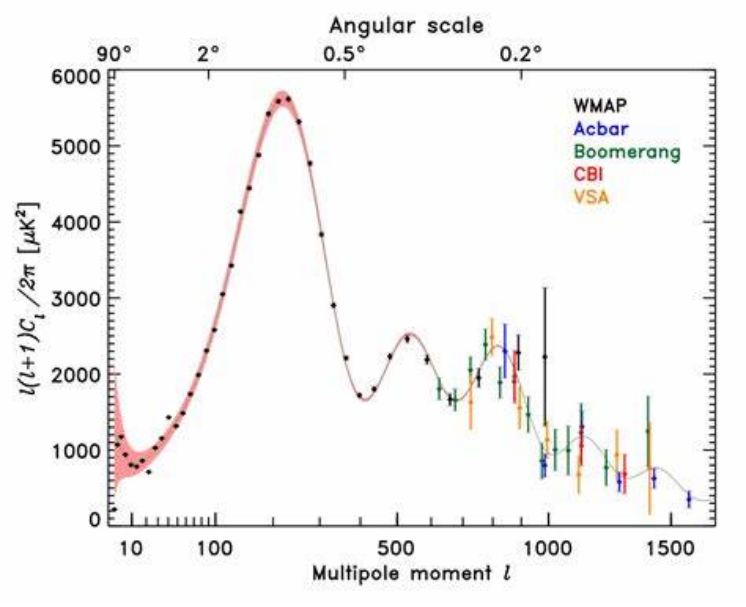
Regression model $y(\vec{x}) \equiv f(\vec{x} \mid \vec{\theta})$, what does regression model look like?!

Nutshell

How does regression work?

Regression model $y(\vec{x}) \equiv f(\vec{x} \mid \vec{\theta})$, what does regression model look like?!

what knowledge do you have to build a regression model?



Well, I understand structure growth in early Universe! I can build up a physical model to account for the data!

Non-linear Regression

Bayesian! MCMC!

Well, I have no clue about all the possible internal drives that govern the stock market. But I don't care. I observe, make predictions and can equally make money!!

This lecture!

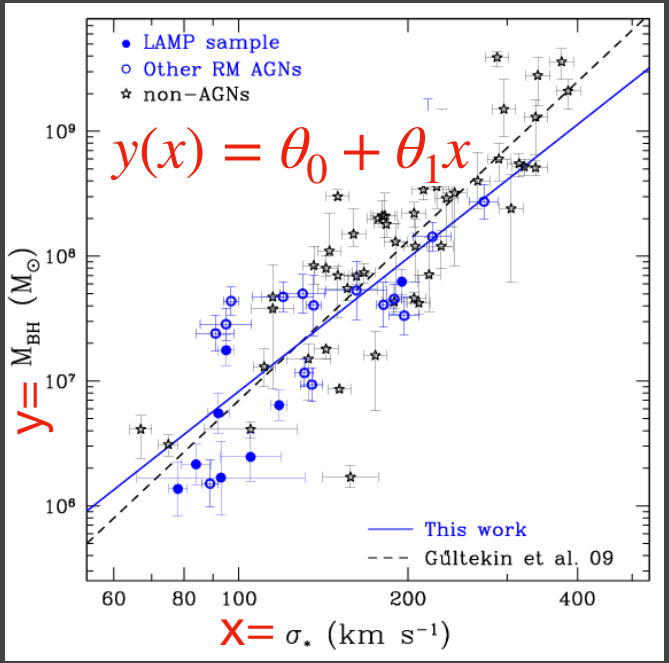
About regression model

- ★ Standard linear regression
- ★ Principal component regression
- ★ Gaussian process regression
- ★ Local (kernel) regression
- ★ More general non-linear regression

……

Regression model - Linear regression

Most common and widely-used regression model:



Linear regression

Model parameters, as linear weights for basis functions of x or for different features.

$$y(x) \equiv f(x \mid \vec{\theta}) = \Sigma_{p=1}^{k} \theta_p g_p(x)$$

Where $g_p(x)$ do not depend on any model parameter $\vec{\theta}$, but can be nonlinear functions of x.

$$y(x) \equiv f(x \mid \vec{\theta}) = \theta_0 + \theta_1 \sin(\theta_2 x)$$ **Non-linear regression**

$$y(x) \equiv f(x \mid \vec{\theta}) = \theta_0 + \sin\theta_1 \cos x + \cos\theta_2 \sin x$$

$$y(x) \equiv f(x \mid \vec{\theta}) = \theta_0 + \theta_1' \cos x + \theta_2' \sin x$$

$$y(x) \equiv f(x \mid \vec{\theta}) = \theta_0 \exp(\theta_1 x)$$

$$y'(x) \equiv \ln[y(x)] = \ln\theta_0' + \theta_1 x = \theta_0'' + \theta_1 x$$

Linear regression

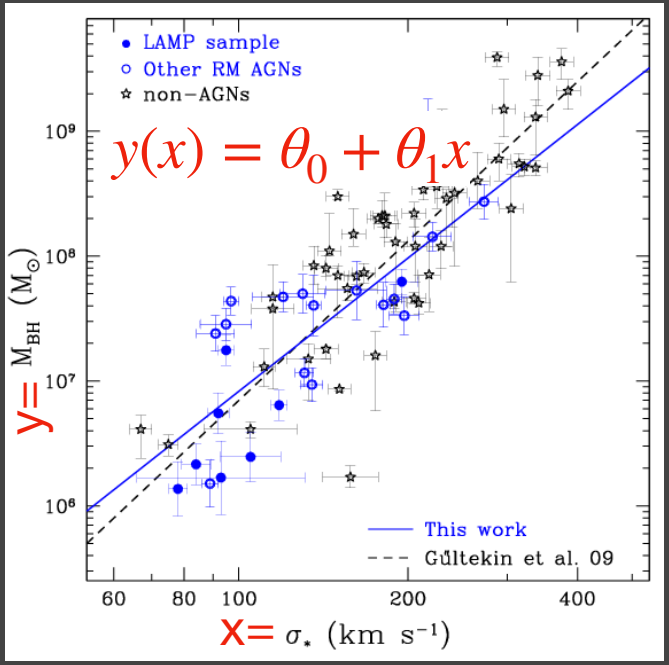
Linear for $\vec{\theta}$

Can be transformed into new parameters to become linear regression

Regression model - Linear regression

Most common and widely-used regression model:

Linear regression



Let's look at a very simple case where measurement uncertainty is on y:

i.e., the measurement (x_i, y_i) is generated by a model, e.g., assuming a simple linear case: $y(x) = \theta_0 + \theta_1 x$, with an error offset ϵ_i for the $i_{\rm th}$ data point, the observed value is then generated through: $y_i = \theta_0 + \theta_1 x_i + \epsilon_i$.

Now if assuming a Gaussian error behavior for each data point, i.e., $y_i \sim \mathcal{N}(y(x_i), \sigma_i^2)$, or equivalently $\epsilon_i = \left[y_i - y(x_i)\right] \sim \mathcal{N}(0, \sigma_i^2)$, where σ_i is the uncertainty for this data point. We can obtain best model parameter (θ_0, θ_1) for the linear regression model $y(x) = \theta_0 + \theta_1 x$, by maximizing the joint log-likelihood:

$$\ln L \propto -\Sigma_i^N \frac{[y_i - y(x_i)]^2}{2\sigma_i^2}.$$

Note: every measurement for a data point is essentially a realization of a Gaussian random process. The uncertainty in the denominator in the likelihood is a statistical quantity σ_i (due to both intrinsic scatter of the property and measurement uncertainty).

What if independent variables x also have measurement uncertainties?

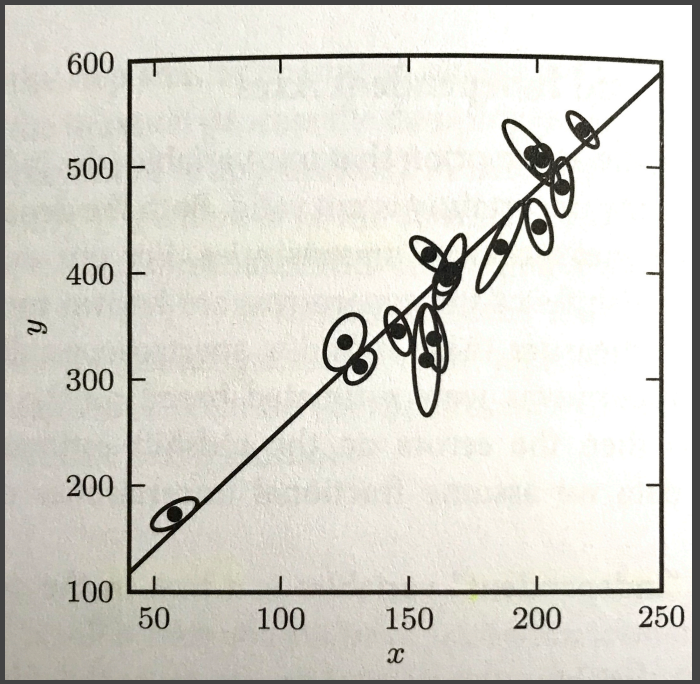
$y_i^* = \theta_0 + \theta_1 x_i^*$ pure model

$x_i = x_i^* + \delta_i$ real observations

$y_i = y_i^* + \epsilon_i$ real observations

$$==> y_i = \theta_0 + \theta_1(x_i - \delta_i) + \epsilon_i$$

errors on $\{x\}$ will translate into regression results!



Example: assuming the measurement errors on x and y follow bivariate *Gaussian*, i.e.,

$$p(\delta_i, \epsilon_i \,|\, \sigma_{x_i}, \sigma_{y_i}, \sigma_{x_i y_i}) = \frac{1}{2\pi\sigma_{x_i}\sigma_{y_i}\sqrt{1-\rho_{XY}^2}} \exp\left(-\frac{z_i^2}{2(1-\rho_{XY}^2)}\right)$$

where $z_i^2 = \frac{\delta_i^2}{\sigma_{x_i}^2} + \frac{\epsilon_i^2}{\sigma_{y_i}^2} - 2\rho_{XY}\frac{\delta_i\epsilon_i}{\sigma_{x_i}\sigma_{y_i}}$, $\rho_{x_i y_i} = \frac{\sigma_{x_i y_i}}{\sigma_{x_i}\sigma_{y_i}}$,

and covariance $\sigma_{x_i y_i} \equiv E[\delta_i \epsilon_i]$. When $\sigma_{x_i y_i} = 0$,

$p(\delta_i, \epsilon_i \,|\, \sigma_{x_i}, \sigma_{y_i}, \sigma_{x_i y_i}) \sim \mathcal{N}(0, \sigma_{x_i}^2)\mathcal{N}(0, \sigma_{y_i}^2)$.

The error model can be described by the covariance matrix:

$$C_i = \begin{bmatrix} \sigma_{x_i}^2 & \sigma_{x_i y_i} \\ \sigma_{x_i y_i} & \sigma_{y_i}^2 \end{bmatrix}$$, which can be known or unknown for each data point i. Beware that δ_i and ϵ_i are *unknown*!

