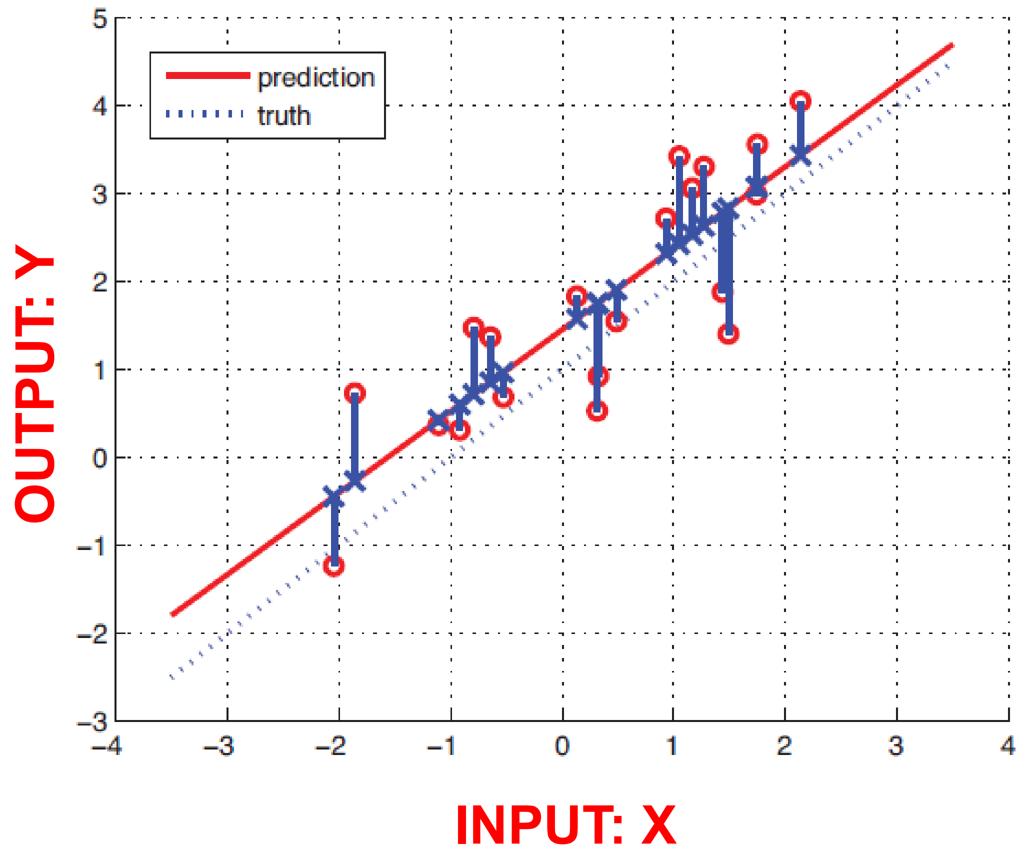
Computer
Science

CSC696H: Advanced Topics in Probabilistic Graphical Models

Gaussian Process Regression

Prof. Jason Pacheco

Linear Regression



Regression Learn a function that predicts outputs from inputs,

$$y = f(x)$$

Outputs y are real-valued

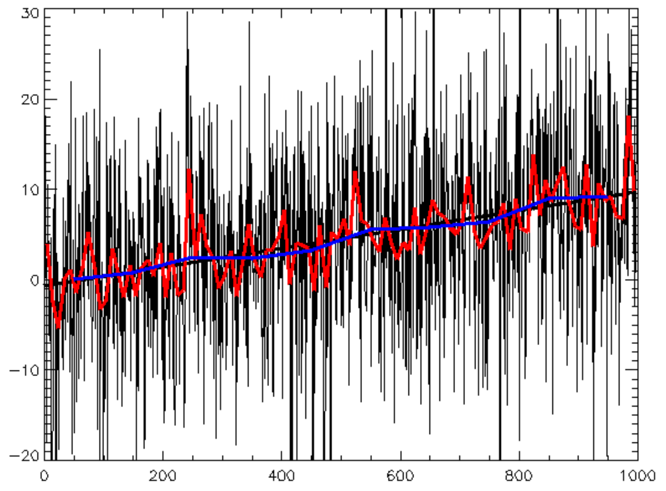
Linear Regression As the name suggests, uses a *linear function*:

$$y = w^T x + b$$

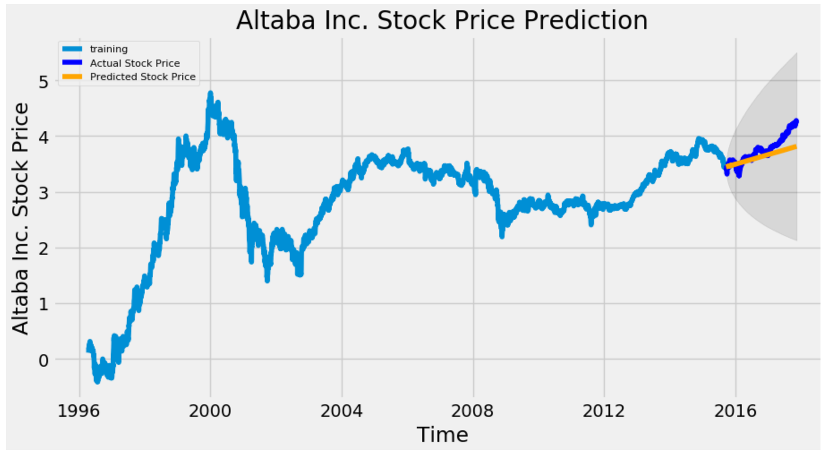
We will add noise later…

Linear Regression

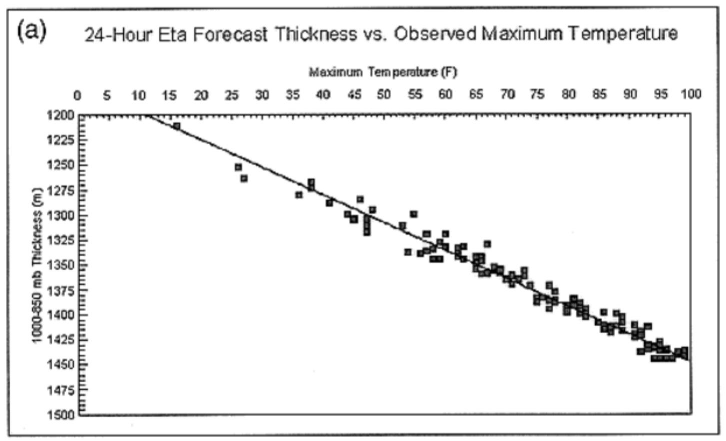
Where is linear regression useful?



Trendlines



Stock Prediction



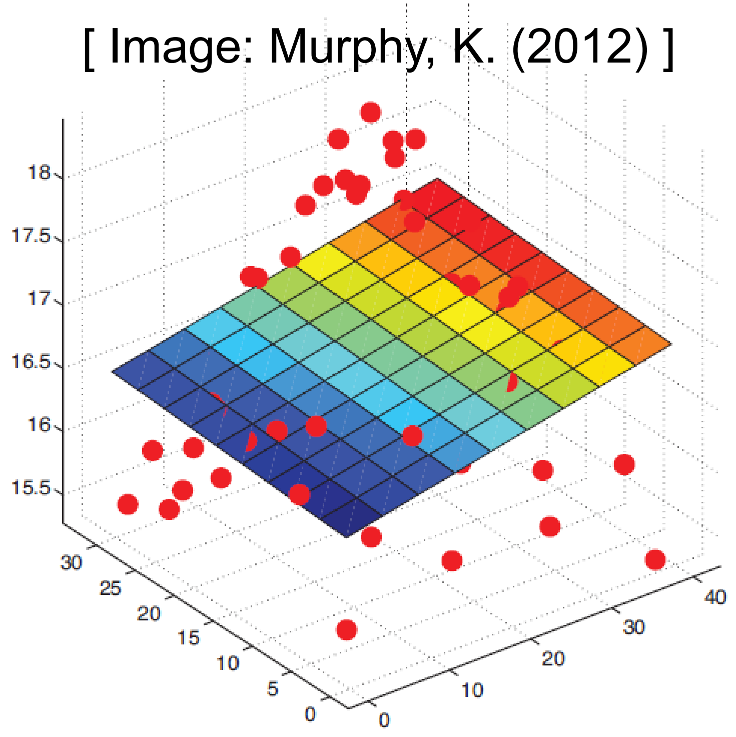
Climate Models

Massie and Rose (1997)

Used anywhere a linear relationship is assumed between continuous inputs / outputs

