Introduction to Machine Learning

Brown University CSCI 1950-F, Spring 2012
Prof. Erik Sudderth

Lecture 15:
Online Learning: Stochastic Gradient Descent
Perceptron Algorithm
Kernel Methods

Many figures courtesy Kevin Murphy's textbook,
Machine Learning: A Probabilistic Perspective

Batch versus Online Learning

- Many learning algorithms we've seen can be phrased as *batch* minimization of the following objective:

$$f(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} f(\boldsymbol{\theta}, \mathbf{z}_i) \qquad \mathbf{z}_i = (\mathbf{x}_i, y_i) \qquad f(\boldsymbol{\theta}, \mathbf{z}_i) = -\log p(y_i|\mathbf{x}_i, \boldsymbol{\theta})$$

(one of N data points)
(for ML, similar for MAP)

- This produces effective prediction algorithms, but can require significant *computation* and *storage* for training
- We can do *online learning* from *streaming data* via *stochastic gradient* descent:

$$\boldsymbol{\theta}_{k+1} = \mathrm{proj}_{\Theta}(\boldsymbol{\theta}_k - \eta_k \mathbf{g}_k) \qquad \mathbf{g}_k = \nabla f(\boldsymbol{\theta}_k, \mathbf{z}_k)$$

- SGD takes a small step based on single observations "sampled" from the data's empirical distribution:

$$f(\boldsymbol{\theta}) = \mathbb{E}\left[f(\boldsymbol{\theta}, \mathbf{z})\right] \qquad p(z) = \frac{1}{N}\sum_{i=1}^{N} \delta_{z_i}(z)$$

Stochastic Gradient Descent

- How can we produce a single parameter estimate?

$$\overline{\boldsymbol{\theta}}_k = \frac{1}{k}\sum_{t=1}^{k}\boldsymbol{\theta}_t \qquad \overline{\boldsymbol{\theta}}_k = \overline{\boldsymbol{\theta}}_{k-1} - \frac{1}{k}(\overline{\boldsymbol{\theta}}_{k-1} - \boldsymbol{\theta}_k)$$

Polyak-Ruppert averaging

- How should we set the step size η_k?

$$\eta_k = (\tau_0 + k)^{-\kappa} \qquad \begin{matrix}\tau_0 \geq 0 \\ \kappa \in (0.5, 1]\end{matrix} \qquad \sum_{k=1}^{\infty}\eta_k = \infty, \ \sum_{k=1}^{\infty}\eta_k^2 < \infty$$

Robbins-Monro conditions

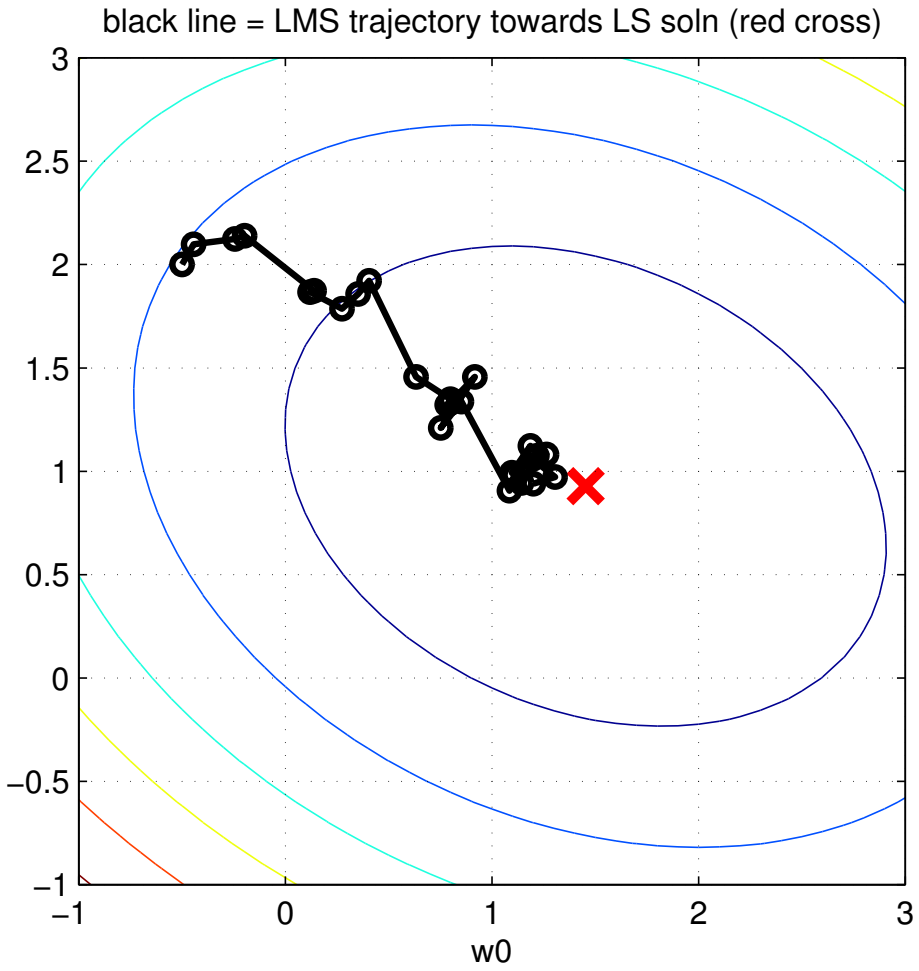
Conventional batch step size rules fail (in theory & practice)

