ECE7252
Statistical Learning for Signal Processing

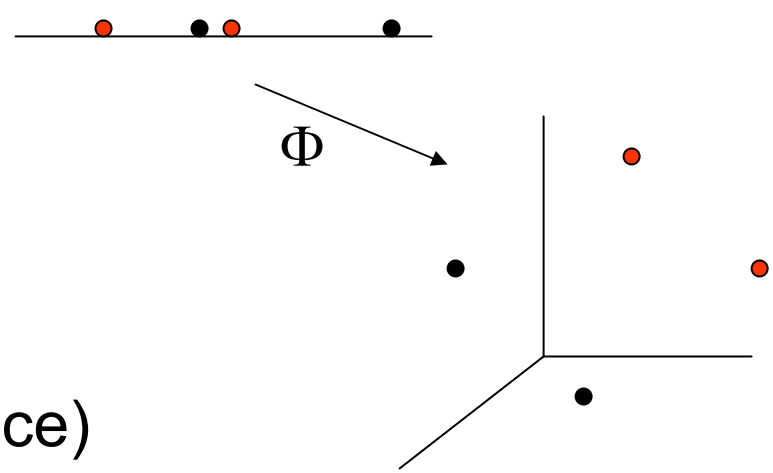
Lectures 17-18: Kernel-Based Learning

Chin-Hui Lee
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, GA 30332, USA
chl@ece.gatech.edu

Kernel and Feature Space

Primal vs. dual problems: generalized linear models vs. kernel classification $f(x) = w^T \phi(x) + b$

$$\Phi : x \to \Phi(x), \quad R^d \to F$$



non-linear mapping to F

1. high-D space L_2
2. infinite-D countable space :
3. function space (Hilbert space)

example: $(x, y) \to (x^2, y^2, \sqrt{2}xy)$

Kernel Trick

Note: In the dual representation we used the Gram matrix to express the solution

Kernel Trick: Replace

kernel

$$x \rightarrow \Phi(x),$$

$$G_{ij} =< x_i, x_j > \rightarrow G_{ij}^{\Phi} =< \Phi(x_i), \Phi(x_j) >= K(x_i, x_j)$$

If we use algorithms that only depend on the Gram-matrix, G, then we never have to know (compute) the actual features Φ

This is the crucial point of kernel methods

Properties of a Kernel

Definition: *A finitely positive semi-definite function* $k : x \times y \rightarrow R$ *is a symmetric function of its arguments for which matrices formed by restriction on any finite subset of points is positive semi-definite.*

$$\alpha^T K \alpha \geq 0 \quad \forall \alpha$$

Theorem: *A function* $k : x \times y \rightarrow R$ *can be written as* $k(x, y) = < \Phi(x), \Phi(y) >$ *where* $\Phi(x)$ *is a feature map* $x \rightarrow \Phi(x) \in F$ *iff k(x,y) satisfies the semi-definiteness.*

Relevance: We can now check if k(x,y) is a proper Kernel using only properties of k(x,y) itself, i.e. without the need to know the feature map!

Modularity

Kernel methods consist of two modules:

1) The choice of kernel (this is non-trivial)
2) The algorithm which takes kernels as input

Modularity: Any kernel can be used with any kernel-algorithm.

some kernels:

$$k(x,y) = e^{(-\|x-y\|^2/c)}$$

$$k(x,y) = (< x,y > +\theta)^d$$

$$k(x,y) = \tanh(\alpha < x,y > +\theta)$$

$$k(x,y) = \frac{1}{\sqrt{\| x-y \|^2 + c^2}}$$

some kernel algorithms:

- support vector machine
- Fisher discriminant analysis
- kernel regression
- kernel PCA
- kernel CCA

Learning Kernels

- All information is tunneled through the Gram-matrix information bottleneck.
- The real art is to pick an appropriate kernel.
  e.g. take the RBF kernel: $k(x,y) = e^{(-\|x-y\|^2/c)}$