Linear Regression

[Image: Murphy, K. (2012)]



For D-dimensional input vector $x \in \mathbb{R}^D$ the plane equation,

$$y = w^T x + b$$

Often we simplify this by including the intercept into the weight vector,

$$\widetilde{w} = \begin{pmatrix} w_1 \\ \vdots \\ w_D \\ b \end{pmatrix} \qquad \widetilde{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_D \\ 1 \end{pmatrix} \qquad y = \widetilde{w}^T \widetilde{x}$$

Since:

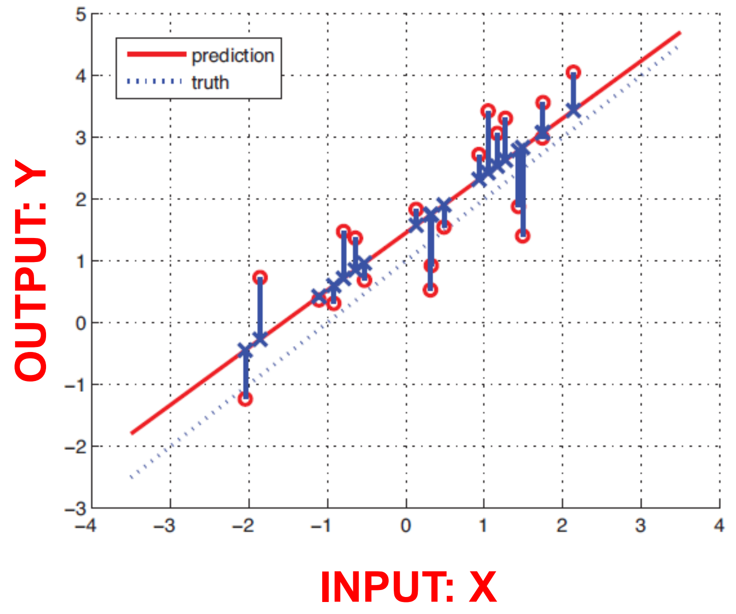
$$\widetilde{w}^T \widetilde{x} = \sum_{d=1}^{D} w_d x_d + b \cdot 1$$
$$= w^T x + b$$

Linear Regression

Input-output mapping is not exact, so we will add zero-mean Gaussian noise,

Multivariate Normal (uncorrelated)

$$y = w^T x + \epsilon \quad \text{where} \quad \epsilon \sim \mathcal{N}(0, \sigma^2 I)$$



This is equivalent to the likelihood function,

$$p(y \mid w, x) = \mathcal{N}(y \mid w^T x, \sigma^2 I)$$

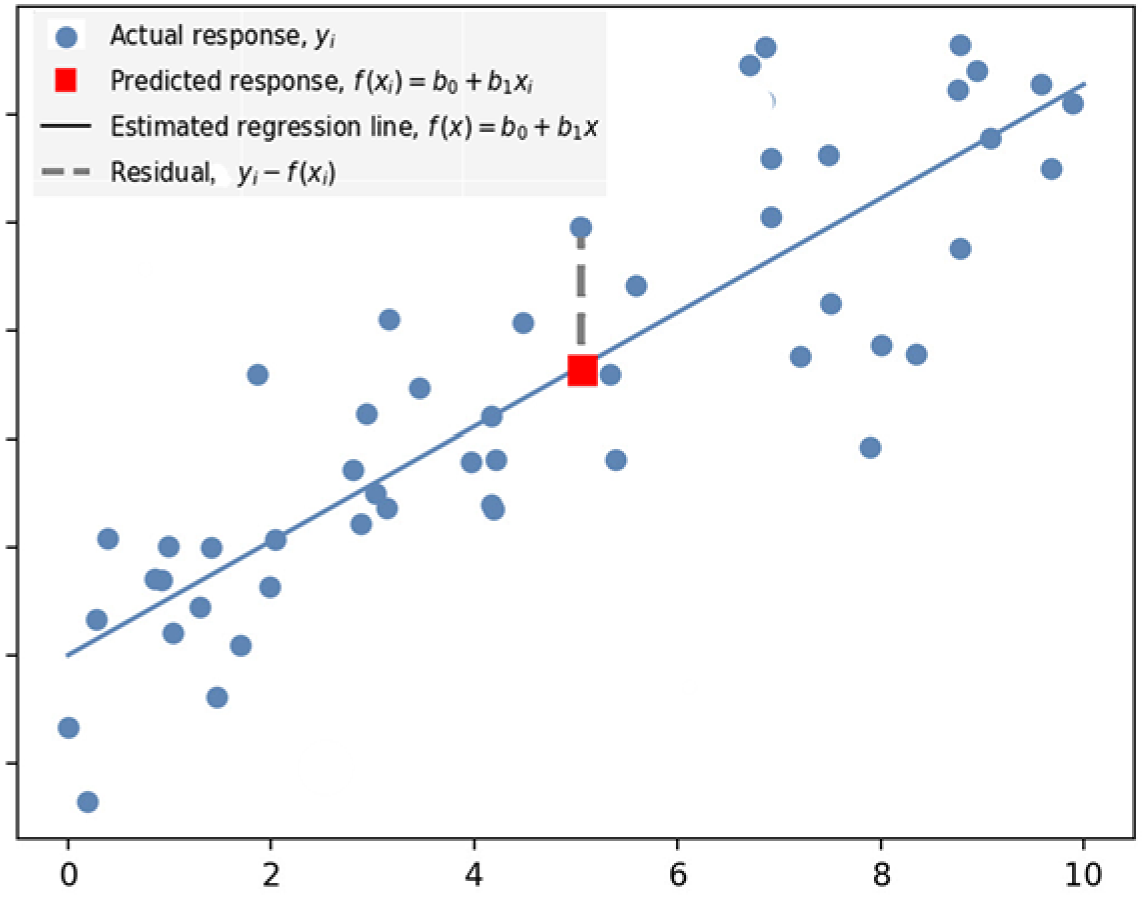
Because Adding a constant to a Normal RV is still a Normal RV,

$$z \sim \mathcal{N}(m, P) \qquad\qquad z + c \sim \mathcal{N}(m + c, P)$$

In the case of linear regression $z \to \epsilon$ and $c \to w^T x$

Least Squares Regression

Need to estimate regression weights…



The distance from each point to the line is the **residual**

$$y - w^T x$$

Find a line that minimizes the sum of squared residuals

$$w^* = \arg\min_w \sum_{i=1}^{N} (y_i - w^T x_i)^2$$

Over training all the data,

$$\{(x_i, y_i)\}_{i=1}^{N}$$

https://www.activestate.com/resources/quick-reads/how-to-run-linear-regressions-in-python-scikit-learn/

Maximum Likelihood Estimate = Least Squares

Given training data $\{(x_i, y_i)\}_{i=1}^N$ likelihood function is,

$$\log \prod_{i=1}^N p(y_i \mid x_i, w) = \sum_{i=1}^N \log p(y_i \mid x_i, w)$$

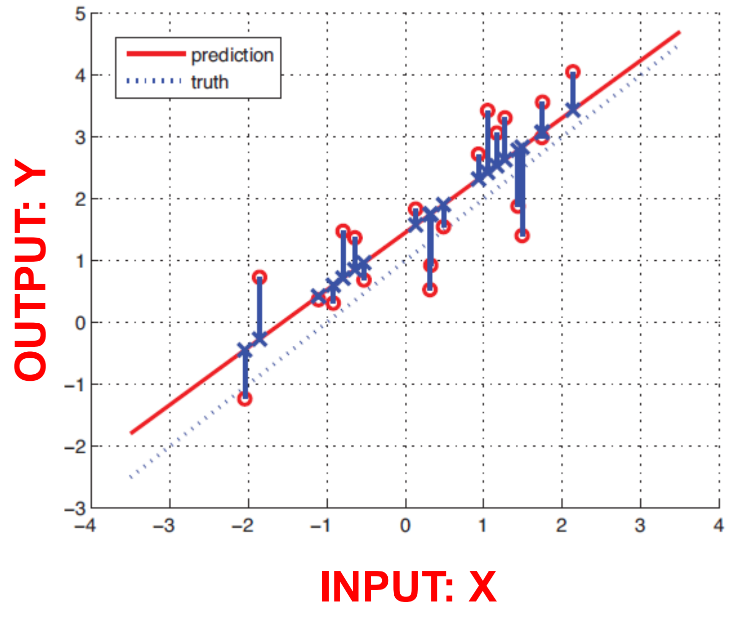
Recall that the likelihood is Gaussian:

$$p(y \mid w, x) = \mathcal{N}(y \mid w^T x, \sigma^2 I)$$

MLE maximizes the log-likelihood over data,

$$w^{\text{MLE}} = \arg\max_w \sum_{i=1}^N \log \mathcal{N}(y_i \mid w^T x_i, \sigma^2 I)$$

$$= \arg\min_w \sum_{i=1}^N (y_i - w^T x_i)^2$$



Maximume Likelihood Estimation of regression weights w is least squares solution

Least Squares in Higher Dimensions

Things are a bit more complicated in higher dimensions and involve more linear algebra,