What if independent variables x also have measurement uncertainties?

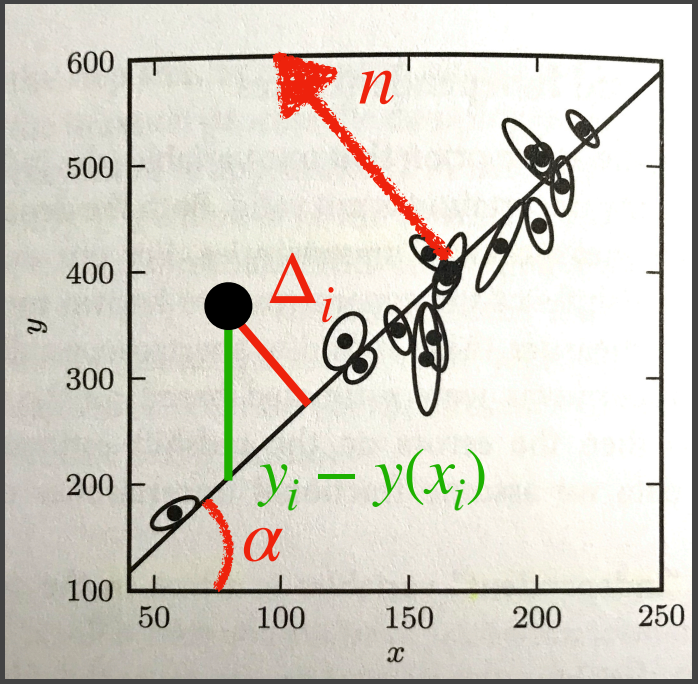
$y_i^* = \theta_0 + \theta_1 x_i^*$ pure model

$x_i = x_i^* + \delta_i$ real observations

$y_i = y_i^* + \epsilon_i$ real observations

$$==> y_i = \theta_0 + \theta_1(x_i - \delta_i) + \epsilon_i$$

errors on $\{x\}$ will translate into regression results!



For a straight line regression with slope θ_1:

Its *normal vector* $n = \begin{bmatrix} -\sin\alpha \\ \cos\alpha \end{bmatrix}$,

where $\alpha = \arctan(\theta_1)$.

The distance between a point and the line is given by:

$$\Delta_i = n^T z_i - \theta_0 \cos\alpha \qquad \text{where } z_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix}$$

Project the covariance matrix Σ_i to the *normal vector space* and obtain a total variance of $S_i^2 = n^T \Sigma_i n$

The log-likelihood (to be maximized) then becomes:

$$\ln L = -\Sigma_i^N \frac{\Delta_i^2}{2S_i^2},$$

(instead of $\ln L' = -\Sigma_i^N \frac{[y_i - y(x_i)]^2}{2\sigma_y^2}$).

Regression model - Linear regression

For the simple example case above, where we have N data points; for every point the independent variable x lives in *1d* space (e.g., galaxy luminosity); the model has $k = 2$ parameters θ_0 and θ_1 to predict y (e.g., galaxy stellar mass).

Write in Matrix form

$$Y = M\theta + \text{error}$$

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$

Model parameters for *simple linear regression*

$$\begin{aligned} y_0 &= \theta_0 + \theta_1 x_0 + \epsilon_0 \\ y_1 &= \theta_0 + \theta_1 x_1 + \epsilon_1 \\ &\cdots\cdots \\ y_{N-1} &= \theta_0 + \theta_1 x_{N-1} + \epsilon_{N-1} \end{aligned}$$

The # of parameters k determines the # of columns in the Design Matrix M; the # of total data constraints N determine the # of raws in M:

$$Y = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{N-1} \end{bmatrix} \qquad M = \begin{bmatrix} 1 & x_0 \\ 1 & x_1 \\ \vdots & \\ 1 & x_{N-1} \end{bmatrix}$$

Data constraints

$$C = \begin{bmatrix} \sigma_0^2 & & 0 \\ & \sigma_1^2 & \\ & .. & \\ 0 & & \sigma_{N-1}^2 \end{bmatrix}$$

Covariance matrix for errors on Y

Regression model - Linear regression

Write in Matrix form

$$Y = M\theta + \text{error}$$

Now instead of only 2 model parameters, there are many! e.g., k_{th}-order polynomial function of x. Note: x still lives in *1d* space (e.g., galaxy luminosity), and model still is linear for θ.

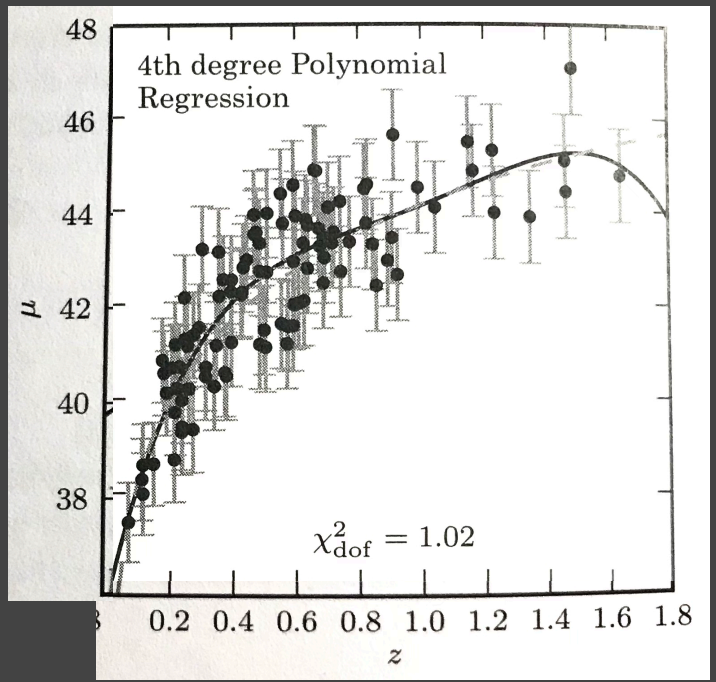
with Polynomial & Basis Function Regression

Multiple linear regression

$$y_i = \theta_0 + \theta_1 x_i + \theta_2 x_i^2 + \ldots + \theta_k x_i^k + \epsilon_i$$

Essentially => Taylor expansion

Note: i indicates the i^{th} data point (x_i, y_i).



$$Y = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{N-1} \end{bmatrix} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_k \end{bmatrix}$$

$$M = \begin{bmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^k \\ 1 & x_1 & x_1^2 & \ldots & x_1^k \\ \ldots & & & & \\ 1 & x_{N-1} & x_{N-1}^2 & \ldots & x_{N-1}^k \end{bmatrix}$$

Regression model - Linear regression

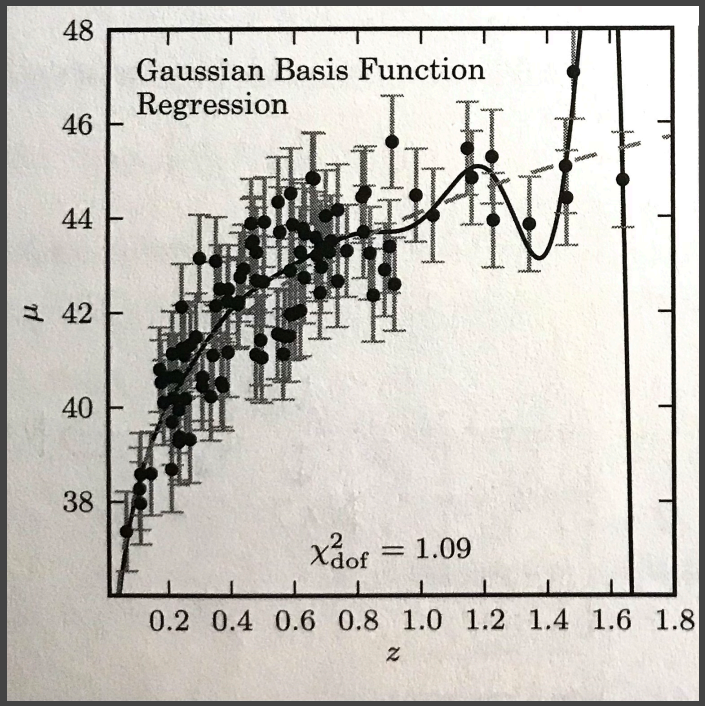
Caution must be made at boundaries of data when applying basis function fitting…

Write in Matrix form

$$Y = M\theta + \text{error}$$

Other basic functions can be: *Gaussian*, *Trigonometric* function etc…e.g.,

$$y_i = \theta_0 + \theta_1 \exp\Big[-\frac{(x_i - x_1^G)^2}{2\sigma_G^2}\Big] + \theta_2 \exp\Big[-\frac{(x_i - x_2^G)^2}{2\sigma_G^2}\Big] + \ldots + \theta_k \exp\Big[-\frac{(x_i - x_k^G)^2}{2\sigma_G^2}\Big] + \epsilon_i$$



Note: $x^{\rm G}_{j=1,2,..k}$ and σ_G are not model parameters, but are pre-fixed for each Gaussian component.

It is $\{\theta\}$ — the Gaussian amplitudes that are essentially the goal of model regression.

15 Gaussians with $x^{\rm G}_{j=1,2,..k}$ evenly spaced between z=0 and 2, with widths of 0.14.

Regression model - Linear regression

Write in Matrix form

$$Y = M\theta + \text{error}$$

Now instead of linear function of x living in *1d* space (e.g., galaxy luminosity), we have linear regression depending on $\vec{x}$ living in *k-dimensional* feature space (e.g., *k*=2, luminosity and star-formation rate).

$$y_i = \theta_0 + \theta_1 x_{i1} + \theta_2 x_{i2} + \ldots + \theta_k x_{i,k} + \epsilon_i$$

Note: i indicates the i^{th} data point $(\vec{x_i}, y_i)$, where the predictor variable $\vec{x_i}$ is a vector in *k*-dimensional feature space.