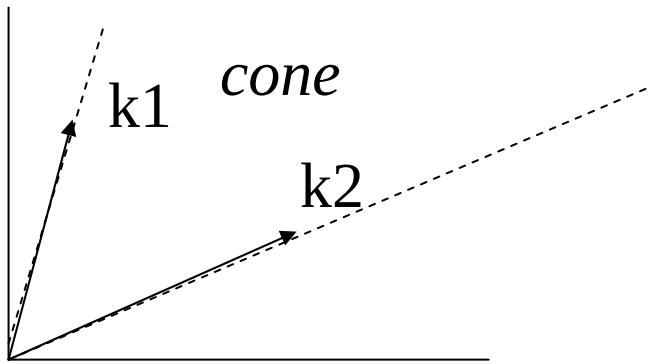
if c is very small: G=I (all data are dissimilar): over-fitting
if c is very large: G=1 (all data are very similar): under-fitting

We need to *learn* the kernel. Here is some ways to combine kernels:

$$\alpha k_1(x,y) + \beta k_2(x,y) = k(x,y) \quad \alpha,\beta \geq 0$$

$$k_1(x,y)k_2(x,y) = k(x,y)$$
$$k_1(\Phi(x),\Phi(y)) = k(x,y)$$

any positive polynomial



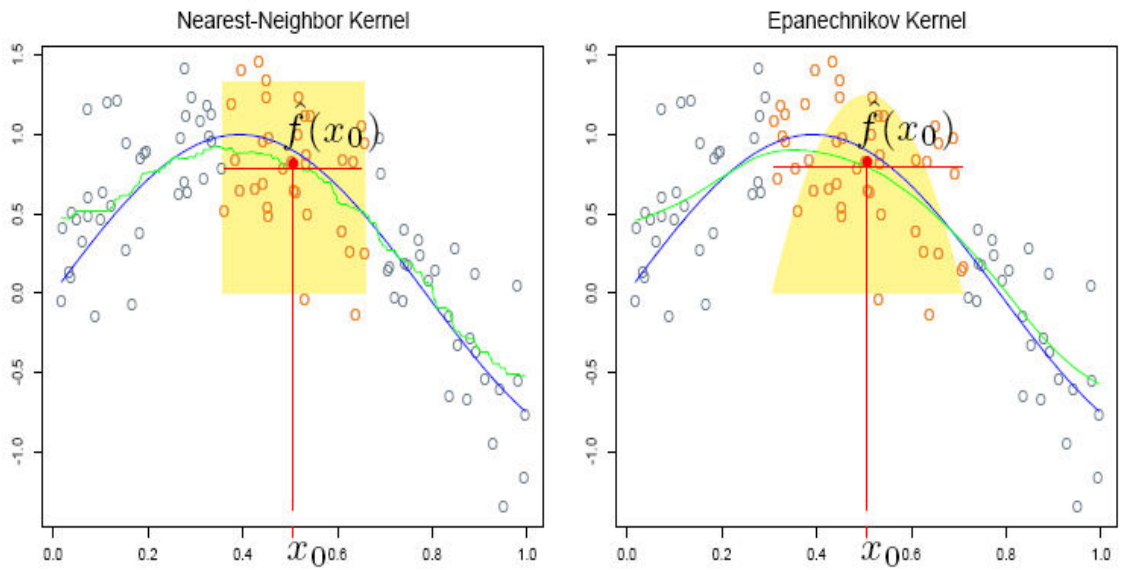
Kernel Methods: Key Points

- Essentially a regression (function fitting) technique
- Only the observations (training set) close to the query point are considered for regression computation
- While regressing, an observation point gets a weight that decreases as its distance from the query point increases
- The resulting approximation function is smooth
- These features are made possible by a function called kernel
- Requires very little training (*i.e.*, not many parameters to compute offline from the training set, not much offline computation needed)
- This kind of regression is known as memory based technique as it requires entire training set to be available while regressing

One-Dimensional Kernel Smoothers

- *KNN (k*-nearest neighbor) directly estimates *Pr(Y|X=x) by* assigning equal weight to all points in their neighborhood
- The average curve is often bumpy and discontinuous
- Instead assign weights that decrease smoothly with distance from the target points (training set)

$$\hat{f}(x) = Ave(y_i \mid x_i \in N_k(x))$$



Nadaraya-Watson Kernel-weighted Average

- N-W kernel weighted average:

$$\hat{f}(x_0) = \frac{\sum_{i=1}^{N} K_\lambda(x_0, x_i) y_i}{\sum_{i=1}^{N} K_\lambda(x_0, x_i)}$$