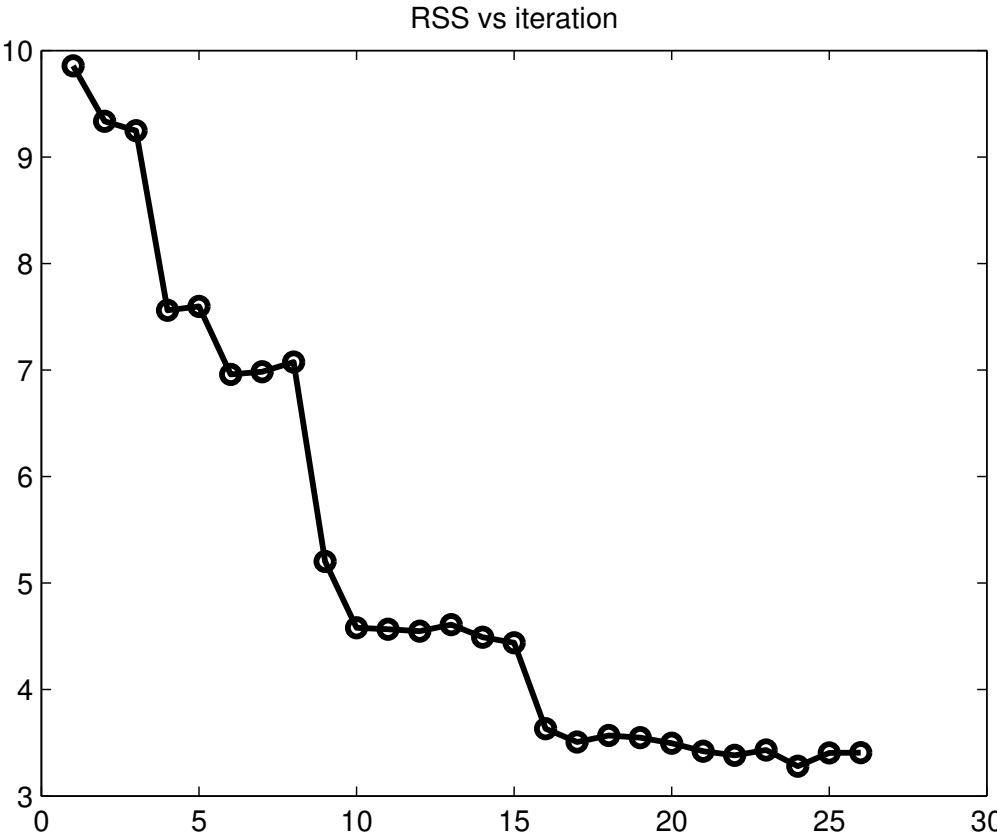
- How does this work in practice?
  - Excellent for big datasets, but tuning parameters tricky
  - Refinement: Take batches of data for each step: $1 < B \ll N$

$$\boldsymbol{\theta}_{k+1} = \text{proj}_{\Theta}(\boldsymbol{\theta}_k - \eta_k \mathbf{g}_k) \qquad \mathbf{g}_k = \nabla f(\boldsymbol{\theta}_k, \mathbf{z}_k)$$

$$f(\boldsymbol{\theta}) = \mathbb{E}\left[f(\boldsymbol{\theta}, \mathbf{z})\right] \qquad p(z) = \frac{1}{N}\sum_{i=1}^{N}\delta_{z_i}(z)$$