Multivariate linear Regression

A hyperplane instead of a curve

Feature 1, Feature 2, ….. Feature k

$$Y = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{N-1} \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_k \end{bmatrix} \quad M = \begin{bmatrix} 1 & x_{01} & x_{02} & \cdots & x_{0,k} \\ 1 & x_{11} & x_{12} & \cdots & x_{1,k} \\ \cdots & & & & \\ 1 & x_{N-1,1} & x_{N-1,2} & \cdots & x_{N-1,k} \end{bmatrix}$$

Regression model - Linear regression

Assuming *Gaussian* error $\mathcal{N}(f(\vec{x}_i), \sigma_i^2)$ for each $\vec{x}_i$ with *known* σ_i

$$L_p = \ln[p(\vec{\theta} \mid \{\vec{x}_i, y_i\}, I)] = \text{constant} - \Sigma_{i=1}^{N} \frac{[y_i - f(\vec{x}_i \mid \vec{\theta})]^2}{2\sigma_i^2}$$

Maximize Gaussian likelihood estimate

Minimize cost function -> Least square estimate

i.e., minimizing $(Y - M\theta)^T C^{-1} (Y - M\theta)$, where $C = \begin{bmatrix} \sigma_0^2 & \sigma_{01} & \cdots & \sigma_{0,N-1} \\ \sigma_{10} & \sigma_1^2 & \cdots & \\ \cdots & \cdots & \cdots & \\ \sigma_{N-1,0} & & & \sigma_{N-1}^2 \end{bmatrix}$

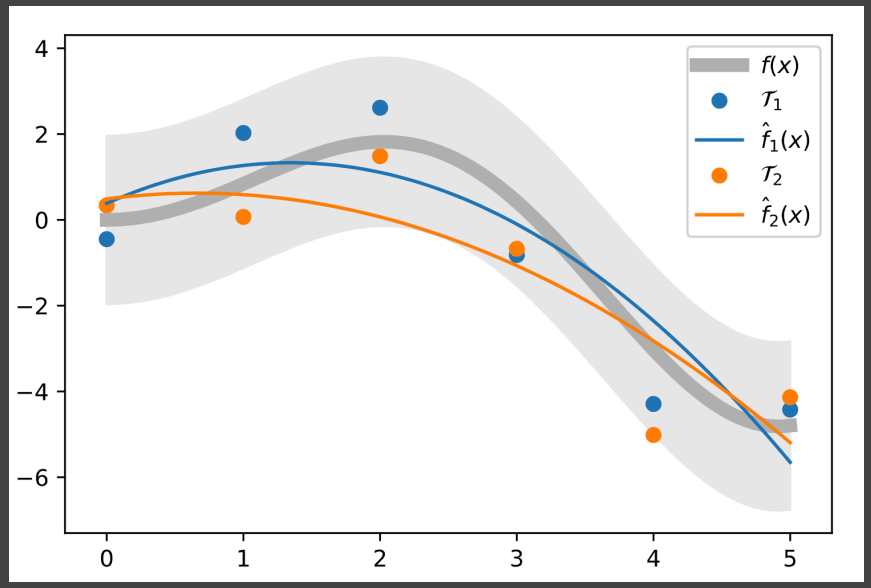
the covariance matrix C even allow for covariant data with non-zero off-diagonal elements.

- For all cases above, we have solution: $\theta = (M^T C^{-1} M)^{-1} (M^T C^{-1} Y)$
- Parameter Uncertainty Matrix Σ_θ is given below:

$$\Sigma_\theta \equiv \begin{bmatrix} \sigma_{\theta_0}^2 & \sigma_{\theta_0\theta_1} & \sigma_{\theta_0\theta_2} & \cdots \\ \sigma_{\theta_0\theta_1} & \sigma_{\theta_1}^2 & \sigma_{\theta_1\theta_2} & \cdots \\ \sigma_{\theta_0\theta_2} & \sigma_{\theta_1\theta_2} & \sigma_{\theta_2}^2 & \cdots \\ \cdots & \cdots & \cdots & \cdots \end{bmatrix} = [M^T C^{-1} M]^{-1}$$

Regularization

Datasets τ_1 and τ_2 … are generated by the same *true* model $t(x)$, but each time being a different realization with different noise:



The model regression even using model $t(x)$, will result $\hat{f}_1(\{x\}_{\tau_1})$ for a finite dataset τ_1, which will not be the same as $\hat{f}_2(\{x\}_{\tau_2})$ for dataset τ_2 and so on, due to two effects:

(1) due to *random noise* $\tau_1 \neq \tau_2$; and (2) discrete sampling in x.

As a result, when doing model regression using ant given model $f(x)$, we have:

***Model* Bias**: $\mathrm{Bias}(x) = \mathrm{E}_\tau[\hat{f}_\tau(x)] - t(x)$

— If *model bias* is large, $\mathrm{E}_\tau[\hat{f}_\tau(x)]$ cannot converge to $t(x)$, *model* $f(x)$ *has bias!*

***Model* Variance**: $\mathrm{Variance}(x) = \mathrm{E}_\tau([\hat{f}_\tau(x) - \mathrm{E}_\tau(\hat{f}_\tau(x))]^2)$

— scatter among regression model predictions using multiple realizations of measurements

***Model* Mean square error**: $\mathrm{MSE}(x) = \mathrm{E}_\tau([\hat{f}_\tau(x) - t(x)]^2)$

— (square) error in regression model predictions when compared to the ground-truth.

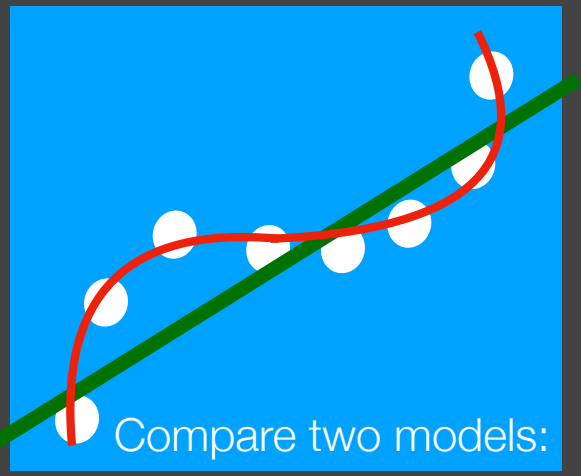
MSE = Variance + Bias^2

Best model estimate will minimize MSE!!

Regularization

Green: Simpler models may not fit all features in a given dataset (thus have bigger bias), however regression results using simple models over many realizations of measurements would be very similar thus have smaller variance.

Red: with increased model complexity through e.g., increasing parameter dimension k, we decrease model (apparent) bias. However, as $k \uparrow$, the model regression each time would also fit more and more the *specific noise feature* particular to that realization of dataset — Each regression result becomes more and more sensitive to that particular given dataset, thus model variance among different regression results becomes larger and larger.

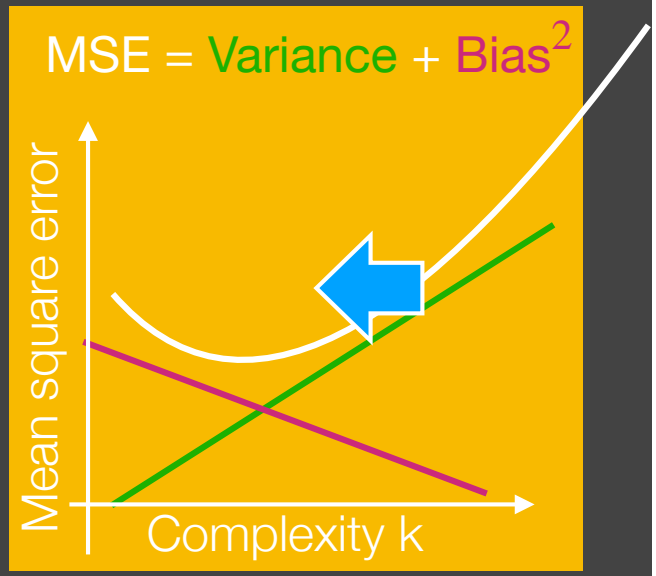


Bigger bias, smaller variance

Smaller bias, bigger variance

Increasing model complexity can surely decrease model bias. However, MSE in fact increases due to growing model variance for more complex models. Regularization is a process which helps to penalize and limit model complexity, thus would decrease model variance at cost of increasing model bias, such that:

$\text{MSE} = \text{Variance} + \text{Bias}^2$ *shall be restrained around the minimal against going towards higher complexity.*



Regularization

$$y_i = \theta_0 + \theta_1 x_i + \theta_2 x_i^2 + \ldots + \theta_k x_i^k + \epsilon_i$$

Assuming doing a k_{th}-order polynomial model regression, how to properly penalize model complexity?

From a Bayesian perspective, one way of doing regularization can be viewed as adding a Gaussian prior on $\vec{\theta}$ with $\mathcal{N}(0, \left(\frac{1}{\sqrt{\lambda}}\right)^2)$:

λ is regularization or smooth parameter

$$p(\vec{\theta} \mid \{\vec{x}_i, y_i\}, I) \propto p(\{y_i\} \mid \{\vec{x}_i\}, \vec{\theta}, I)\, p(\vec{\theta} \mid I)$$

$$p(\vec{\theta}, I) \propto \exp\left(\frac{-(\lambda \theta^T \theta)}{2}\right)$$

$$y_i = \theta_0 + \theta_1 x_i + \theta_2 x_i^2 + \ldots + \theta_k x_i^k + \epsilon_i$$

large λ penalize θ harder!

The larger λ is, the smaller $|\theta|^2$ shall be, which will prefer model with less polynomial order k.

Log posterior distribution: $\ln L_p = \ln L - \lambda |\theta|^2$ ***ridge regression***

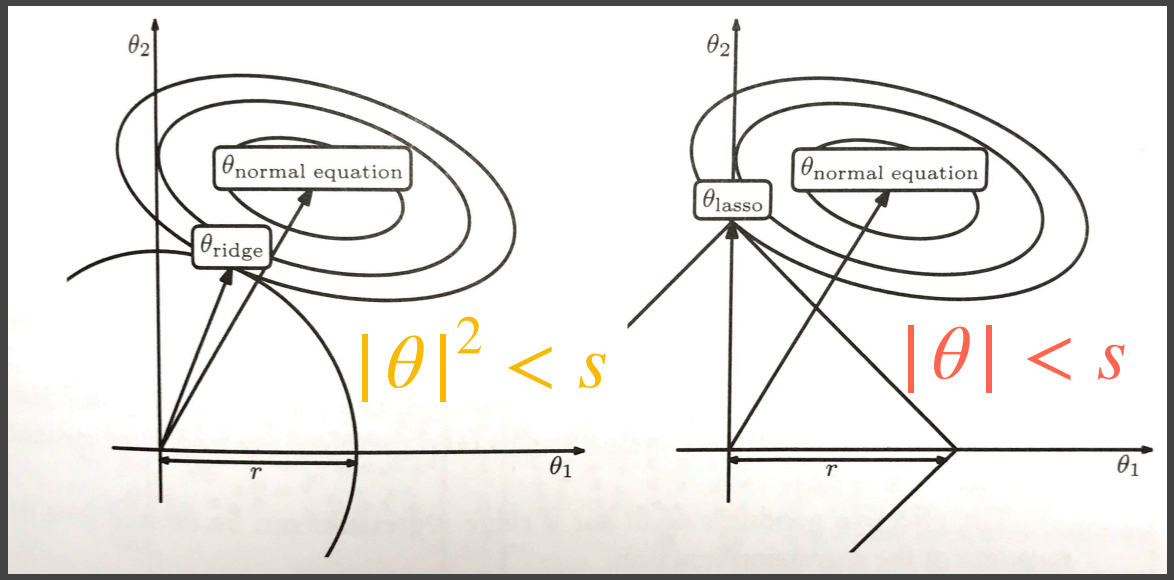
Regularization minimize: $(Y - M\theta)^T C^{-1} (Y - M\theta) + \lambda |\theta^T \theta|$

Least square estimate *due to regularization*

Solution for ridge regression: $\theta = (M^T C^{-1} M + \lambda I)^{-1} (M^T C^{-1} Y)$,

An advantage with λ is that even if $M^T C^{-1} M$ is singular, solution can still exist.