- K_λ is a kernel function:

where $K_\lambda(x_0, x) = D\left(\frac{|x - x_0|}{\lambda}\right)$

with $D(t) = \frac{3}{4}(1 - t^2)$ for $|t| \le 1$

Determine local width

Any smooth function K such that

$$K(x) \ge 0, \int K(x)dx = 1, \int xK(x)dx = 0 \text{ and } \int x^2 K(x)dx > 0$$

or more generally $K_\lambda(x_0, x) = D\left(\frac{|x - x_0|}{h_\lambda(x_0)}\right)$

- Typically K is also symmetric about 0

Some Points About Kernels

- $h_\lambda(x_0)$ is a width function also dependent on λ
- For the N-W kernel average $h_\lambda(x_0) = \lambda$
- For KNN average $h_\lambda(x_0) = |x_0 - x_{[k]}|$, where $x_{[k]}$ is the k^{th} closest x_i to x_0
- λ determines the width of local neighborhood and degree of smoothness
- λ also controls the tradeoff between bias and variance
  - Larger λ makes lower variance but higher bias (Why?)
- λ is computed from training data (how?)

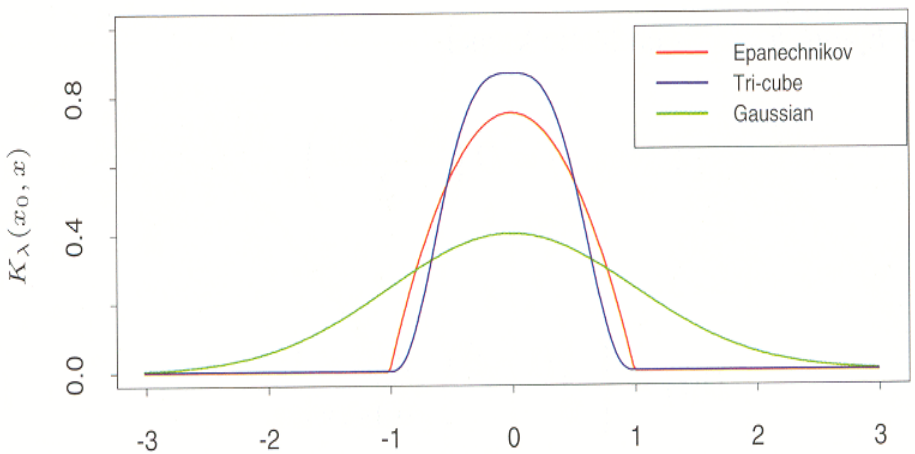
Example Kernel Functions

- Epanechnikov quadratic kernel (used in N-W method)

$$K_\lambda(x_0,x)=D\left(\frac{|x-x_0|}{\lambda}\right) \qquad D(t)=\begin{cases}\frac{3}{4}(1-t^2) & \text{if } |t|\le 1;\\ 0 & \text{otherwise}\end{cases}$$

- Tri-cube kernel $K_\lambda(x_0,x)=D\left(\frac{|x-x_0|}{\lambda}\right) \qquad D(t)=\begin{cases}(1-|t|^3)^3 & \text{if } |t|\le 1;\\ 0 & \text{otherwise.}\end{cases}$