Least Mean Squares (LMS)





$$f(\theta, y_i, x_i) = \frac{1}{2}(y_i - \theta^T \phi(x_i))^2$$

Stochastic gradient descent applied to linear regression model:

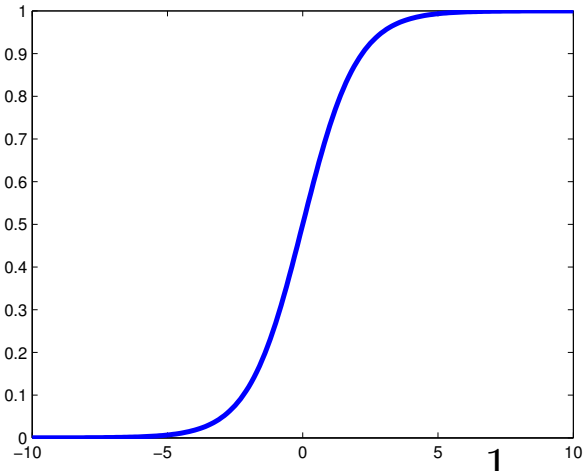
$$\theta_{k+1} = \theta_k + \eta_k(y_k - \hat{y}_k)\phi(x_k) \qquad \hat{y}_k = \theta_k^T \phi(x_k)$$

SGD for Logistic Regression

$$p(y_i \mid x_i, \theta) = \mathrm{Ber}(y_i \mid \mu_i)$$

$$\mu_i = \sigma(\theta^T \phi(x_i))$$

$$f(\theta) = -\sum_{i=1}^{N} [y_i \log \mu_i + (1 - y_i) \log(1 - \mu_i)]$$



$$\sigma(z) = \mathrm{sigm}(z) = \frac{1}{1 + e^{-z}}$$

- Batch gradient function:

$$\nabla f(\theta) = \sum_{i=1}^{N} (\mu_i - y_i)\phi(x_i)$$

- Stochastic gradient descent:

$$\theta_{k+1} = \theta_k + \eta_k (y_k - \mu_k)\phi(x_k)$$

$$\mu_k = \sigma(\theta_k^T \phi(x_k)) \qquad 0 < \mu_k < 1$$

Perceptron MARK 1 Computer

Frank Rosenblatt, late 1950s

Decision Rule: $\hat{y}_i = \mathbb{I}(\theta^T \phi(x_i) > 0)$

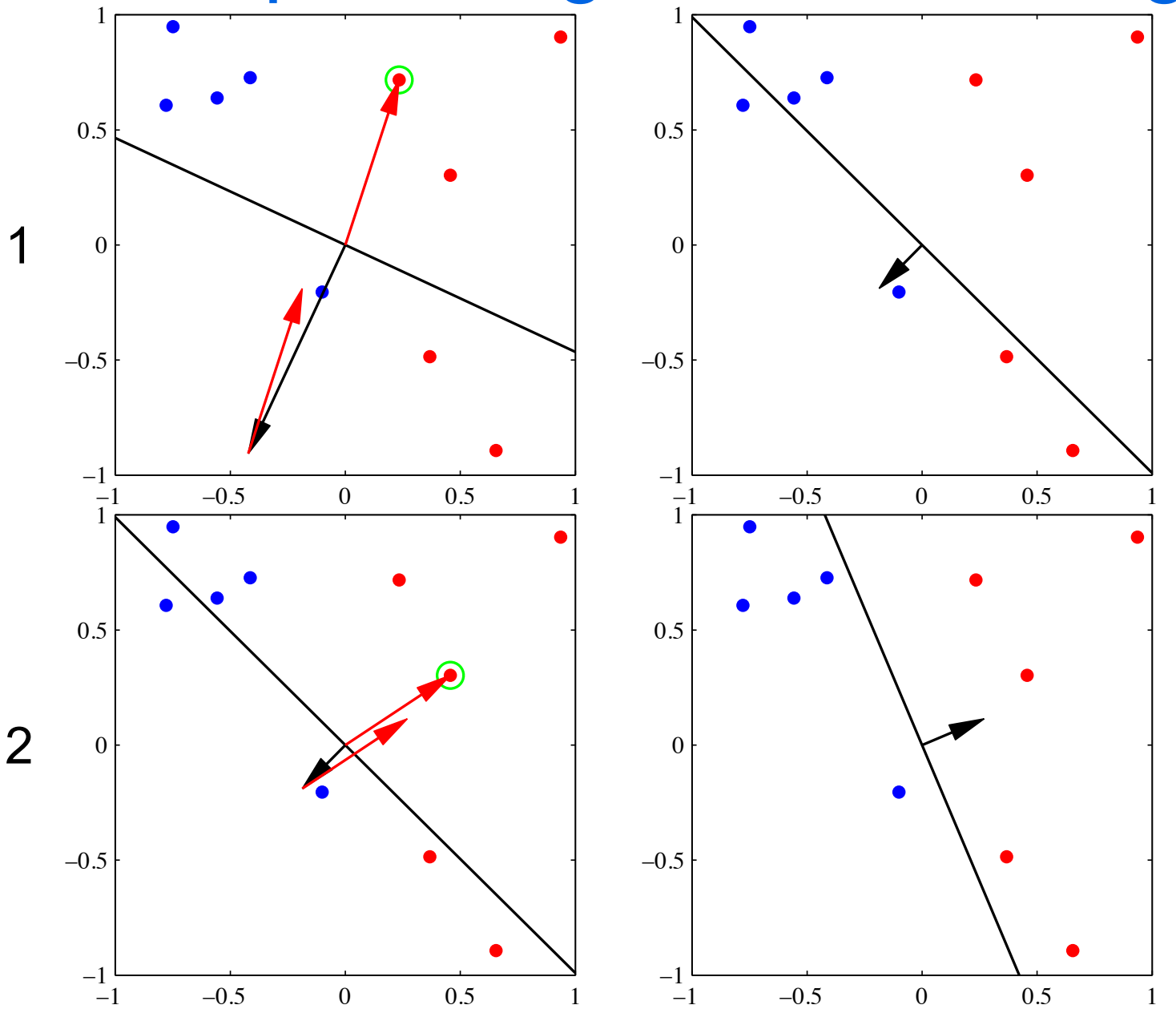
Learning Rule:

$$\text{If } \hat{y}_k = y_k, \theta_{k+1} = \theta_k$$

$$\text{If } \hat{y}_k \neq y_k, \theta_{k+1} = \theta_k + \tilde{y}_k \phi(x_k)$$

$$\tilde{y}_k = 2y_k - 1 \in \{+1, -1\}$$

Perceptron Algorithm Convergence



C. Bishop, Pattern Recognition & Machine Learning

Perceptron Algorithm Properties

Strengths

- Guaranteed to converge if data linearly separable (in feature space; reduces angle to true separators)
- Easy to construct kernel representation of algorithm

Weaknesses

- May be slow to converge (worst-case performance poor)
- If data not linearly separable, will never converge
- Solution depends on order data visited; no notion of a "best" separating hyperplane
- Non-probabilistic: No measure of confidence in decisions, difficult to generalize to other problems

Covariance Matrices

- Eigenvalues and eigenvectors:

$$\Sigma u_i = \lambda_i u_i, i = 1, \ldots, d$$

- For a *symmetric* matrix:

$$\lambda_i \in \mathbb{R} \qquad u_i^T u_i = 1 \qquad u_i^T u_j = 0$$

$$\Sigma = U \Lambda U^T = \sum_{i=1}^{d} \lambda_i u_i u_i^T$$

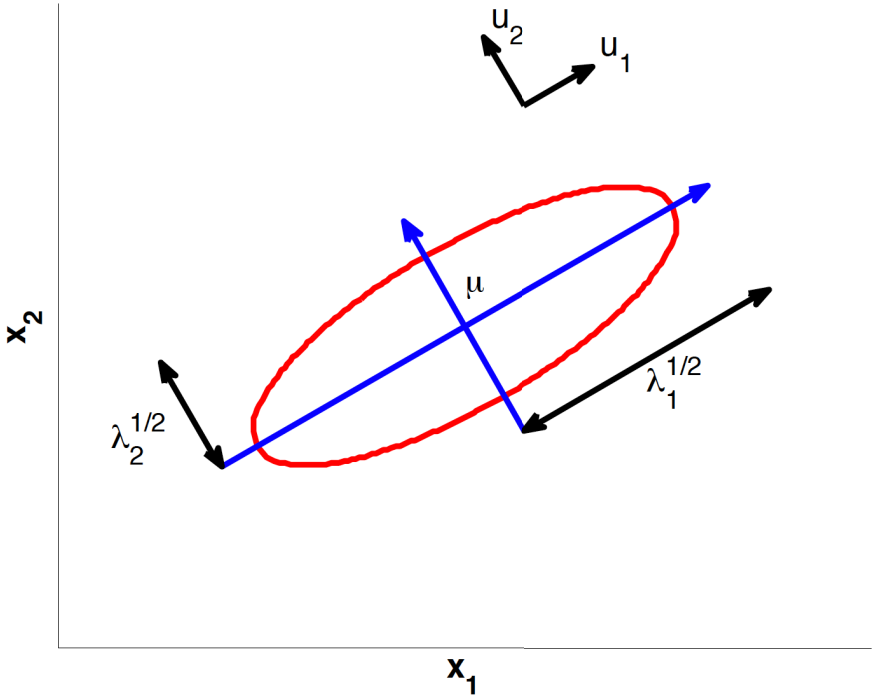
- For a *positive semidefinite* matrix:

$$\lambda_i \geq 0$$

- For a *positive definite* matrix:

$$\lambda_i > 0$$

$$\Sigma^{-1} = U \Lambda^{-1} U^T = \sum_{i=1}^{d} \frac{1}{\lambda_i} u_i u_i^T$$



$$\Delta^2 = (\mathbf{x} - \boldsymbol{\mu})^{\mathrm{T}} \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})$$

$$\Delta^2 = \sum_{i=1}^{D} \frac{y_i^2}{\lambda_i}$$

$$y_i = u_i^T (x - \mu)$$

Mercer Kernel Functions

$\mathcal{X} \longrightarrow$ arbitrary input space (vectors, functions, strings, graphs, …)

- A *kernel function* maps pairs of inputs to real numbers:

$$k : \mathcal{X} \times \mathcal{X} \to \mathbb{R} \qquad k(x_i, x_j) = k(x_j, x_i)$$

Intuition: Larger values indicate inputs are "more similar"

- A kernel function is *positive semidefinite* if and only if for *any* $n \geq 1$, and *any* $x = \{x_1, x_2, \ldots, x_n\}$, the *Gram matrix* is positive semidefinite:

$$K \in \mathbb{R}^{n \times n} \qquad K_{ij} = k(x_i, x_j)$$

- Mercer's Theorem: Assuming certain technical conditions, every positive definite kernel function can be represented as

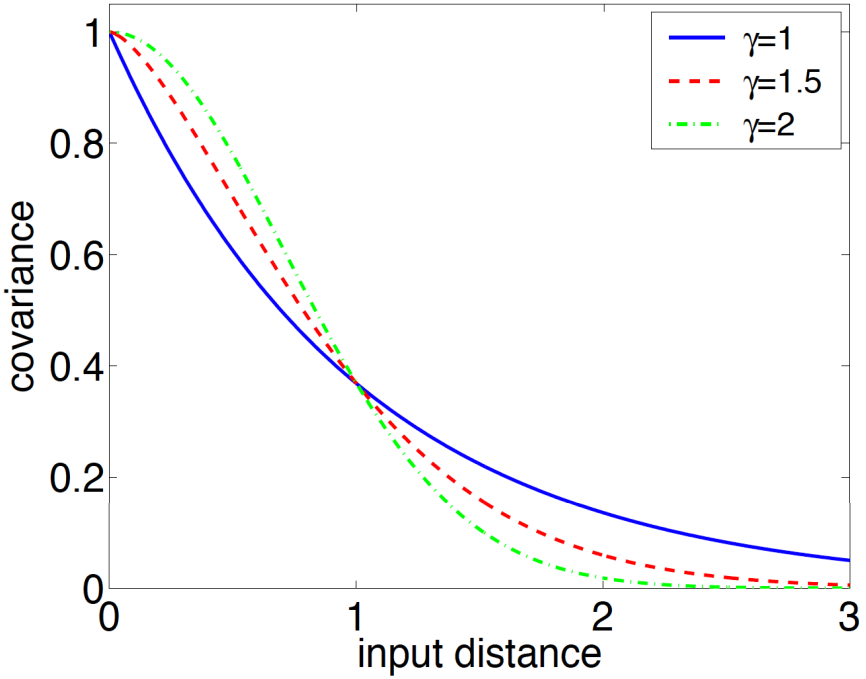
$$k(x_i, x_j) = \sum_{\ell=1}^{d} \phi_\ell(x_i)\phi_\ell(x_j)$$