Regularization



ridge (Tikhonov) regularization ***LASSO regularization***

ridge regularization minimize: $(Y - \theta M)^T C^{-1} (Y - \theta M) + \lambda |\theta^T \theta|$

LASSO regularization minimize: $(Y - \theta M)^T C^{-1} (Y - \theta M) + \lambda |\theta|$

ridge regularizes model parameters less aggressively than LASSO.
the former has closed form solution, the latter does NOT.

How to decide the best λ?

Note, model parameter to be constrained is $\vec{\theta}$. Different models with different λ yield different regression performance. The best model is the one with λ that minimizes the cross-validation error (see later)!

Previously discussed: regularization to panelize higher model complexities

When model regression is carried out through data with
many features, there is also a certain kind of “regularization”
which panelizes feature of less importance!

Principal Component Regression

Linear regression

PCR regression

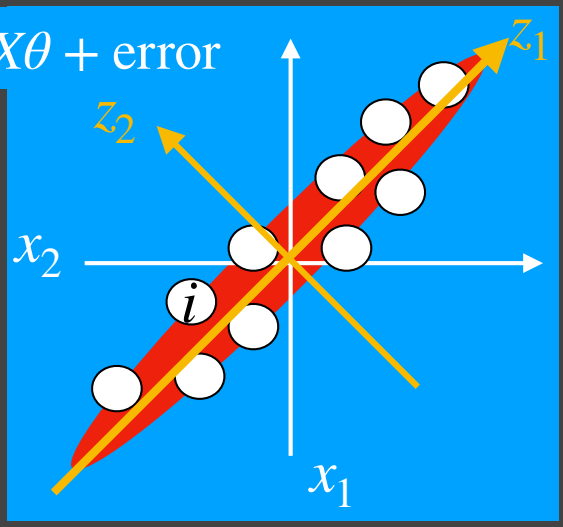
A special case in multivariate linear Regression

$$y_i = \theta_1 x_{i1} + \theta_2 x_{i2} + \ldots + \theta_k x_{ik} + \epsilon_i$$

$$X \equiv \left(X_1\right)\left(X_2\right)\left(\ldots\right)\left(X_k\right) \qquad X_j = \begin{bmatrix} x_{0j} \\ x_{1j} \\ \cdot \\ \cdot \\ x_{N-1,j} \end{bmatrix}$$

$X_{N\times k}$ is a $N \times k$ matrix,
with N data points and k features.

$Y = X\theta + \text{error}$



Data covariance matrix $C_X = X^T X$

Non-zero off-diagonal elements in C_X

When independent variables X_j are highly correlated in *k-dim* feature space, the matrix solutions before can be unstable.

Note: each $\overrightarrow{x_i}$ lives in *k-dimensional* feature space, assuming already re-centered to the mean $\overrightarrow{x}_{\text{mean}}$ of sample distribution.

We want to find an alternative space Z to project X, which guarantees minimal correlations, such that

$Z_{N\times k} = XR$ Where R is a $k \times k$ rotation matrix, $R^T R = I$.

$C_Z = Z^T Z = R^T X^T X R = R^T C_X R = \Sigma$, with Σ only has zero off-diagonal elements.

$$\Sigma = \begin{bmatrix} \lambda_1 & 0 & 0 & 0 \\ 0 & \lambda_2 & 0 & 0 \\ & & \cdots & \\ 0 & 0 & 0 & \lambda_k \end{bmatrix}$$

Principal Component Regression

Linear regression

PCR regression

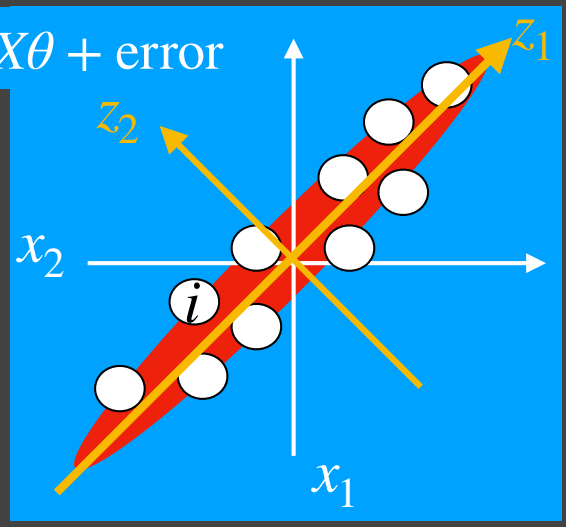
A special case in multivariate linear Regression

$$y_i = \theta_1 x_{i1} + \theta_2 x_{i2} + \ldots + \theta_k x_{ik} + \epsilon_i$$

$$X \equiv \Big(X_1\Big)\Big(X_2\Big)\Big(\ldots\Big)\Big(X_k\Big) \qquad X_j = \begin{bmatrix} x_{0j} \\ x_{1j} \\ \cdot \\ \cdot \\ x_{N-1,j} \end{bmatrix}$$

$X_{N\times k}$ is a $N \times k$ matrix,
with N data points and k features.

$Y = X\theta + \text{error}$



Data covariance matrix $C_X = X^T X$

Non-zero off-diagonal elements in C_X

When independent variables X_j are highly correlated in *k-dim* feature space, the matrix solutions before can be unstable.

Note: each $\overrightarrow{x_i}$ lives in *k-dimensional* feature space, assuming already re-centered to the mean $\overrightarrow{x}_{\text{mean}}$ of sample distribution.

We want to find an alternative space Z to project X, which guarantees minimal correlations, such that

Eigenvalue / singular value decomposition tells R and Σ are simply related via: $X^T X = R\Sigma R^T$, with

$$R \equiv \Big(R_1\Big)\Big(R_2\Big)\Big(\ldots\Big)\Big(R_k\Big) \qquad R_j = \begin{bmatrix} R_{j,1} \\ R_{j,2} \\ \cdot \\ R_{j,k} \end{bmatrix} \qquad \Sigma = \begin{bmatrix} \lambda_1 & 0 & 0 & 0 \\ 0 & \lambda_2 & 0 & 0 \\ & & \cdots & \\ 0 & 0 & 0 & \lambda_k \end{bmatrix}$$

With $\{R_j\}$ being the eigenvectors of $X^T X$, and $\{\lambda_j\}$ eigenvalues reflecting variance of data along orthogonal bases.

Principal Component Regression

Linear regression

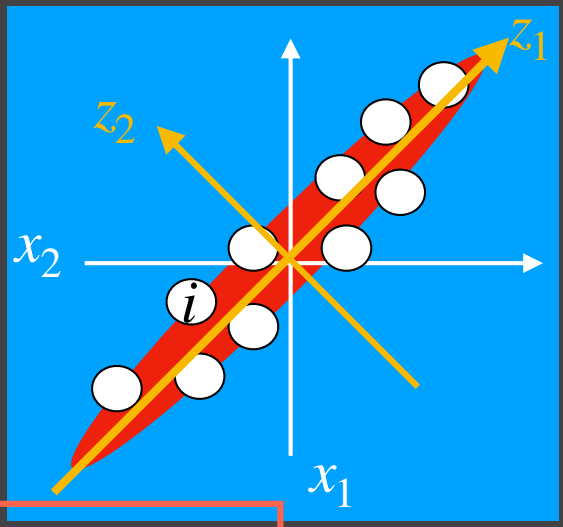
PCR regression

A special case in multivariate linear Regression

$$y_i = \theta_1 x_{i1} + \theta_2 x_{i2} + \ldots + \theta_k x_{ik} + \epsilon_i$$

$$X \equiv \left(X_1\right)\left(X_2\right)\left(\ldots\right)\left(X_k\right) \qquad X_j = \begin{bmatrix} x_{0j} \\ x_{1j} \\ \cdot \\ \cdot \\ x_{N-1,j} \end{bmatrix}$$

$X_{N\times k}$ is a $N \times k$ matrix,
with N data points and k features.



$$Z_{N\times k} = XR$$ Project independent variable to new orthogonal space!

$$Z \equiv \left(z_1\right)\left(z_2\right)\left(z_3\right)\left(\ldots\right)\left(z_k\right)$$

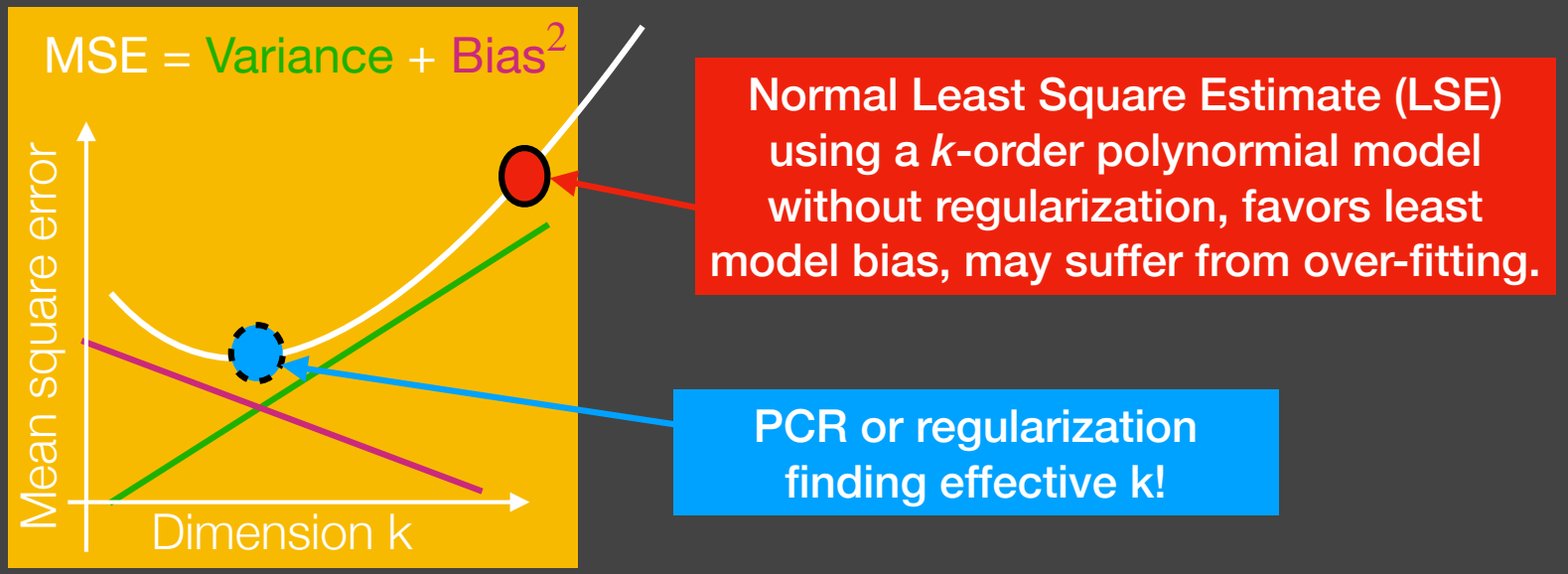
Note: some of the new principle features are more important than the others, e.g., $\lambda_1, \lambda_3, \lambda_k$ are large, and these "combined" features would dominate the variance in data.