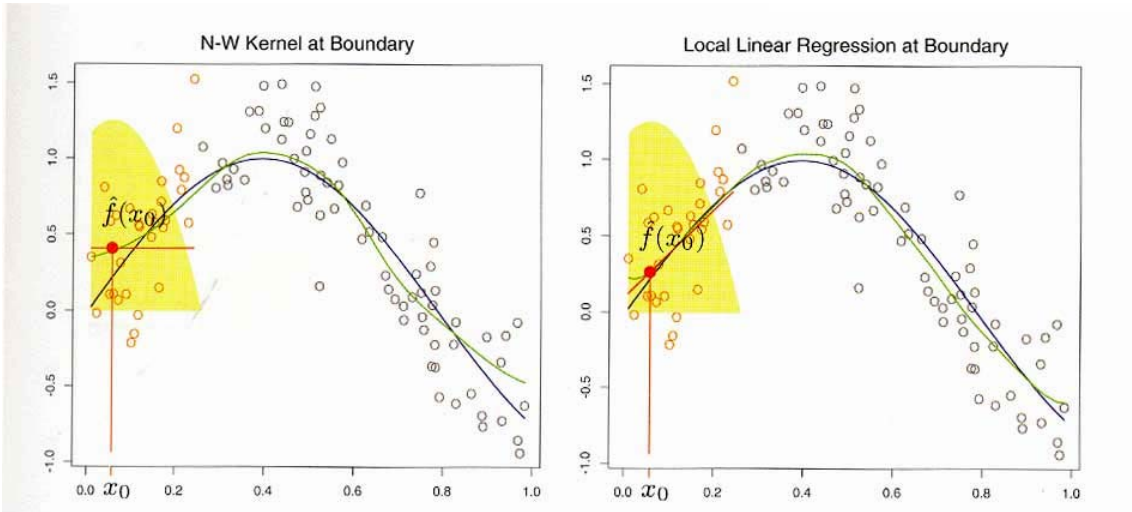
- Gaussian kernel $K_\lambda(x_0,x)=\frac{1}{\sqrt{2\pi}\lambda}\exp\left(-\frac{(x-x_0)^2}{2\lambda^2}\right)$



Kernel Characteristics:
- Compact – vanishes beyond a finite range (such as Epanechnikov, tri-cube)
- Everywhere differentiable (Gaussian, tri-cube)

Local Linear Regression

- In kernel-weighted average method estimated function value has a high bias at the boundary
- This high bias is a result of the asymmetry at the boundary
- The bias can also be present in the interior when the *x* values in the training set are not equally spaced
- Fitting straight lines rather than constants locally helps us to remove bias (why?)



Locally Weighted Linear Regression

- Least squares solution:

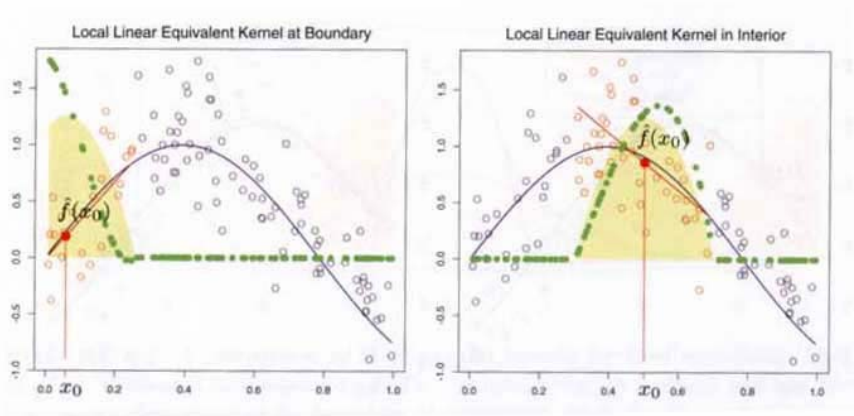
$$\min_{\alpha(x_0),\beta(x_0)} \sum_{i=1}^{N} K_\lambda(x_0, x_i)\left[y_i - \alpha(x_0) - \beta(x_0)x_i\right]^2$$

$$\hat{f}(x_0) = \hat{\alpha}(x_0) + \hat{\beta}(x_0)x_0 = b(x_0)^T \left(B^T W(x_0) B\right)^{-1} B^T W(x_0) y = \sum_{i=1}^{N} l_i(x_0) y_i$$

vector-valued function: $b(x)^T = (1, x)$

$N \times 2$ regression matrix B with ith row $b(x_i)$

$N \times N$ diagonal matrix $W(x_0)$ with ith diagonal element $K_\lambda(x_0, x_i)$



- Note that the estimate is linear in y_i
- The weights $l_i(x_i)$ are sometimes referred to as the equivalent kernel