Incorporate bias term into weights

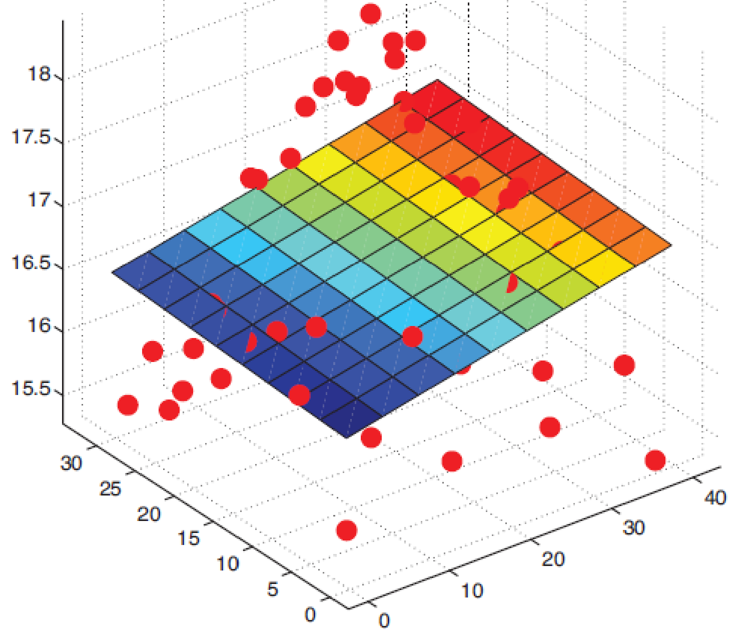
$$\mathbf{X} = \begin{pmatrix} 1 & x_{11} & \dots & x_{1D} \\ 1 & x_{21} & \dots & x_{2D} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_{N1} & \dots & x_{ND} \end{pmatrix}$$

Design Matrix
(each training input on a column)

$$\mathbf{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix}$$

Vector of Training labels

[Image: Murphy, K. (2012)]



Can write regression over *all training data* more compactly…

$$\mathbf{y} = \mathbf{X}w$$

Least Squares in Higher Dimensions

Least squares can also be written more compactly,

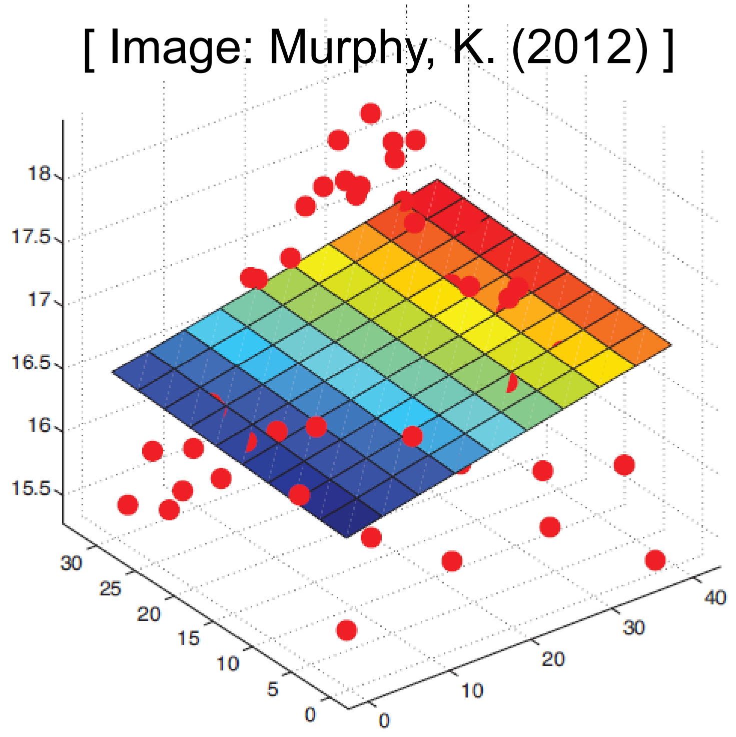
$$\min_w \sum_{i=1}^{N} (y_i - w^T x_i)^2 = \|\mathbf{y} - w^T \mathbf{X}\|^2$$

Taking vector gradients, setting to zero, and solving tive the solution:

$$w^{\text{OLS}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Ordinary Least Squares (OLS) solution

[Image: Murphy, K. (2012)]



Derivation a bit advanced for this class, but…

- We know it has a closed-form and why
- We can evaluate it
- Generally know where it comes from

Linear Regression Summary

1. Definition of linear regression model,

$$y = w^T x + \epsilon \qquad \text{where} \qquad \epsilon \sim \mathcal{N}(0, \sigma^2 I)$$

2. For N iid training data fit using least squares,

$$w^{\text{OLS}} = \arg\min_w \sum_{i=1}^{N} (y_i - w^T x_i)^2$$

3. Equivalent to maximum likelihood *estimate* with closed form :

$$\mathbf{X} = \begin{pmatrix} 1 & x_{11} & \dots & x_{1D} \\ 1 & x_{21} & \dots & x_{2D} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_{N1} & \dots & x_{ND} \end{pmatrix} \qquad \mathbf{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix} \qquad w^{\text{OLS}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

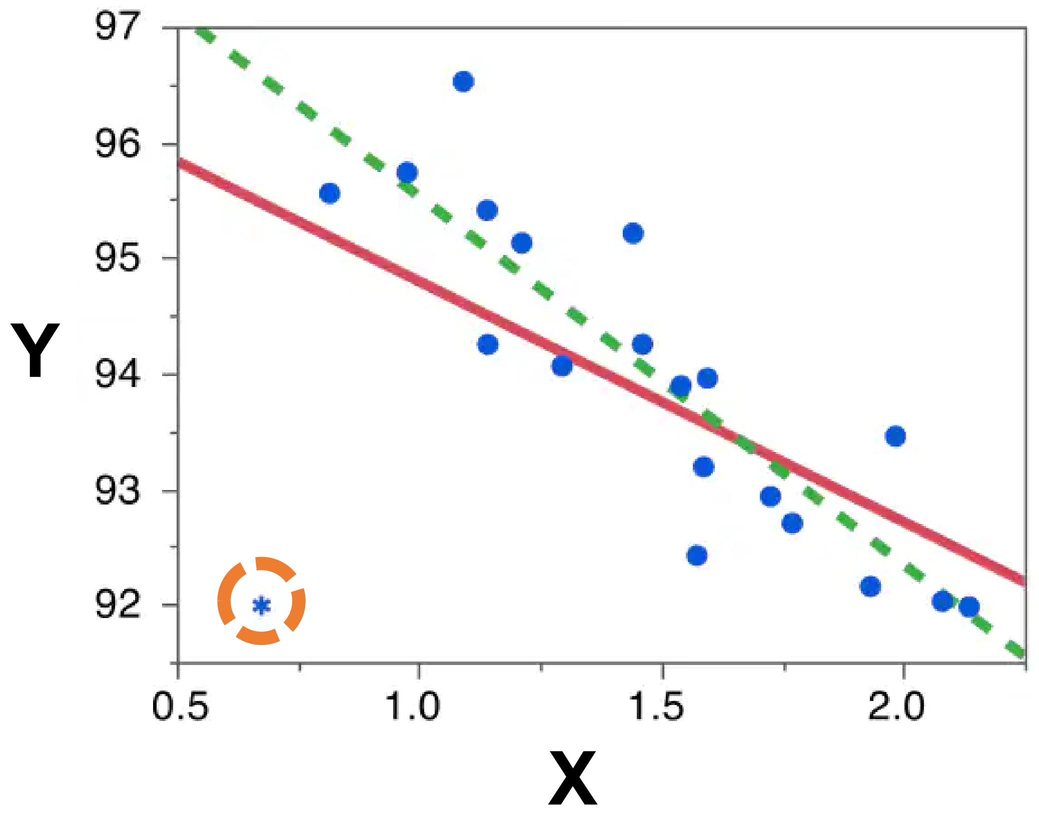
Design Matrix
(each training input on a column)

Vector of
Training labels

QUESTIONS?