PCR $Y = M_z \theta' + \text{error}$ (Apply standard linear regression)

To eliminate collinear correlation causing unstable solution, one can exclude principal components with smaller λ_i, e.g., if below a few per cent of average λ. Such a "truncation" is equivalent to applying a certain regularization.

Regularization vs. PCR

Both methods can reduce the effective parameter dimension space: regularization penalizes complexity from high-order basis functions, while PCR panelizes unimportant *principal* features in Z, thus eliminating strong collinear correlations in X feature space.



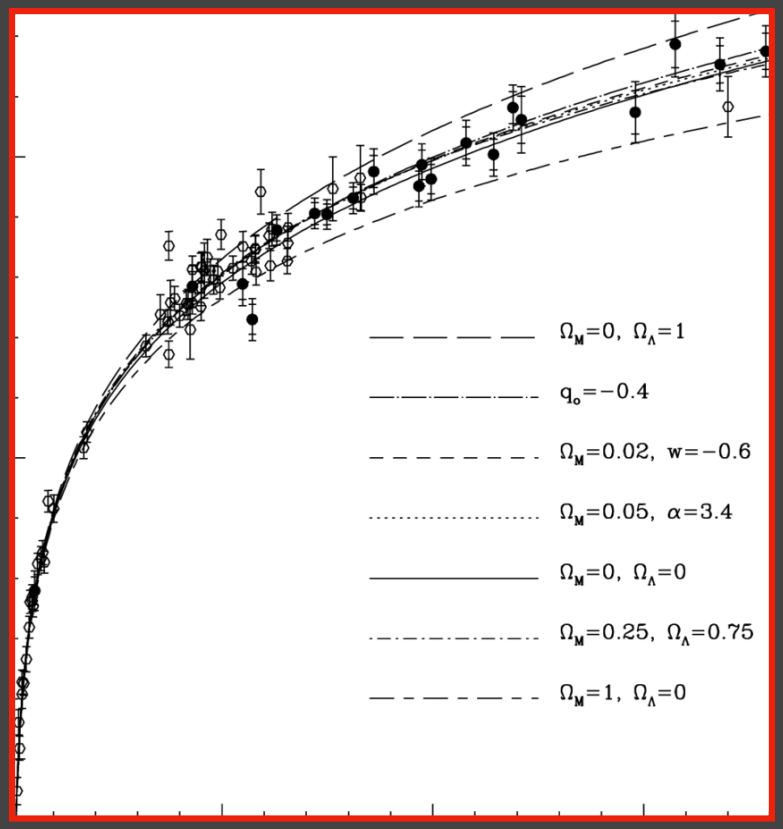
Regularization: weighting of regression coefficients (under regularization of λ) are continuous.

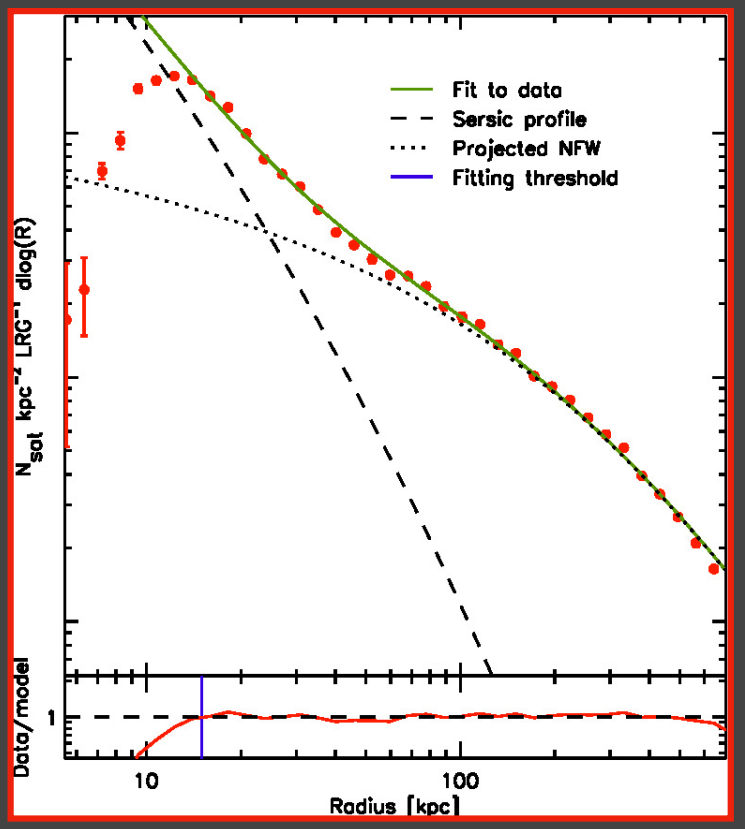
$$\ldots + 10^{-3}x^5 + 10^{-5}x^6 + 10^{-8}x^7 + \ldots 10^{-12}x^{19}$$

PCR: weighting of regression coefficients (with truncating θ of smaller eigenvalues) are 0 for the eliminated ones

$\ldots + 0.2z_5 + 0\,z_6 + 3\,z_7 + \ldots 0\,z_{19}$ (due to projection and truncation)

Non-linear Regression





When the regression model is motivated by specific theories and thus taking specific parameterization forms, one can always go for Bayesian (MCMC).

When one is purely going for predictive/descriptive power, one can adopt following standard non-linear regression techniques:

1. Local Regression via kernel
2. Gaussian Process Regression

Local Regression via kernel

A set of kernel functions, each $K(x_i, x)$ is local to each existing data point x_i.
The following way writes out the regression model "straightaway" for value $y = f(x \mid K)$on data point x — with predictive power!

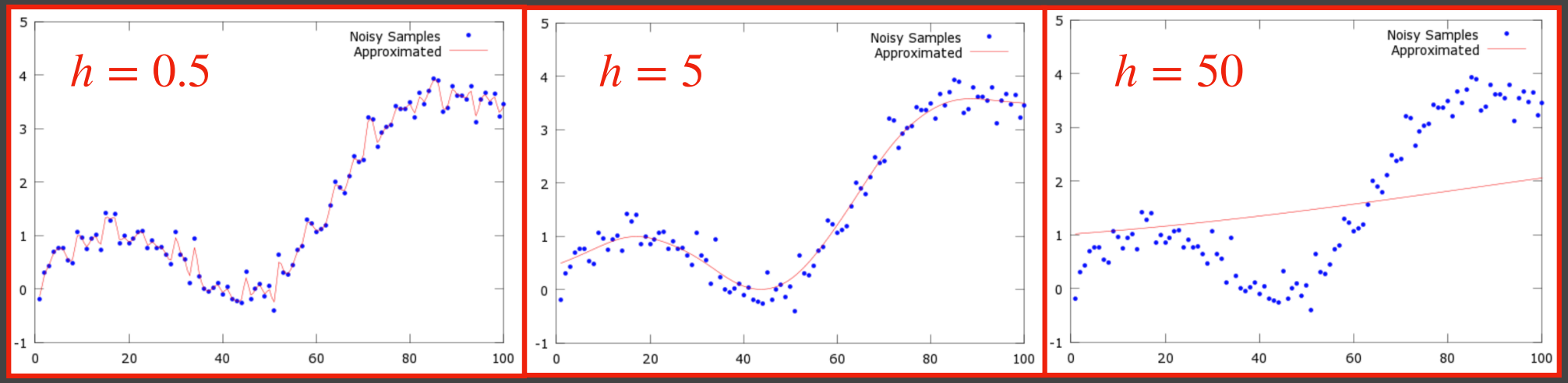
$$f(x \mid K) = \frac{\Sigma_{i=1}^{N} K\left(\frac{|x - x_i|}{h}\right) y_i}{\Sigma_{i=1}^{N} K\left(\frac{|x - x_i|}{h}\right)}$$

e.g., the kernel can take shape of Gaussian,

$$K\left(\frac{|x - x_i|}{h}\right) = \exp\left[\frac{-(x - x_i)^2}{2h^2}\right]$$

The kernel reaches maximal weight locally at x_i where measurement y_i is available, and asymptotes to zero as distance in x approaches infinity.

In this case, the bandwidth h is more important than exact kernel shape!
— determined through cross-validation.



In previous case, we focus on obtaining model parameter $\vec{\theta}$ for regression, under standard Bayesian framework, i.e., $p(\vec{\theta}\,|\,\{\vec{x}_i, y_i\}) \propto p(\{y_i\}\,|\,\{\vec{x}_i\}, \vec{\theta})\,p(\vec{\theta})$, where $p(y_i\,|\,\vec{x}_i, \vec{\theta}) = e(y_i\,|\,y = f(\vec{x}_i\,|\,\vec{\theta}))$ is essentially a combination of the regression model and the error function.

In some other cases, the regression problem really cares about making predictions given new observation input $\vec{x}$, i.e., instead of caring for $\vec{\theta}$ (which is constrained by old data $\{\vec{x}_i, y_i\}$), now the game aims at finding $y = f(\vec{x})$.

Note this concept is also widely applied in machine learning context.
This can be achieved using conditional probability relation:
$p(y\,|\,\{\vec{x}_i, y_i\}, \vec{x}) = \int p(y\,|\,\vec{x}, \vec{\theta}, \{\vec{x}_i, y_i\})\,p(\vec{\theta}\,|\,\{\vec{x}_i, y_i\})\,\mathrm{d}\vec{\theta}$. This is the basic of Bayesian linear regression, among which a very commonly used powerful regression is:

Gaussian Process Regression

$$p(y_j\,|\,\{\vec{x}_i, y_i\}, \vec{x}_j) = \mathcal{N}(\mu, \Sigma)$$

This massively simplifies the expansive evaluation of $p(y_j\,|\,\{\vec{x}_i, y_i\}, \vec{x}_j)$, but this serves a good estimate with solid foundation.