Bias Reduction In Local Linear Regression

Local linear regression automatically modifies the kernel to correct the bias exactly to the first order

$$E\hat{f}(x_0) = \sum_{i=1}^{N} l_i(x_0) f(x_i)$$

Write a Taylor series expansion of $f(x_i)$

$$= f(x_0)\sum_{i=1}^{N} l_i(x_0) + f'(x_0)\sum_{i=1}^{N}(x_i - x_0) l_i(x_0) + f''(x_0)\sum_{i=1}^{N}(x_i - x_0)^2 l_i(x_0) + R$$

$$= f(x_0) + f''(x_0)\sum_{i=1}^{N}(x_i - x_0)^2 l_i(x_0) + R$$

$$bias = E\hat{f}(x_0) - f(x_0) = f''(x_0)\sum_{i=1}^{N}(x_i - x_0)^2 l_i(x_0) + R$$

since : $\sum_{i=1}^{N} l_i(x_0) = 1$ and $\sum_{i=1}^{N}(x_i - x_0) l_i(x_0) = 0$ Ex. 6.2 in [HTF]

Local Polynomial Regression

- Why have a polynomial for the local fit?

$$\min_{\alpha(x_0),\beta_j(x_0),j=1,\ldots,d} \sum_{i=1}^{N} K_\lambda(x_0,x_i)\left[y_i-\alpha(x_0)-\sum_{j=1}^{d}\beta_j(x_0)x_i^j\right]^2$$

$$\hat{f}(x_0)=\hat{\alpha}(x_0)+\sum_{j=1}^{d}\hat{\beta}_j(x_0)x_0^j=b(x_0)^T\left(B^TW(x_0)B\right)^{-1}B^TW(x_0)y=\sum_{i=1}^{N}l_i(x_0)y_i$$

vector-valued function : $b(x)^T=(1,x,\ldots,x^d)$

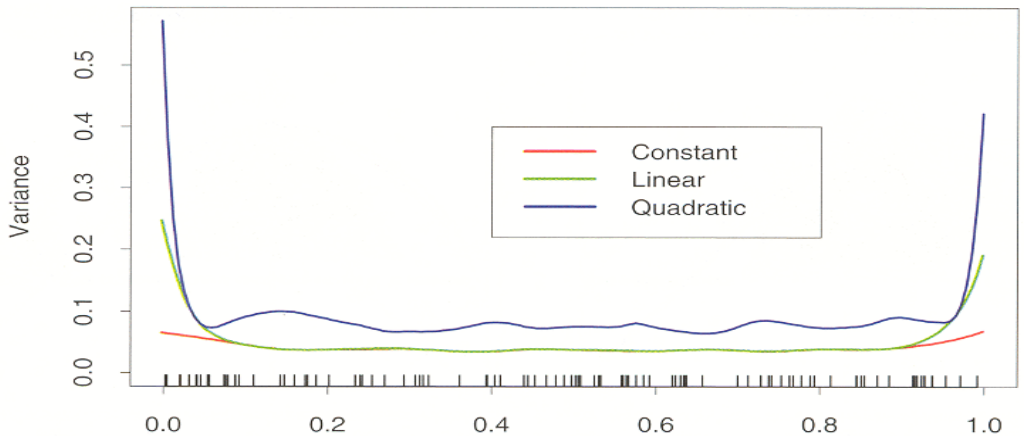
$N\times d+1$ regression matrix B with *ith* row $b(x_i)^T$

$N\times N$ diagonal matrix $W(x_0)$ with *ith* diagonal element $K_\lambda(x_0,x_i)$

- We will gain on bias; however we will pay the price in terms of variance (why?)

Bias and Variance Tradeoff

- As the degree of local polynomial regression increases, bias decreases and variance increases
- Local linear fits can help reduce bias significantly at the boundaries at a modest cost in variance
- Local quadratic fits tend to be most helpful in reducing bias due to curvature in the interior of the domain
- So, would it be helpful have a mixture or linear and quadratic local fits?