Outliers in Linear Regression

Ordinary least squares regression is sensitive to outliers…



Quadratic regularizer reduces sensitivity:

$$w^{\mathrm{L2}} = \arg\min_w \sum_{i=1}^{N} \underbrace{(y_i - w^T x_i)^2}_{\text{Quadratic}} + \underbrace{\frac{\lambda}{2}\|w\|^2}_{\text{Quadratic}}$$

Quadratic objective / closed-form solution:

$$w^{\mathrm{L2}} = (\lambda I + \mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

(called “ridge regression” in statistics)

https://www.jmp.com/en_us/statistics-knowledge-portal/what-is-multiple-regression/mlr-residual-analysis-and-outliers.html

Bayesian Interpretation

Weights are RVs with Gaussian prior and joint probability:

$$p(w) = \mathcal{N}(0, \lambda^{-1} I)$$

$$p(y \mid w, x) = \mathcal{N}(w^T x, 1)$$

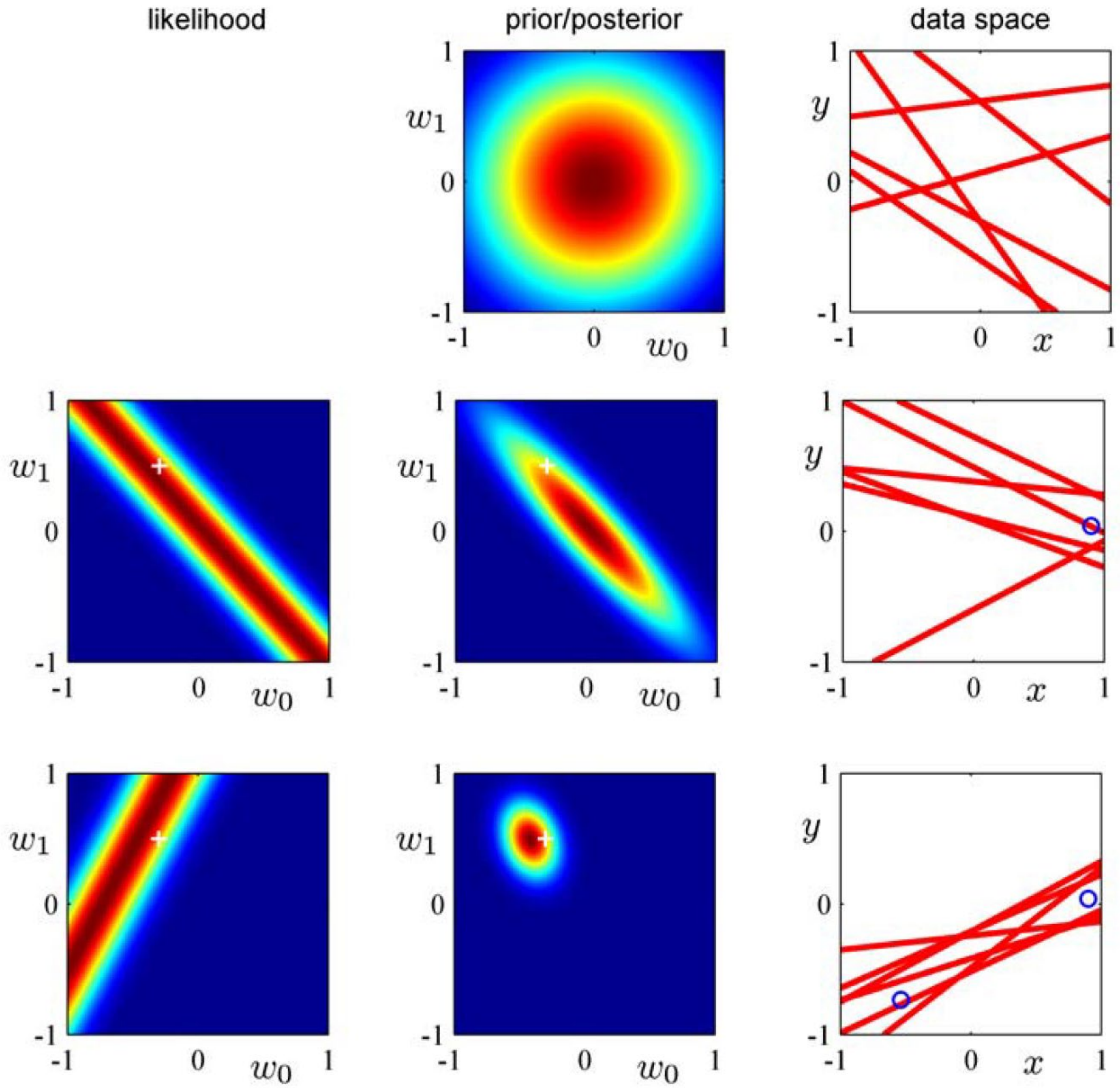
Given training data $\{(x_i, y_i)\}_{i=1}^N$ the posterior is given by Bayes' rule:

$$p(w \mid x_1^N, y_1^N) \propto \mathcal{N}(w \mid 0, \lambda^{-1} I) \prod_{i=1}^N \mathcal{N}(y_i \mid w^T x_i, 1)$$

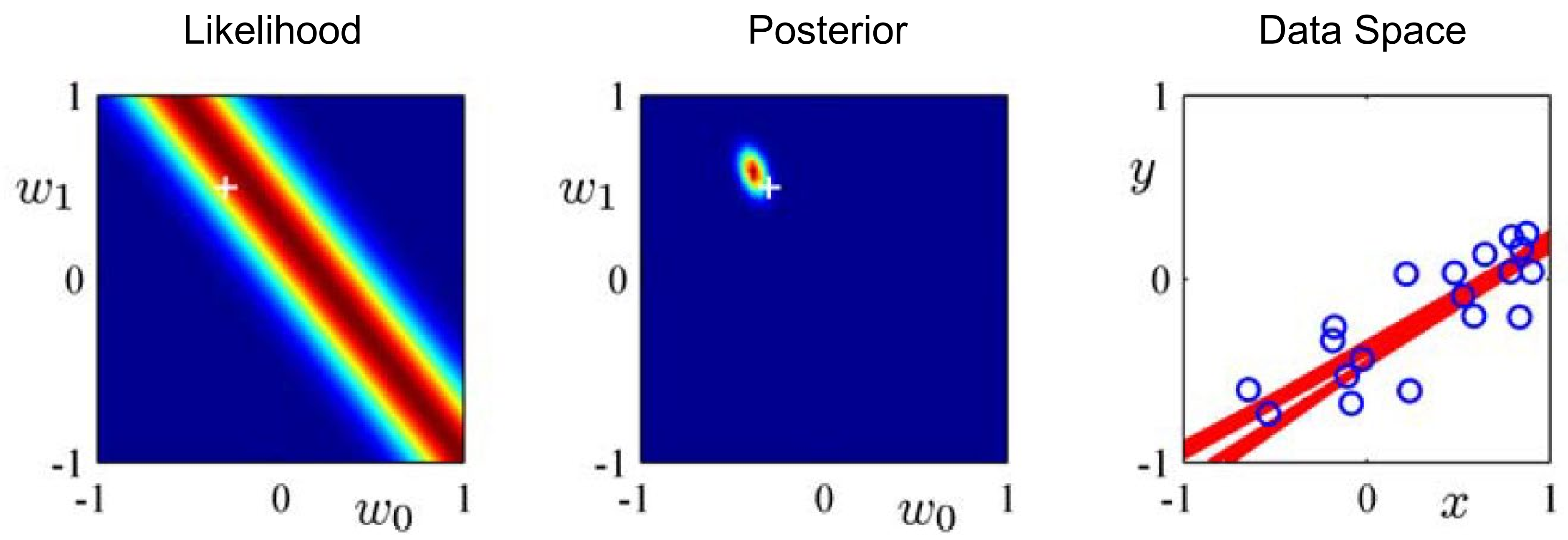
Taking the natural log and dropping constants we have:

$$w^{\text{MAP}} = \arg\max_w \log p(w \mid x_1^N, y_1^N) = \arg\min_w \sum_{i=1}^N (y_i - w^T x_i)^2 + \frac{\lambda}{2} \|w\|^2$$