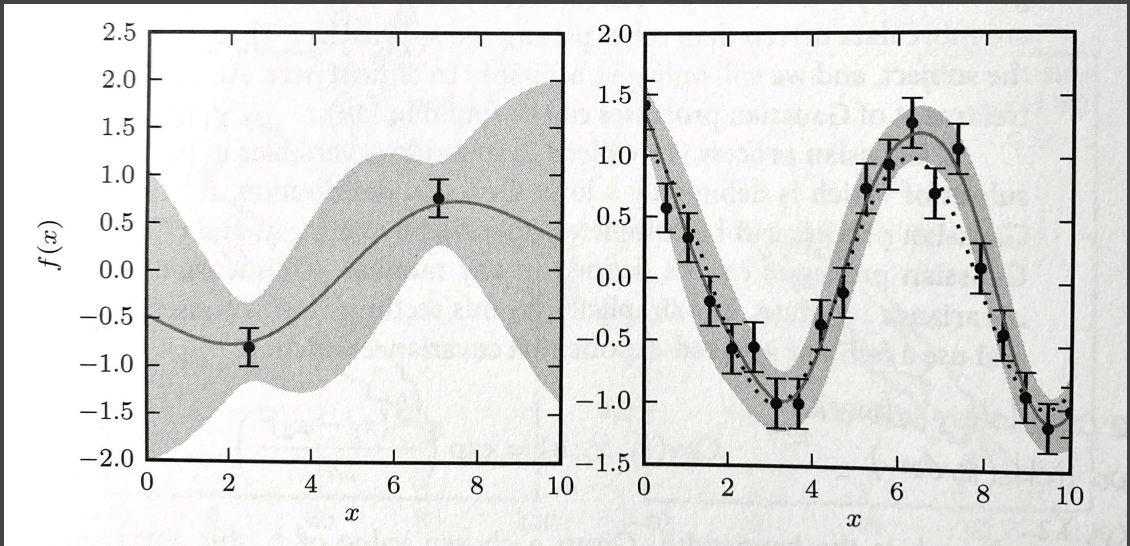
Gaussian Process Regression

Recommend online lecture from Cornell University:
https://www.youtube.com/watch?v=R-NUdqxKjos

$$p(y_j \mid \{\vec{x}_i, y_i\}, \vec{x}_j) = \mathcal{N}(\mu, \Sigma)$$

Training set

The goal is to find the μ and Σ for this Gaussian probability distribution.



Solution is:

- Mean is given by $\mu = K_{1j}^T K_{11}^{-1} \mathbf{y}_1$

  (check: this is exactly is the kernel regression solution, see two pages ago!)
- Variance is given by: $\Sigma = K_{jj} - K_{1j}^T K_{11}^{-1} K_{1j}$

The basic idea is that the relation between any two points is given by a positive covariance function:

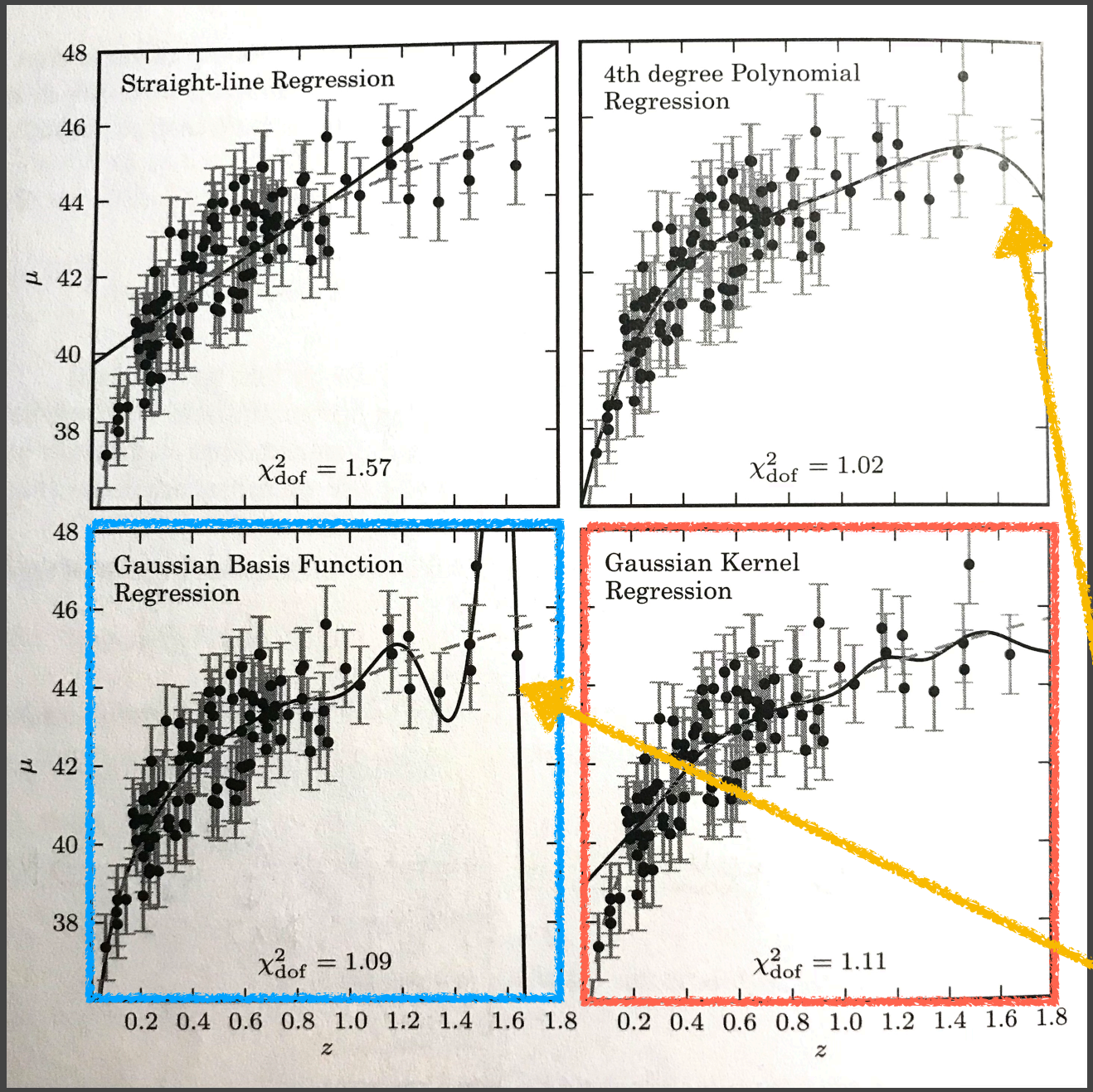
$$\mathrm{Cov}(x_1, x_2; h) = \exp\left(\frac{-(x_1 - x_2)^2}{2h^2}\right)$$

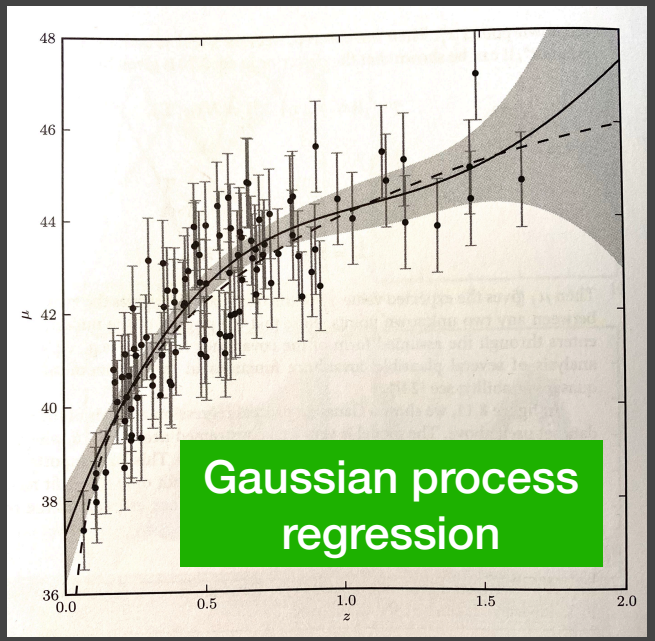
Points that are close together are more correlated than far-away.

Based on this, a covariance matrix can be constructed as: $K = \begin{pmatrix} K_{11} & K_{1j} \\ K_{1j}^T & K_{jj} \end{pmatrix}$,

each part K_{mn} is the regression kernel as defined two pages before, taking the form of the covariance function above,

1 referring to the training data subset.





Gaussian process regression does not provide a *best-fit*, but an uncertainty (based on covariance estimate) at each point.

Boundary behaviors are due to the particular format of basis function used!

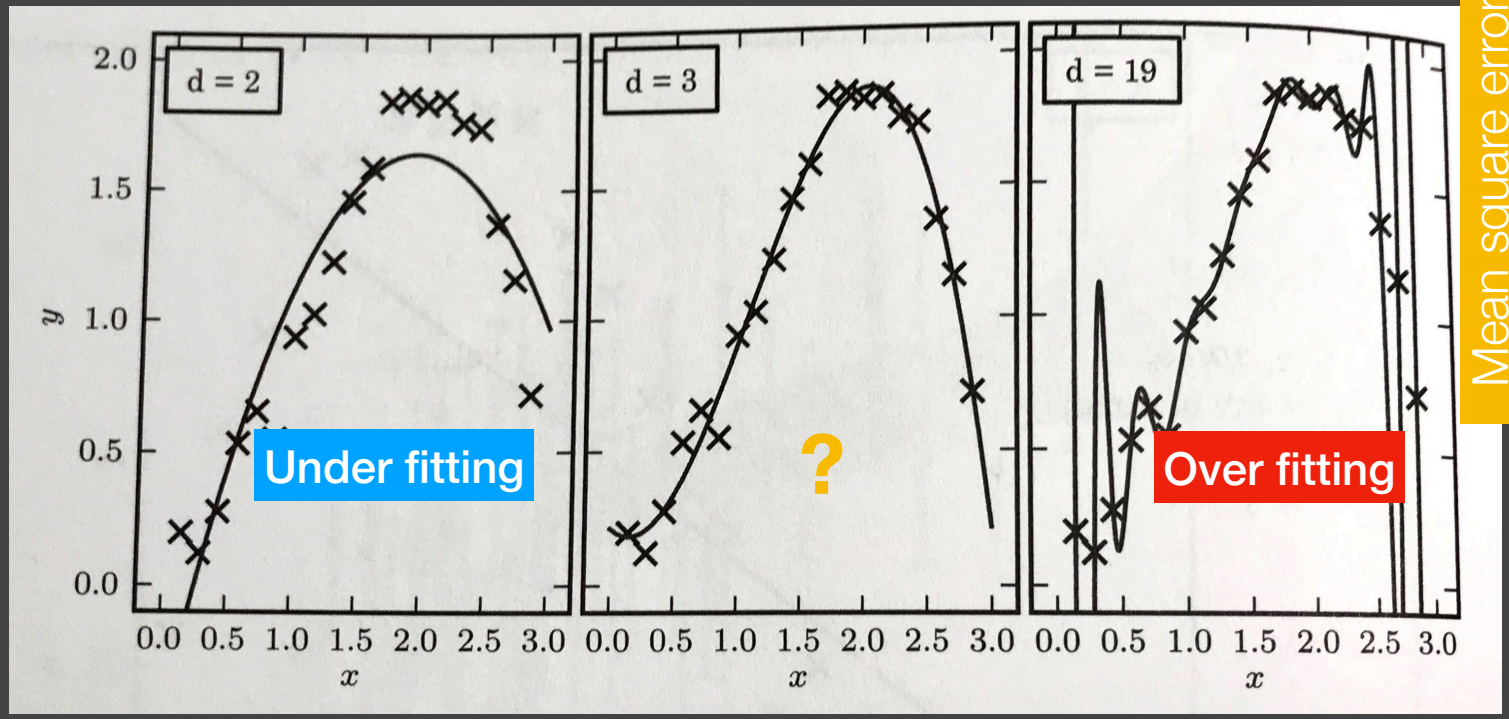
Gaussian basis functions have 15 Gaussians evenly spaced between z=0 and 2, with widths of 0.14.