Local Regression in Higher Dimensions

- We can extend 1D local regression to higher dimensions

$$\min_{\alpha(x_0),\beta(x_0)} \sum_{i=1}^{N} K_\lambda(x_0, x_i)\left[y_i - b(x_i)^T \beta(x_0)\right]^2$$

$$K_\lambda(x_0, x) = D\left(\frac{\|x - x_0\|}{\lambda}\right)$$

$$\hat{f}(x_0) = b(x_0)^T \hat{\beta}(x_0) = b(x_0)^T \left(B^T W(x_0) B\right)^{-1} B^T W(x_0) y = \sum_{i=1}^{N} l_i(x_0) y_i$$

p dimension with d degree

$1 \times H_{p+1}^{d}$ vector - valued function $: b(x)^T$

$N \times H_{p+1}^{p}$ regression matrix B with ith row $b(x_i)^T$

$N \times N$ diagonal matrix $W(x_0)$ with ith diagonal element $K_\lambda(x_0, x_i)$

- Standardize each coordinates in the kernel, because Euclidean (square) norm is affected by scaling

Local Regression: Issues in High Dimension

- The boundary poses even a greater problem in higher dimensions
  - Many training points are required to reduce the bias; Sample size should increase exponentially in p to match the same performance.
- Local regression becomes less useful when dimensions go beyond 2 or 3
- It's impossible to maintain localness (low bias) and sizeable samples (low variance) in the same time

Combating Dimensions: Structured Kernels

- In high dimensions, input variables (i.e., x variables) could be very much correlated. This correlation could be a key to reduce the dimensionality while performing kernel regression.
- Let *A* be a positive semidefinite matrix (what does that mean?). Let's now consider a kernel that looks like:

$$K_{\lambda,A}(x_0, x) = D\left(\frac{(x - x_0)^T A (x - x_0)}{\lambda}\right)$$

- If $A=\Sigma^{-1}$, the inverse of the covariance matrix of the input variables, then the correlation structure is captured
- Further, one can take only a few principal components of *A* to reduce the dimensionality

Combating Dimension: Low Order Additive Models

- ANOVA (analysis of variance) decomposition:

$$f(x_1, x_2, \ldots, x_p) = \alpha + \sum_{j=1}^{p} g_j(x_j) + \sum_{k<l} g_{kl}(x_k, x_l) + \ldots$$

- One-dimensional local regression is all needed:

$$f(x_1, x_2, \ldots, x_p) = \alpha + \sum_{j=1}^{p} g_j(x_j)$$

Probability Density Function Estimation

- In many classification or regression problems we desperately want to estimate probability densities– recall the instances
- So can we not estimate a probability density, directly given some samples x_i from it?
- Local methods of Density Estimation:

$$f(x_0) = \frac{\#\, x_i \in Nbhood(x_0)}{N\lambda}$$

- This estimate is typically bumpy, non-smooth (why?)

Smooth PDF Estimation Using Kernels

- Parzen method:

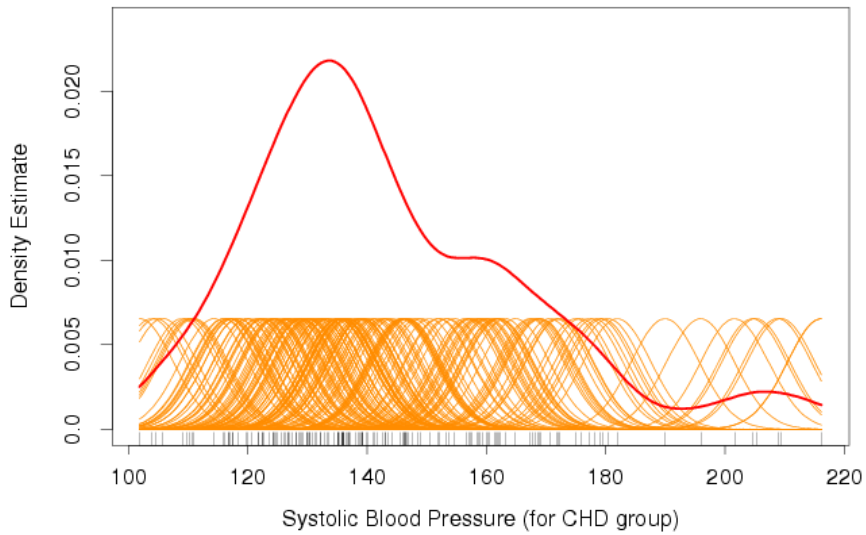
$$\hat{f}(x_0) = \frac{1}{N\lambda}\sum_{i=1}^{N} K_\lambda(x_0, x_i)$$