L2 Regularized Least Squares = Bayesian MAP Estimate w/ Gaussian Prior



Source: Chris Bishop, PRML

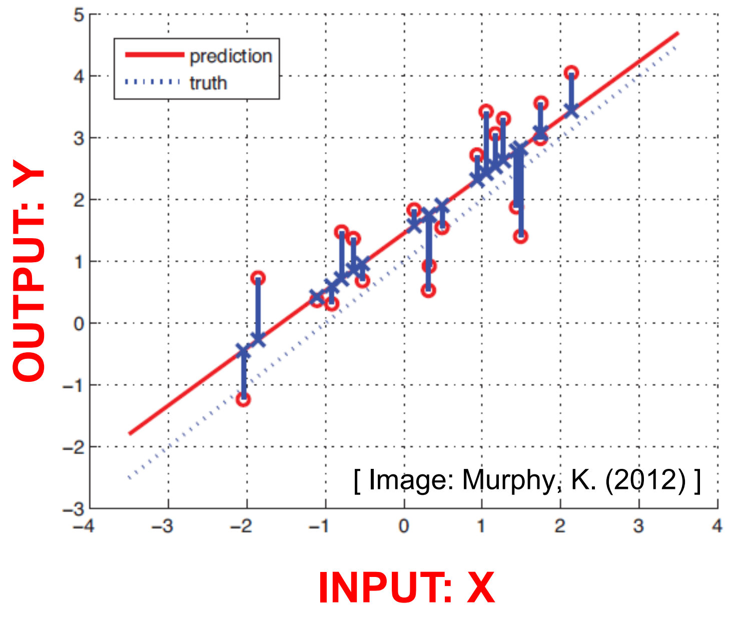


Posterior concentrates on true weights as more data observed

Likelihood outweighs prior in the limit (converges to MLE)

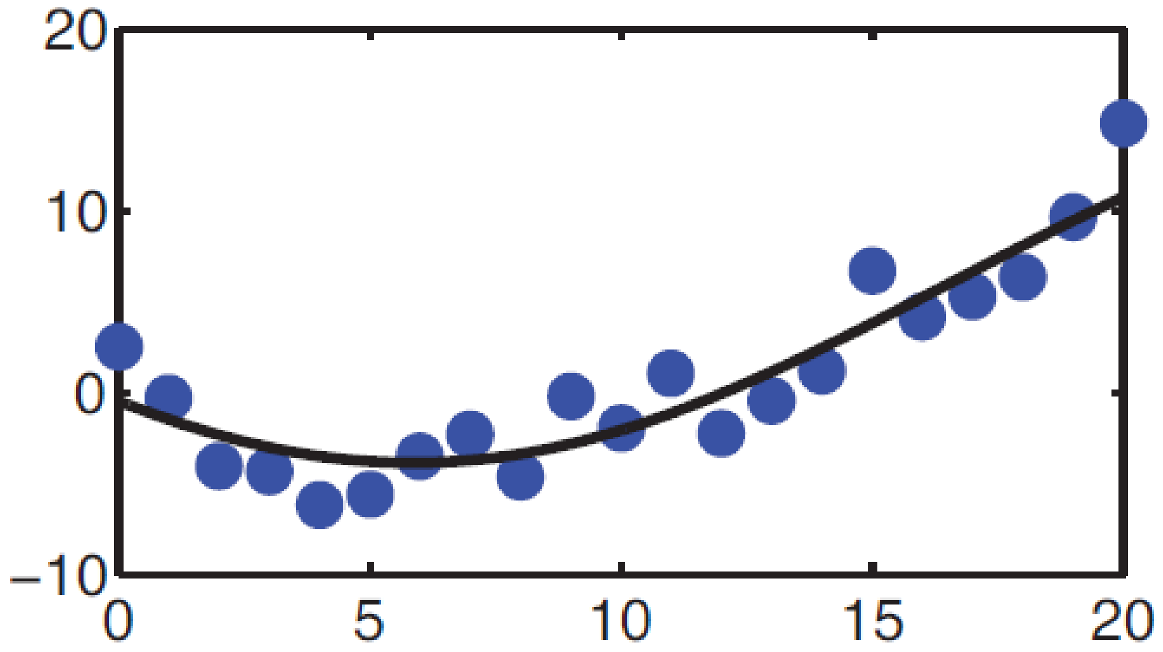
Source: Chris Bishop, PRML

Linear vs. Nonlinear Models



Linear Regression Fit a *linear function* to the data,

$$y = w^T x + b$$



What if our data are *not* well-described by a linear function?

Example: Earthquake Prediction

Suppose that we want to predict the number of earthquakes that occur of a certain magnitude. Our data are given by,

FIGURE 5-3A: WORLDWIDE EARTHQUAKE FREQUENCIES, JANUARY 1964–MARCH 2012

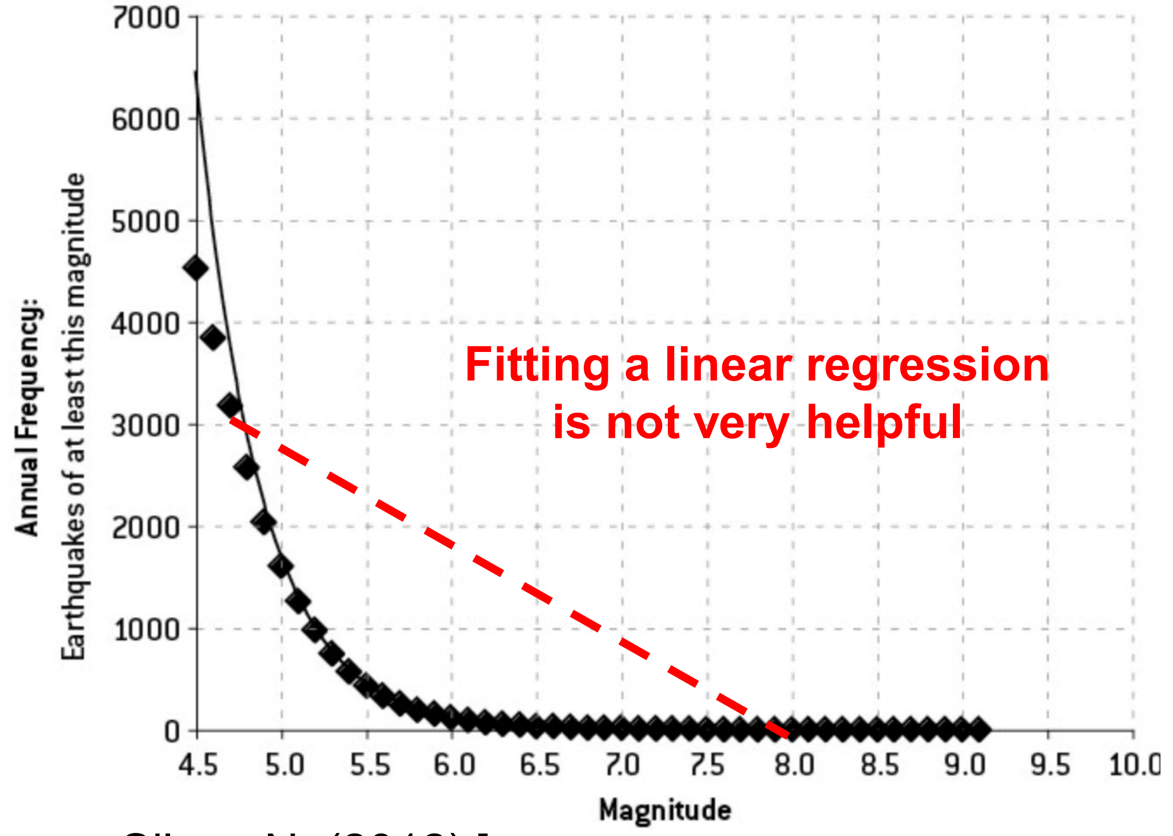
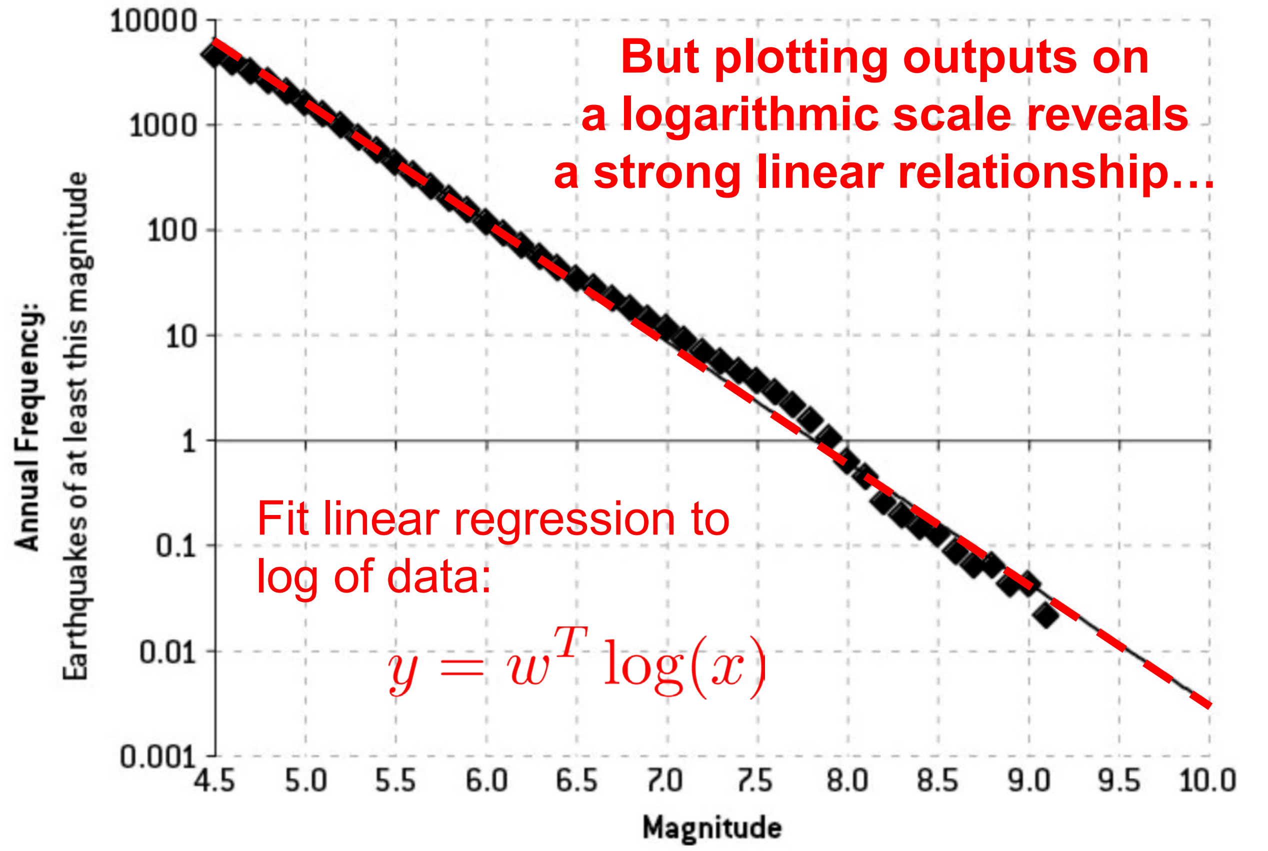