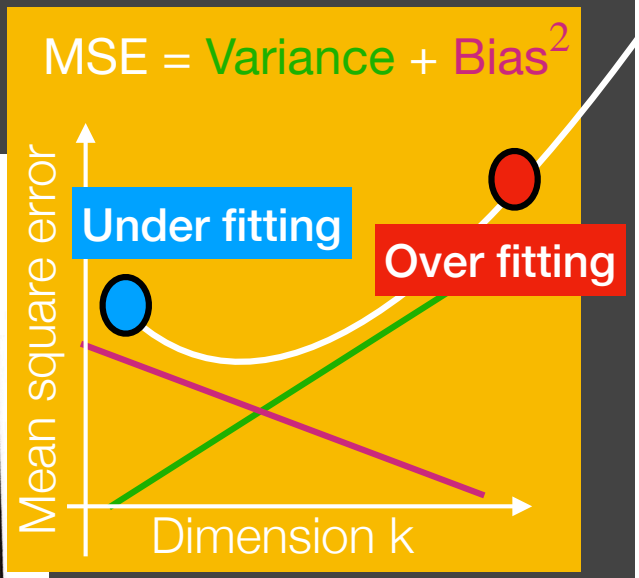
Kernel regression uses a Gaussian kernel with width 0.1.

Cross-Validation

A prelude…

Polynomial order d





Large model bias, small model variance

Low model bias, large model variance

CV helps to determine (1) whether a potential model is a good fit to the data, in this case the techniques are complementary to the model selection techniques such as AIC/BIC, or through the odds ratio); (2) optimal parameters of smoothing or regularization, size of training dataset etc, given a fixed family of model.

Cross-Validation

CV set general "rule of thumb":

1. *Training set*: f=50-70% of original dataset
2. Cross-validation: 0.5 x (1- f)
3. Test set: 0.5 x (1- f)

Training set:

Used to determine the best parameters of a given model applied to this training dataset. ***Here different models refer to with certain polynomial order, certain Huber C or regularization λ etc***. For each model, best parameters are obtained by ***minimizing training error*** ϵ_{tr}:

$$\epsilon_{\text{tr}} = \frac{1}{N_{\text{tr}}} \Sigma_{i=1}^{N_{\text{tr}}} (y_i - f(x_i \mid \vec{\theta}))^2$$

Importantly:

for each model, there is an ϵ_{tr} (minimizing this to obtain best parameter). For each model, there is also an ϵ_{cv}; the smallest one among different models indicates the winning model!

Cross-validation set:

Used to determine the best model. This is acheived by choosing the model with ***minimal cross-validation error*** ϵ_{cv}, which will be large in case of overfitting. Note $\vec{\theta}$ is the best fit parameter already sought for the training set given a model! (x_i, y_i) are cross-validation data set.

$$\epsilon_{\text{cv}} = \frac{1}{N_{\text{cv}}} \Sigma_{i=1}^{N_{\text{cv}}} (y_i - f(x_i \mid \vec{\theta}))^2$$

Test set:

Using ***test-set error*** ϵ_{ts} to estimate reliability of model (error) for new unlabelled data. (x_i, y_i) are of the test data set.

$$\epsilon_{\text{ts}} = \frac{1}{N_{\text{ts}}} \Sigma_{i=1}^{N_{\text{ts}}} (y_i - f(x_i \mid \vec{\theta}))^2$$

Cross-Validation

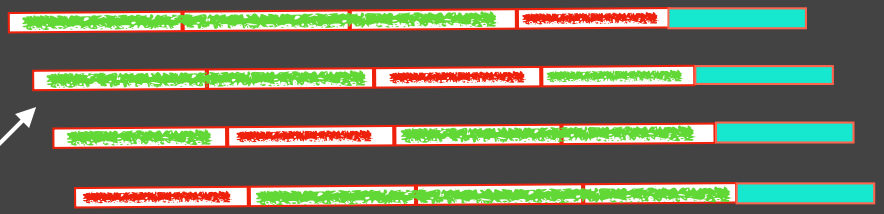
Other techniques (in particular for smaller dataset)

Two-fold cross-validation:

1. Split up data into d_0, d_1 and d_2
2. Train model on set d_0, cross-validation on set d_1, obtain ϵ_{tr}, ϵ_{cv};
3. Train model on set d_1, cross-validation on set d_0, obtain ϵ_{tr}, ϵ_{cv};
4. For each model, the cross-validation error is the mean (or median) of ϵ_{cv} from step 2 and 3;
5. Compare among different models, chose the winning one;
6. Test set d_2 for evaluating final (best) model liability.

K-fold cross-validation:

1. Split the total dataset into k +1 (for d_2) trunks;
2. Train model for *k* (e.g., 4) time, each time on the combined $(k-1)$ data trunks, and cross-validate using the remaining 1 trunk;
3. For each model, taking the mean or median of *k* results of ϵ_{cv} from step 2 as the cross-validation error;
4. Compare among regression models;
5. Test set d_2 for evaluating final model liability.



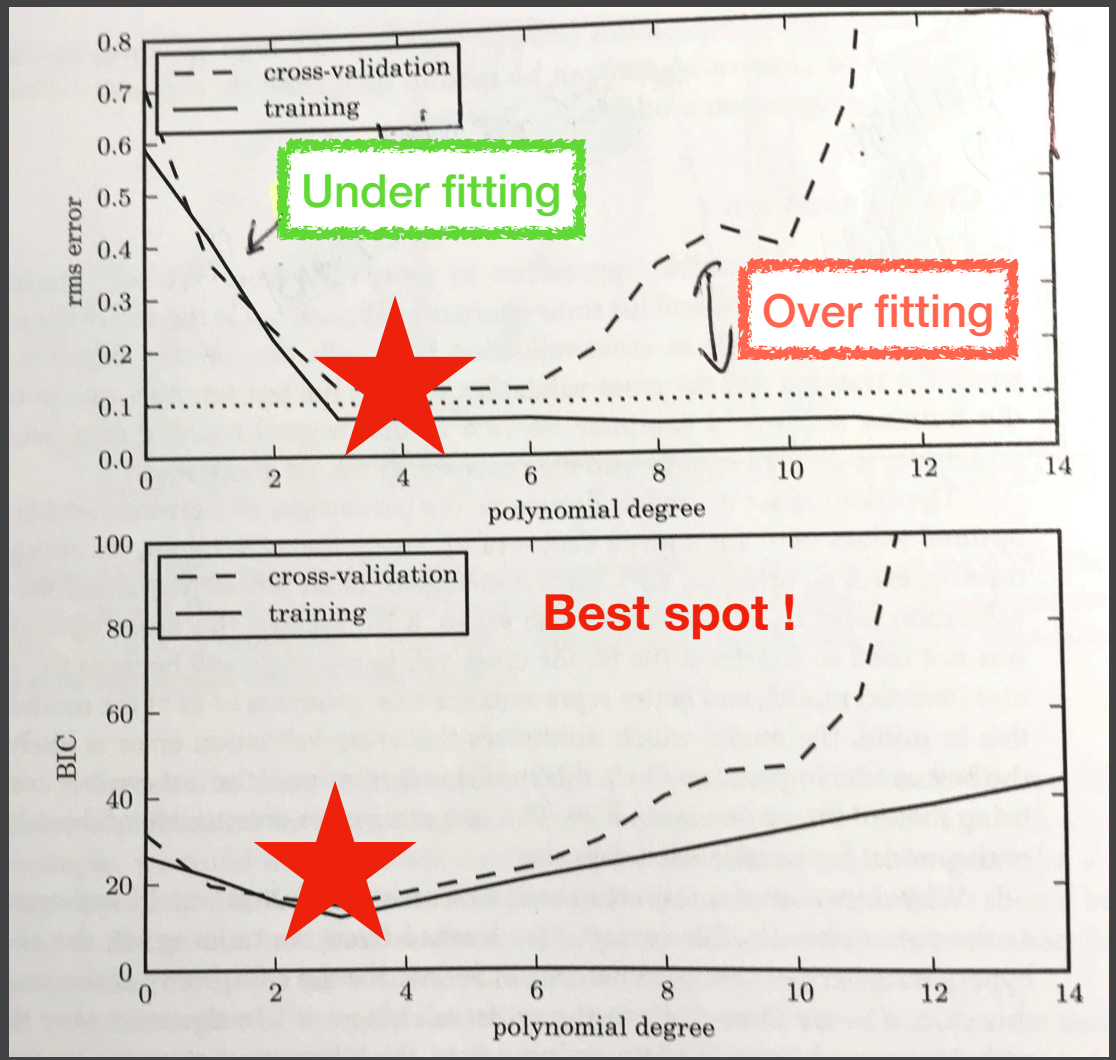
Leave-one-out cross-validation

(extreme of K-fold, very small dataset):

Each trunk contains only 1 data point; the rest are used as training set for each training and validation.

Cross-Validation

Given a dataset, the question we ask is how to determine an optimal regression model (with good balance between bias and variance), i.e., a model that is neither over-fitting nor under-fitting the given dataset.



The models would under-fit the data initially, but the rms error will decrease as model and data in better compatibility. However, the model will then over-fit the training dataset. While training error getting smaller, CV error starts increasing indicating model "over-fitting noise in the training dataset" .

The best regression model appears at CV error ϵ_{cv} reaching minimum.

In the case of under-fitting:
(high bias, low variance)

- Increase model complexity
- Decrease regularization λ

In the case of overfitting:
(low bias, large variance)

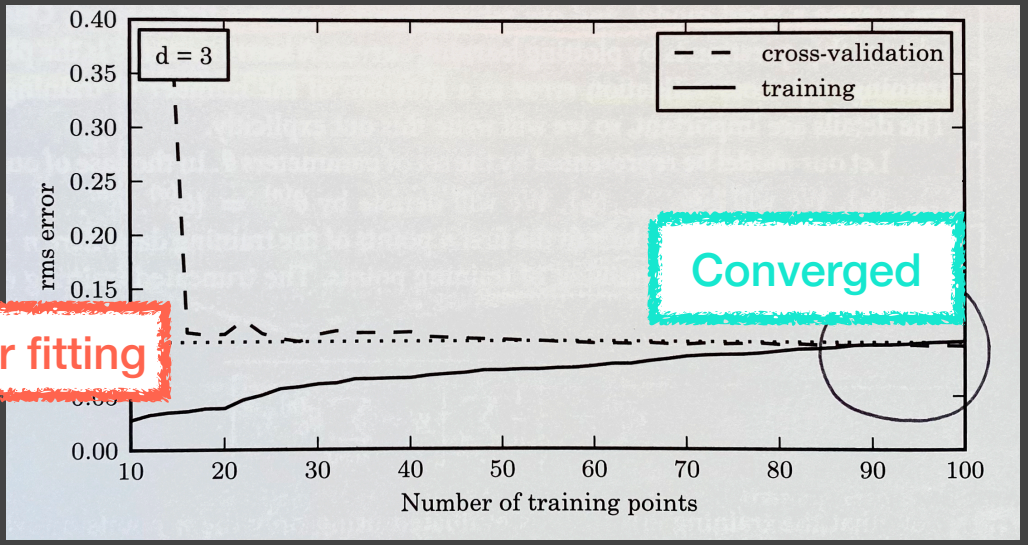
- Decrease model complexity
- Increase regularization λ

Cross-validation

Cross-Validation:

Given a fitting model (physical), the question that we ask is how to determine the best size of training dataset or the best number of training epochs, in order to better constrain a theoretical model, and to understand the compatibility between model and data.