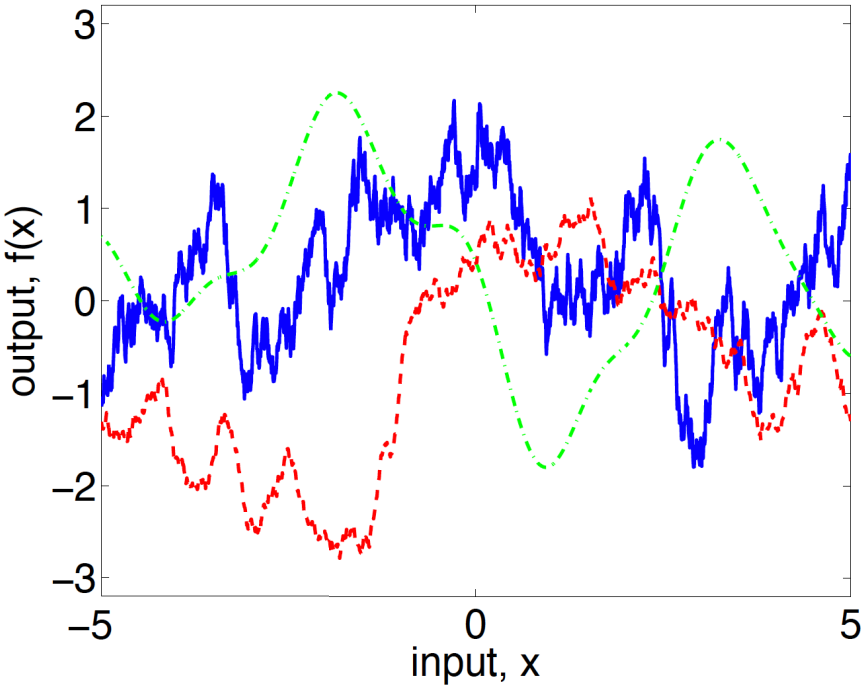
for some feature mapping ϕ (but may need $d \to \infty$)

Exponential Kernels

$\mathcal{X} \rightarrow$ real vectors of some fixed dimension

$$k(x_i, x_j) = \exp\left\{-\left(\frac{|x_i - x_j|}{\sigma}\right)^{\gamma}\right\} \qquad 0 < \gamma \leq 2$$





We can construct a covariance matrix by evaluating kernel at any set of inputs, and then sample from the zero-mean Gaussian distribution with that covariance. This is a Gaussian process.

Polynomial Kernels

$\mathcal{X} \longrightarrow$ real vectors of some fixed dimension

$$\kappa(\mathbf{x}, \mathbf{x}') = (\gamma \mathbf{x}^T \mathbf{x}' + r)^M$$

$$\begin{aligned}(1 + \mathbf{x}^T \mathbf{x}')^2 &= (1 + x_1 x_1' + x_2 x_2')^2 \\ &= 1 + 2x_1 x_1' + 2x_2 x_2' + (x_1 x_1)^2 + (x_2 x_2')^2 + 2x_1 x_1' x_2 x_2'\end{aligned}$$

$$\phi(\mathbf{x}) = [1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1 x_2]^T$$

- The polynomial kernel has an explicit feature mapping, but the number of features grows *exponentially* with both the input dimension and the polynomial degree
- The squared exponential kernel requires an *infinite* number of features (roughly, radial basis functions at all possible locations in the input space)

String Kernels

$\mathcal{X} \longrightarrow$ strings of characters from some finite alphabet, of size *A*