FIGURE 5-3B: WORLDWIDE EARTHQUAKE FREQUENCIES, JANUARY 1964–MARCH 2012, LOGARITHMIC SCALE



$$y = w^T \log(x)$$

[Source: Silver, N. (2012)]

Basis Functions

- A **basis function** can be any function of the input features X
- Define a set of m basis functions $\phi_1(x), \dots, \phi_m(x)$
- Fit a linear regression model in terms of basis functions,

$$y = \sum_{i=1}^{m} w_i \phi_i(x) = w^T \phi(x)$$

- Regression model is **linear** in the basis transformations
- Regression model is **nonlinear** in the original features *X*

Can we fit a regression in the same way? Is there a Bayesian Interpretation?

YES and YES

Linear Regression

Recall the ordinary least squares solution is given by,

$$\mathbf{X} = \begin{pmatrix} 1 & x_{11} & \dots & x_{1D} \\ 1 & x_{21} & \dots & x_{2D} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_{N1} & \dots & x_{ND} \end{pmatrix} \qquad \mathbf{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix} \qquad w^{\text{OLS}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Design Matrix
(each training input on a column)

Vector of
Training labels

Can similarly solve in terms of basis functions,

$$\mathbf{\Phi} = \begin{pmatrix} 1 & \phi_1(x_1) & \dots & \phi_M(x_1) \\ 1 & \phi_1(x_2) & \dots & \phi_M(x_2) \\ \vdots & \vdots & \vdots & \vdots \\ 1 & \phi_1(x_N) & \dots & \phi_M(x_N) \end{pmatrix} \qquad w^{\text{OLS}} = (\mathbf{\Phi}^T\mathbf{\Phi})^{-1}\mathbf{\Phi}^T\mathbf{y}$$

Posterior Distribution is Gaussian

In general we can have an arbitrary prior covariance,

$$w \sim \mathcal{N}(0, \Sigma_p) \qquad y \mid w, \phi(x) \sim \mathcal{N}(w^T \phi(x), \sigma_n^2)$$

Weight posterior is Gaussian (yay for Gaussian-Gaussian conjugacy),

$$p(w \mid \mathbf{\Phi}, \mathbf{y}) = \mathcal{N}(\underbrace{\frac{1}{\sigma_n^2} A^{-1} \mathbf{\Phi} \mathbf{y}}_{\text{MAP Estimate}}, A^{-1})$$

Where posterior covariance is,

$$A = \frac{1}{\sigma_n^2} \mathbf{\Phi} \mathbf{\Phi}^{\mathbf{T}} + \Sigma_p^{-1}$$

This is slightly more general than standard L2-regularized Regression

Posterior Predictive

Often we don't care about weights…we just want to predict the function value y_* at some new point x_* :

$$p(y_* \mid x_*, \mathbf{X}, \mathbf{y}) = \int p(y_* \mid x_*, w) p(w \mid \mathbf{X}, \mathbf{y}) \, dw$$

$$= \mathcal{N}\left(y_* \mid \frac{1}{\sigma_n^2} \phi(x_*)^T A^{-1} \mathbf{\Phi} \mathbf{y}, \phi(x_*)^T A^{-1} \phi(x_*)\right)$$

- To make predictions we need to invert $A = \sigma_n^{-2} \mathbf{\Phi} \mathbf{\Phi}^{\mathbf{T}} + \Sigma_p^{-1}$
- For N training data this is an NxN matrix and takes time $\mathcal{O}(m^3)$
- With a little algebra we can reduce this to $\mathcal{O}(N^3)$ for features $x \in \mathbb{R}^N$
- Beneficial when N < m (obviously)

Kernel Trick

Change notation to emphasize that we are predicting a *function value:*

$$f_* = f(x_*) = y_* = w^T \phi(x_*)$$

Our original posterior predictive,

Inversion of mXm matrix

$$f_* | \mathbf{x}_*, X, \mathbf{y} \sim \mathcal{N}\left(\frac{1}{\sigma_n^2}\phi(\mathbf{x}_*)^\top A^{-1}\Phi\mathbf{y},\ \phi(\mathbf{x}_*)^\top A^{-1}\phi(\mathbf{x}_*)\right)$$

Define an NxN **kernel matrix** as,

$$K = \mathbf{\Phi^T \Sigma_p \Phi}$$

After algebra…posterior predictive is equivalent to:

$$f_* | \mathbf{x}_*, X, \mathbf{y} \sim \mathcal{N}\left(\phi_*^\top \Sigma_p \Phi \underbrace{(K + \sigma_n^2 I)^{-1}}_{\text{Inversion of NxN matrix}}\mathbf{y},\ \phi_*^\top \Sigma_p \phi_* - \phi_*^\top \Sigma_p \Phi (K + \sigma_n^2 I)^{-1} \Phi^\top \Sigma_p \phi_*\right)$$

Shorthand for $\phi(\mathbf{x}_*)$

Kernel Trick

Our "kernelized" posterior predictive:

$$f_* | \mathbf{x}_*, X, \mathbf{y} \sim \mathcal{N}\big(\phi_*^\top \Sigma_p \Phi (K + \sigma_n^2 I)^{-1} \mathbf{y},\ \phi_*^\top \Sigma_p \phi_* - \phi_*^\top \Sigma_p \Phi (K + \sigma_n^2 I)^{-1} \Phi^\top \Sigma_p \phi_*\big)$$

Notice that basis functions always enter in one of three forms:

$$\Phi^\top \Sigma_p \Phi, \quad \phi_*^\top \Sigma_p \phi_* \qquad \text{or} \qquad \phi_*^\top \Sigma_p \Phi,$$

Define *kernel function* that expresses all of these for any pair *(x,x')*:

$$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^\top \Sigma_p \phi(\mathbf{x}')$$

Since Σ_p is positive semidefinite we can express as inner product:

$$k(x, x') = \psi(x)^T \psi(x') \qquad \text{where} \qquad \psi(x) = \Sigma_p^{1/2} \phi(x)$$

Features vs. Basis vs. Kernels

Features X

- Provided as N-dimensional inputs, requires inversion of NxN matrix
- May not be appropriate for linear model

Basis $\phi(\mathbf{x})$

- m-dimensional transformation of features
- Requires inversion of mXm matrix
- Can be made more appropriate for linear model

Kernel $k(x, x') = \phi(x)^T \phi(x)$

- Basis representation doesn’t need to be made explicit
- Requires inversion of NxN matrix
- Often easier to define kernel on pairs of features than basis functions

Kernel Functions

Example The *linear basis* $\phi(x) = x$ produces the kernel,

$$\kappa(x, x') = \phi(x)^T \phi(x') = x^T x'$$

It is often easier to directly specify the kernel rather than the basis function…

Example Gaussian kernel models similarity according to an unnormalized Gaussian distribution,

$$\kappa(x, x') = \exp\left(-\frac{1}{2\sigma^2}(x - x')^2\right)$$

Also called a *radial basis function* (RBF)

Note Despite the name, this is **not** a Gaussian probability density. It is unnormalized.