We can use a learning curve



★ When training set is small, model over-fits data, the regression performance (on the training set) is mainly dominated by variance, i.e., models that fit to different realizations of data differ a lot.

In particular here: $\epsilon_{\rm cv} > \epsilon_{\rm tr}$

In the case of overfitting:
(low bias, large variance)

- Decrease model complexity
- Increase regularization λ
- Need more training data

★ When training set is sufficiently large and model is no longer suffering over-fitting. With increase of sample size, $\epsilon_{\rm tr}$ and $\epsilon_{\rm cv}$ will gradually converge. This indicates the training and CV datasets getting statistically equal. The regression performance in this case is mainly dominated by model bias.

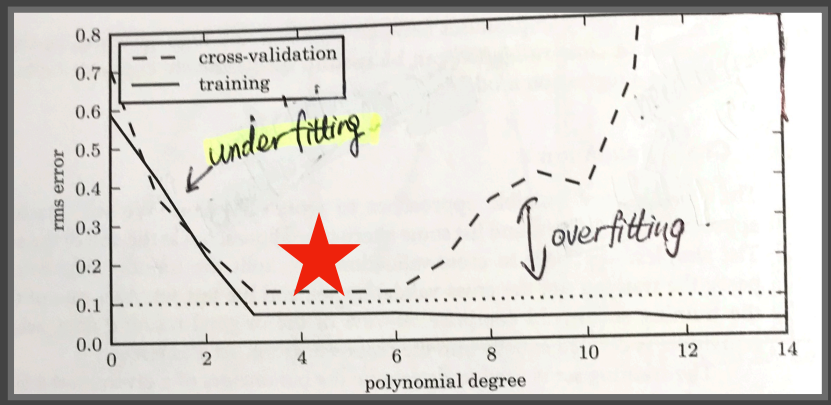
When CV error converges to Training error (model bias dominates), in order to further reduce model bias:

- Increase model complexity
- Decrease regularization
- Add additional feature to data

Cross-Validation

Application examples:

In this case, the model complexity is encoded by the polynomial order, but can be controlled by regularization λ. Different λ essentially correspond to models with different complexities. How do we choose the regularization parameter λ?



Example: deciding polynomial degree

Regularization

Log posterior distribution: $\ln L_p = \ln L + \lambda |\theta|^2$

Regularization minimize: $(Y - M\theta)^T C^{-1} (Y - M\theta) + \lambda |\theta^T \theta|$

Solution: $\theta = (M^T C^{-1} M + \lambda I)^{-1} (M^T C^{-1} Y)$

Applying *k*-fold cross-validation and define cross-validation error:

$$\epsilon_{\mathrm{cv}}(\lambda) = \frac{1}{k} \Sigma_k \frac{1}{N^{\{k\}}} \Sigma_i^{N^{\{k\}}} \frac{[y_i - f(x_i | \vec{\theta}(\lambda)]^2}{\sigma_i^2}$$

where $N^{\{k\}}$ is the number of data point in each $1/k$ cross-validation trunk.

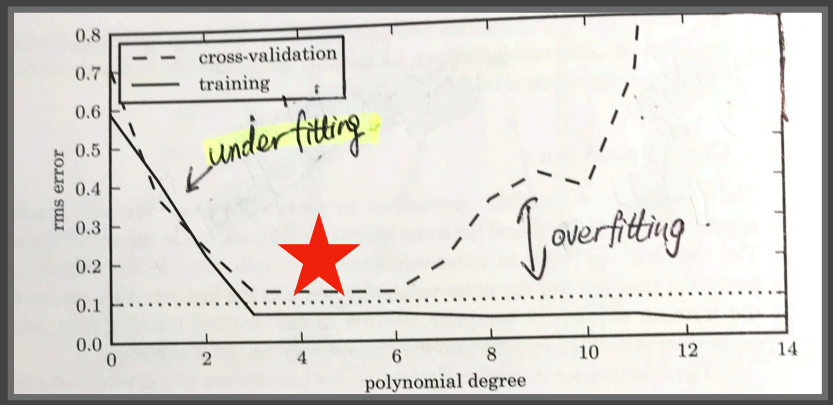
Finding the best λ that minimizes the cross-validation error ϵ_{cv}!

Cross-Validation

Application examples:

Minimizing CV error to find best regularization

$$\epsilon_{\text{cv}}(\lambda) = \frac{1}{k}\Sigma_k \frac{1}{N_k}\Sigma_i^{N_k} \frac{[y_i - f(x_i \mid \vec{\theta}(\lambda)]^2}{\sigma_i^2}$$



Example: deciding polynomial degree

Local regression via kernel

$$f(x \mid K) = \frac{\Sigma_{i=1}^N K\left(\frac{|x - x_i|}{h}\right) y_i}{\Sigma_{i=1}^N K\left(\frac{|x - x_i|}{h}\right)}$$

e.g., Gaussian kernel

$$K\left(\frac{|x - x_i|}{h}\right) = \exp\left[\frac{(x - x_i)^2}{2h^2}\right]$$

How do we choose the bandwidth h?

e.g., applying *leave-one-out* cross-validation for regression of N data points, and define a cross-validation error as the mean of N (subset) CV errors:

$$\epsilon_{\text{cv}}(h) = \frac{1}{N}\Sigma_j^N [y_j - f(x_j \mid K(h))]^2,$$

where $\{x_j, y_i\}$ is the left-one-out point for CV. Alternatively, median can be used.

Finding the best bandwidth h that minimizes the cross-validation error ϵ_{cv}!

Optimal bandwidth decreases with sample size at rate of $N^{-1/5}$.

Summary on Regression and model fitting

Maximize data likelihood:

Error behavior via error function!

$$p(\{(x_i, y_i)\} \mid \vec{\theta}, I) = \Pi_i^N e(y_i \mid y = f(x_i \mid \vec{\theta}))$$

Minimize Cost function $\Sigma_i^N \phi(t_i) = -\Sigma_i^N \ln e(y_i \mid y = f(x_i \mid \vec{\theta}))$:

Loss function: $\phi(t) = t^2/2 \quad \text{if } |t| \leqslant c$

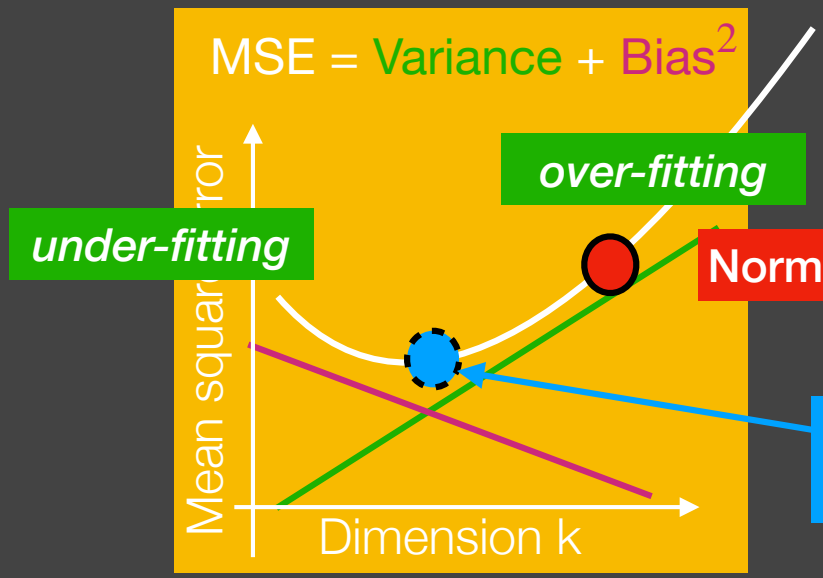
L_2 norm ~ Gaussian-like error

Loss function: $\phi(t_i) = c|t_i| - 1/2c^2 \quad \text{if } |t_i| \geqslant c$

L_1 norm ~ Exponential-like error

Where $t_i = [y_i - f(x_i \mid \vec{\theta})]/\sigma_i$.

Linear regression models with solutions under Gaussian error



Cross-Validation

Normal LSE without regularization

PCR or Regularization finding effective k!

A comment on "optimization" and "parameter estimation"

Gradient descent algorithms are "optimization" tool, with which we find best - most-likely parameters by finding the minimum of the surface of our designed cost function.

(Monte Carlo based) sampler are "parameter estimation" tools, with which we can evaluate parameter most-likely values, variance, covariance etc. Often used in Bayesian inference, where posterior is sampled, we can find MAP etc as parameter estimation.

Optimization and sampling is very different. Optimization will in general give the localized point at the distribution peak, whereas sampling aims to build the distribution.

Parameter estimation with sampler tools:

In most astronomy cases we build a model and finding parameters and their uncertainties mainly through a sampler, e.g., JAM or lens modeling. In these cases, model parameters are the physical quantities that we care about and therefore parameter estimation is the essential game to play. In particular, through Bayesian, we can get unbiased result than assuming MLE in cases where CLT is not applied.

For pure and proper parameter estimation, we shall go for finding most-likely values and parameter uncertainties (at least). Probability sampling based estimates (e.g,. MCMC) can be used not only to estimate the most likely value (e.g., MLE/MAP) but also uncertainty and correlation etc. Therefore, such methods are complete in serving the purpose.

Optimization tools in parameter estimation

Using optimization for parameter estimate purposes, indeed a single optimization in this case can be used for point estimate (like in MAP/MLE). Can we also find out about the uncertainties and correlations about model parameters during the optimization process? This in general is very difficult. It may require some dedicated software (such as GAMS) and may need additional information; the performance would not guaranteed. Some techniques combine the optimization and sampling at the same time to achieve the evaluation on uncertainties. Stein Variational Gradient Descent (SVGD) is one method that kind of blurs sampling/optimization lines. In that case one optimize a particle distribution to try and get it to match the target distribution.

Optimization tools in machine learning

In Machine learning, it is a very clear (and sensible) usage of optimization. The science goal in ML application is not model parameters (i.e., neuron's weights), but model predictions. Using the model, we can make predictions (either regression or classification) on the output given a set of input. Comparing to the previous case, the physical property that we care about in the first case (as model parameters), now become outputs as a prediction given a set of input. In this case, we do NOT need to explicitly evaluating the "uncertainty" of ML model (hyper-) parameters. Instead we want to train the model according to some cost function to find best ML parameters through optimization only aiming at quickly converge to making further predictions on new data.