Amino Acids $\{A, R, N, D, C, E, Q, G, H, I, L, K, M, F, P, S, T, W, Y, V\}$

x IPTSALVKETLALLSTHRTLLIANETLRIPVPVHKNHQLCTEEIFQGIGTLESQTVQGGTV
ERLFKNLSLIKKYIDGQKKKCGEERRRVNQFLDYLQEFLGVMNTEWI

x' PHRRDLCSRSIWLARKIRSDLTALTESYVKHQGLWSELTEAERLQENLQAYRTFHVLLA
RLLEDQQVHFTPTEGDFHQAIHTLLLQVAAFAYQIEELMILLEYKIPRNEADGMLFEKK
LWGLKVLQELSQWTVRSIHDLRFISSHQTGIP

- Feature vector: Count of number of times that *every substring*, of every possible length, occurs within string

$$D = A + A^2 + A^3 + A^4 + \cdots$$

- Using *suffix trees*, the kernel can be evaluated in time *linear* in the length of the input strings

Kernels and Features

- What features lead to valid, positive semidefinite kernels?

  $k(x_i, x_j) = \phi(x_i)^T \phi(x_j)$ *is valid for any* $\phi : \mathcal{X} \to \mathbb{R}^d$

- When is a hypothesized kernel function positive semidefinite?

Can be tricky to verify whether underlying feature mapping exists.

- How can I build new kernel functions?

$$\bar{k}(x_i, x_j) = ck(x_i, x_j), c > 0$$
$$\bar{k}(x_i, x_j) = f(x_i)k(x_i, x_j)f(x_j) \text{ for any } f(x)$$
$$\bar{k}(x_i, x_j) = k_1(x_i, x_j) + k_2(x_i, x_j)$$
$$\bar{k}(x_i, x_j) = k_1(x_i, x_j)k_2(x_i, x_j)$$
$$\bar{k}(x_i, x_j) = \exp(k(x_i, x_j))$$

Kernelizing Learning Algorithms

- Start with any learning algorithm based on features $\phi(x)$
  (Don't worry that computing features might be expensive or impossible.)
- Manipulate steps in algorithm so that it depends not directly on features, but only their inner products: $k(x_i, x_j) = \phi(x_i)^T \phi(x_j)$
- Write code that only uses calls to kernel function
- Basic identity: Squared distance between feature vectors

$$||\phi(x_i) - \phi(x_j)||_2^2 = k(x_i, x_i) + k(x_j, x_j) - 2k(x_i, x_j)$$

  - Feature-based *nearest neighbor classification*
  - Feature-based *clustering algorithms* (later)
  - Feature-based *nearest centroid classification*:

$$\hat{y}_{\text{test}} = \arg\min_c ||\phi(x_{\text{test}}) - \mu_c||_2$$

$$\mu_c = \frac{1}{N_c} \sum_{i|y_i=c} \phi(x_i)$$

mean of the N_c training examples of class c

Kernelized Perceptron Algorithm

Decision Rule: $\hat{y}_{\text{test}} = \mathbb{I}(\theta^T \phi(x_{\text{test}}) > 0)$

Learning Rule:

$$\text{If } \hat{y}_k = y_k, \theta_{k+1} = \theta_k$$

$$\text{If } \hat{y}_k \neq y_k, \theta_{k+1} = \theta_k + \tilde{y}_k \phi(x_k)$$

$$\tilde{y}_k = 2y_k - 1 \in \{+1, -1\}$$

Representation:
D feature weights

Problem: May be intractable to compute/store $\phi(x_k), \theta_k$

Initialize with $\theta_0 = 0$. By induction, for all *k*

$$\theta_k = \sum_{i=1}^{N} s_{ik} \phi(x_i) \qquad \text{for some integers } s_{ik}$$

Representation:
N training example weights

Decision Rule: $\hat{y}_{\text{test}} = \mathbb{I}\left(\sum_{i=1}^{N} \hat{s}_i k(x_{\text{test}}, x_i) > 0 \right)$

Learning Rule:

$$\text{If } \hat{y}_k = y_k, s_{k,k+1} = s_{k,k}$$

$$\text{If } \hat{y}_k \neq y_k, s_{k,k+1} = s_{k,k} + \tilde{y}_k$$