Corresponding basis is infinite-dimensional vector!

Kernel Functions

Given *any* set of data $\{x_i\}_{i=1}^N$ a necessary and sufficient condition of a valid kernel function is that the NxN **gram matrix**,

$$\mathbf{K} = \begin{pmatrix} \kappa(x_1, x_1) & \kappa(x_1, x_2) & \dots & \kappa(x_1, x_N) \\ \kappa(x_2, x_1) & \kappa(x_2, x_2) & \dots & \kappa(x_2, x_N) \\ \vdots & \vdots & \vdots & \vdots \\ \kappa(x_N, x_1) & \kappa(x_N, x_2) & \dots & \kappa(x_N, x_N) \end{pmatrix}$$

Is a *symmetric positive semidefinite matrix.*

Function-Space View

Recall posterior predictive function is a Gaussian over function values,

$$f_* \mid x_*, \mathbf{X}, \mathbf{y} \sim \mathcal{N}(\cdot, \cdot)$$

- So, we can predict the function *at any input*
- And, we can do this at *many inputs*
- So, we have a predictive distribution *over a class of functions*
- Note that this explicitly marginalizes out regression weights (w)
- We call this a **Gaussian Process**

Gaussian Process

Definition 2.1 *A* Gaussian process *is a collection of random variables, any finite number of which have a joint Gaussian distribution.* □

The Gaussian Process (GP) is completely specified by it mean and covariance functions:

$$\begin{aligned} m(\mathbf{x}) &= \mathbb{E}[f(\mathbf{x})], \\ k(\mathbf{x}, \mathbf{x}') &= \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))] \end{aligned}$$

We say that a function f(x) is distributed as a GP with the notation,

$$f(\mathbf{x}) \sim \mathcal{GP}\big(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')\big)$$

- Every draw from the GP is a function f(x)
- In practice, we draw f(x) evaluated at a set of points as a vector with Gaussian distribution (per GP Definition)

Gaussian Process → Bayesian Linear Regression

Returning to our Bayesian linear regression we have GP moments,

$$\begin{aligned}\mathbb{E}[f(\mathbf{x})] &= \phi(\mathbf{x})^\top \mathbb{E}[\mathbf{w}] = 0,\\ \mathbb{E}[f(\mathbf{x})f(\mathbf{x}')] &= \phi(\mathbf{x})^\top \mathbb{E}[\mathbf{w}\mathbf{w}^\top]\phi(\mathbf{x}') = \phi(\mathbf{x})^\top \Sigma_p \phi(\mathbf{x}')\end{aligned}$$

By definition of a GP any vector of function values is jointly Gaussian

$$\begin{pmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_\ell) \end{pmatrix} \sim \mathcal{N}(0, \mathbf{K}(\mathbf{x}, \mathbf{x}))$$

Kernel matrix evaluated at points $x_1, x_2, \ldots, x_\ell$

This allows us to draw random functions from a GP prior

Posterior Inference

Consider joint over **f** training points $\{(x_i, f_i)\}_{i=1}^N$ and query points $\mathbf{f}_*$:

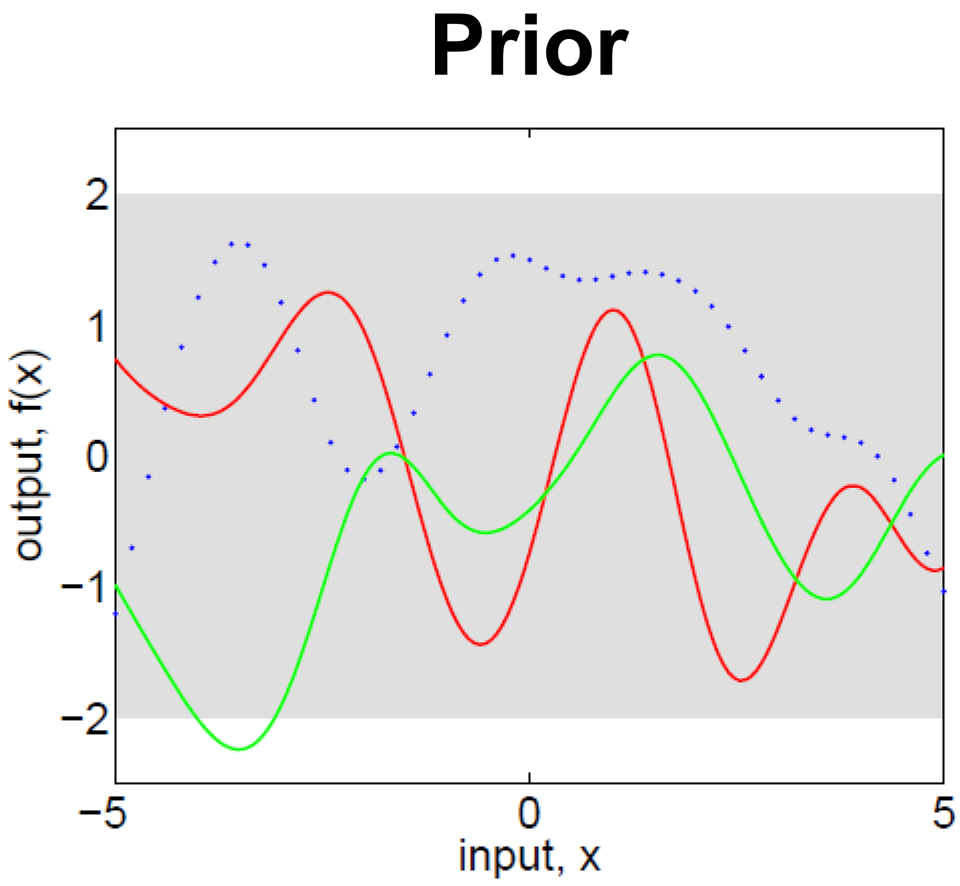
$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(X,X) & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix}\right)$$

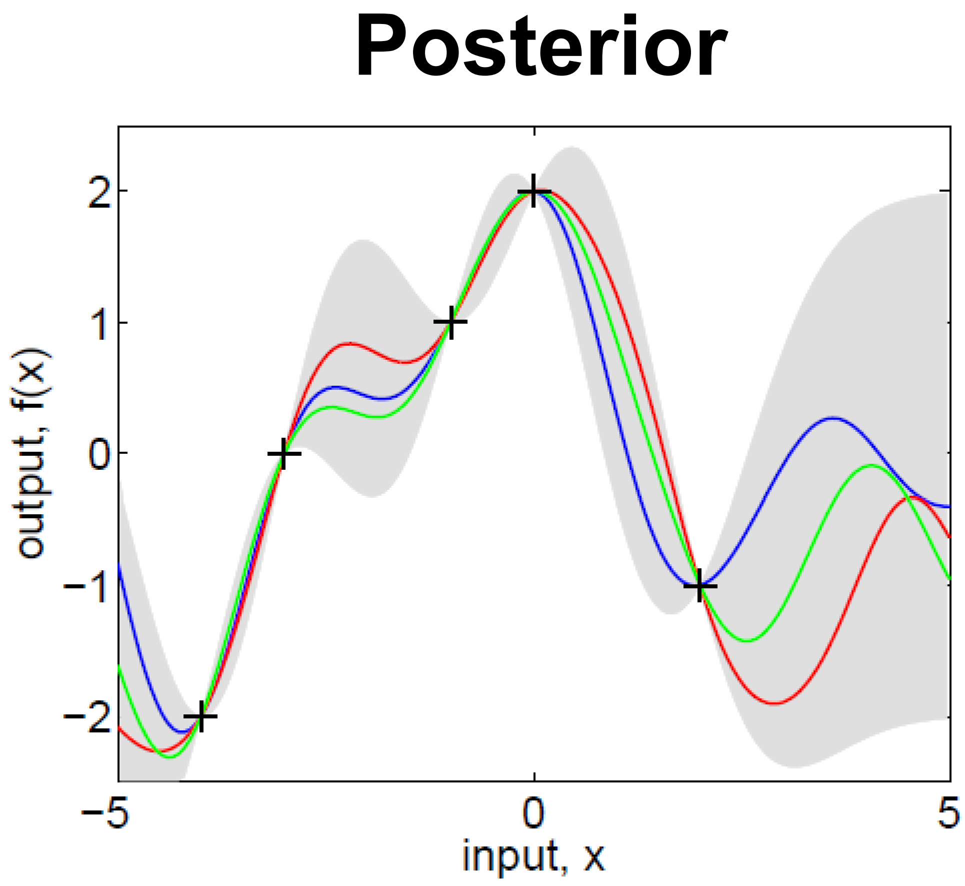
Gaussians are closed under conditioning, so posterior is:

$$\begin{aligned} \mathbf{f}_* | X_*, X, \mathbf{f} \sim \mathcal{N}\big(& K(X_*,X)K(X,X)^{-1}\mathbf{f}, \\ & K(X_*,X_*) - K(X_*,X)K(X,X)^{-1}K(X,X_*)\big) \end{aligned}$$

- Given training set, can predict function values at any query points
- Gaussian distribution quantifies uncertainty over predictions
- Marginalizes out regression parameters (w)

Example





Covariance Kernel = Gaussian (Radial Basis Function)

$$\operatorname{cov}\left(f(\mathbf{x}_p), f(\mathbf{x}_q)\right) = k(\mathbf{x}_p, \mathbf{x}_q) = \exp\left(-\tfrac{1}{2}|\mathbf{x}_p - \mathbf{x}_q|^2\right)$$

Predicting with Noisy Function Evaluations

Previous example assumed that we directly observe function, y=f(x), but it is more realistic to receive *noisy* function evaluations,

$$y = f(x) + \epsilon \qquad \text{where} \qquad \epsilon \sim \mathcal{N}(0, \sigma_n^2)$$

Simple adjustment to the covariance kernel,

$$\text{cov}(y_p, y_q) \; = \; k(\mathbf{x}_p, \mathbf{x}_q) + \sigma_n^2 \delta_{pq} \;\; \text{or} \;\; \text{cov}(\mathbf{y}) \; = \; K(X, X) + \sigma_n^2 I$$

Posterior predictive distribution is,

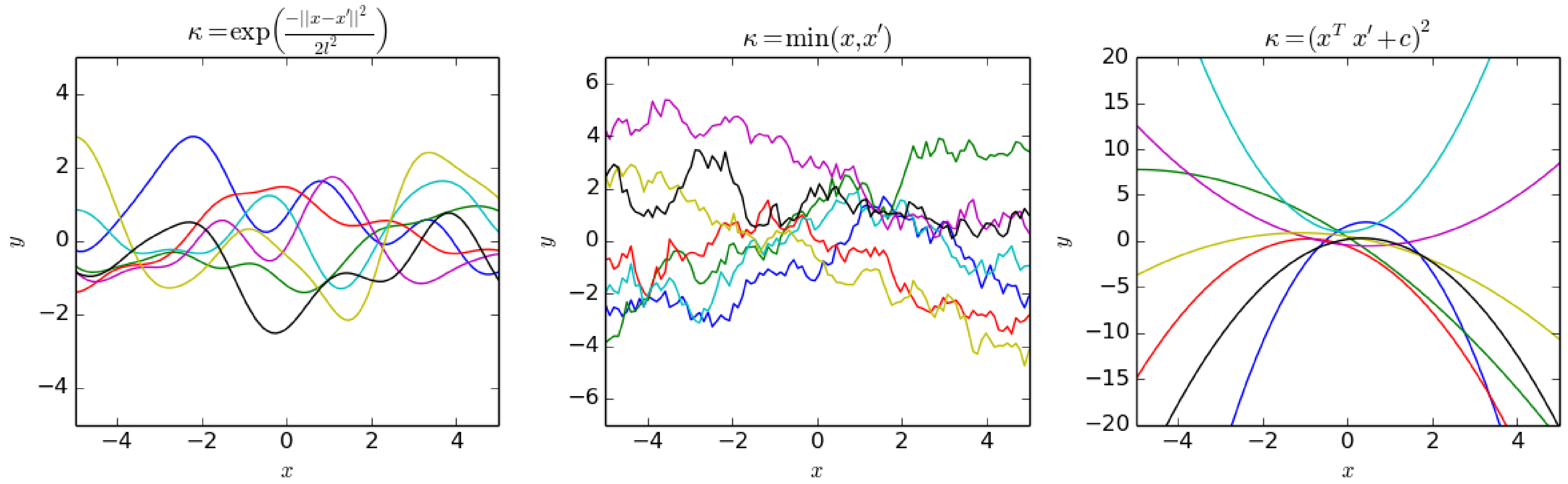
$$\mathbf{f}_* | X, \mathbf{y}, X_* \sim \mathcal{N}\big(\bar{\mathbf{f}}_*, \, \text{cov}(\mathbf{f}_*)\big), \;\; \text{where}$$

$$\bar{\mathbf{f}}_* \triangleq \mathbb{E}[\mathbf{f}_* | X, \mathbf{y}, X_*] \; = \; K(X_*, X)[K(X, X) + \sigma_n^2 I]^{-1} \mathbf{y},$$

$$\text{cov}(\mathbf{f}_*) = K(X_*, X_*) - K(X_*, X)\big[K(X, X) + \sigma_n^2 I\big]^{-1} K(X, X_*)$$

Kernel Choice

The choice of kernel controls the support of a GP…



- **Stationary** kernels are functions of a distance metric: $k(x, x') = k(\rho(x, x'))$
- **Nonstationary** kernels vary based on location of inputs x and x'
- **Periodic** kernels achieved by mapping to $u(x) = (\cos(x), \sin(x))$

Techniques for Constructing New Kernels.

Given valid kernels $k_1(\mathbf{x}, \mathbf{x}')$ and $k_2(\mathbf{x}, \mathbf{x}')$, the following new kernels will also be valid:

$$k(\mathbf{x}, \mathbf{x}') = ck_1(\mathbf{x}, \mathbf{x}') \tag{6.13}$$

$$k(\mathbf{x}, \mathbf{x}') = f(\mathbf{x})k_1(\mathbf{x}, \mathbf{x}')f(\mathbf{x}') \tag{6.14}$$

$$k(\mathbf{x}, \mathbf{x}') = q\left(k_1(\mathbf{x}, \mathbf{x}')\right) \tag{6.15}$$

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(k_1(\mathbf{x}, \mathbf{x}')\right) \tag{6.16}$$

$$k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}') \tag{6.17}$$

$$k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}')k_2(\mathbf{x}, \mathbf{x}') \tag{6.18}$$

$$k(\mathbf{x}, \mathbf{x}') = k_3\left(\boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}')\right) \tag{6.19}$$

$$k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x}' \tag{6.20}$$

$$k(\mathbf{x}, \mathbf{x}') = k_a(\mathbf{x}_a, \mathbf{x}_a') + k_b(\mathbf{x}_b, \mathbf{x}_b') \tag{6.21}$$

$$k(\mathbf{x}, \mathbf{x}') = k_a(\mathbf{x}_a, \mathbf{x}_a')k_b(\mathbf{x}_b, \mathbf{x}_b') \tag{6.22}$$

[Source: Bishop, C.]

Summary: Bayesian Linear Regression

- Good old linear regression that we know and love…
- L2 regularized least squares = Bayesian linear regression with gaussian prior on weights w
- More generally: any regularizer corresponds to some prior
- Bayesian perspective allows us to integrate out weights
- Predictive distribution p(y* | x*,X,y) predicts function at new points x*
- Everything is closed-form Gaussian and O(N^3) complexity
- Can map features X to basis functions $\phi(x) \in \mathbb{R}^m$ for better linear fits
- Can do some algebra to reduce computation to O(m^3)

Summary: Gaussian Processes

- Basis functions show up as inner products in posterior predictive
- Define kernel function $k(x, x') = \phi(x)^T \phi(x')$ and work in *kernel space*
- This is known as **the kernel trick**
- Avoids explicit definition of basis functions (back to O(N^3) complexity)
- Defines prior distribution on functions called **Gaussian Process (GP)**
- GP = Bayesian Linear Regression for specific kernel choice
- GP defines prior over space of functions
- Function evaluated at any finite set of points is Gaussian distributed
- Prediction / inference closed-form based on Gaussian manipulation