- Gaussian kernel:

$$K_\lambda(x_0, x_i) = \frac{1}{\sqrt{2\pi}\lambda}\exp\left(-\frac{(x_0 - x_i)^2}{2\lambda^2}\right)$$

- In p-dimensions

$$f_X(x_0) = \frac{1}{N(2\lambda^2\pi)^{\frac{p}{2}}}\sum_{i=1}^{N} e^{-\frac{1}{2}(\|x_i - x_0\|/\lambda)}$$



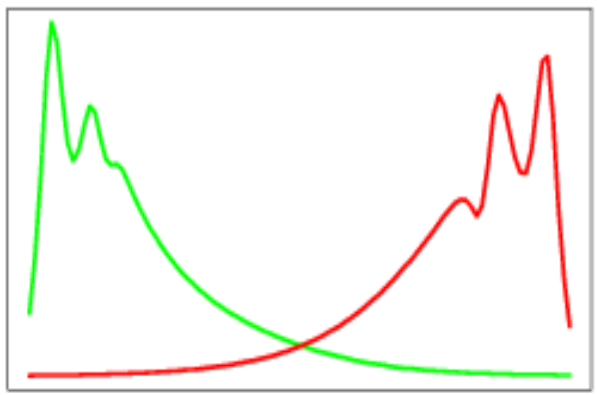
Kernel density estimation

Using Density Estimates in Classification

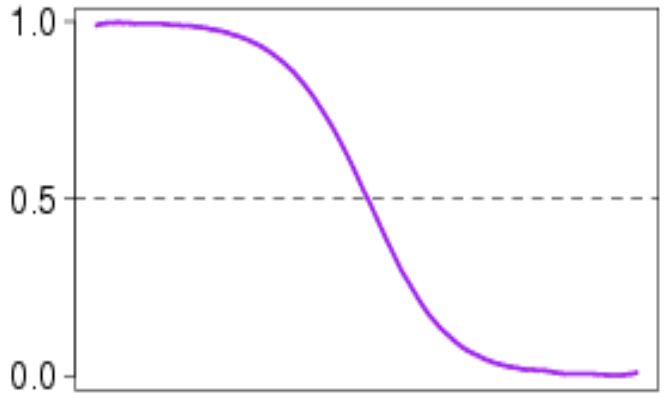
- Posterior probability density: $$P(G=j \mid X=x_0)=\frac{\hat{\pi}_j f_j(x_0)}{\sum_{l=1}^{K}\hat{\pi}_l f_l(x_0)}$$

- In order to estimate this density, we can estimate the class conditional densities using Parzen method

where $f_j(x)=p(x \mid G=j)$ is the j^{th} class conditional density



Class conditional densities



Ratio of posteriors $\frac{P(G=1 \mid X=x)}{P(G=1 \mid X=x)}=\frac{\hat{\pi}_1 f_1(x)}{\hat{\pi}_2 f_2(x)}$

Naive Bayes Classifier

- In Bayesian Classification we need to estimate the class conditional densities: $f_j(x) = p(x \mid G = j)$
- What if x is multi-dimensional?
- If we apply kernel density estimates, we will run into the same high dimension problems
- To avoid these difficulties, assume that the class conditional density factorizes: $f_j(x_1, \ldots, x_p) = \prod_{i=1}^{p} p(x_i \mid G = j)$
- In other words we are assuming that the features are independent – Naïve Bayes model
- Advantages:
  - Each class density for each feature can be estimated (low variance)
  - If some of the features are continuous, some are discrete this method can seamlessly handle the situation
- Naïve Bayes classifier works surprisingly well for many problems

$$\begin{aligned}
\text{logit}\frac{P(G=\ell|X)}{P(G=J|X)} &= \log\frac{\pi_\ell f_\ell(X)}{\pi_J f_J(X)} \\
&= \log\frac{\pi_\ell \prod_{k=1}^{P} f_{\ell k}(X_k)}{\pi_J \prod_{k=1}^{P} f_{Jk}(X_k)} \\
&= \log\frac{\pi_\ell}{\pi_J} + \sum_{k=1}^{P} \log\frac{f_{\ell k}(X_k)}{f_{Jk}(X_k)} \\
&= \beta_{0\ell} + \sum_{k=1}^{J} g_k(X_k)
\end{aligned}$$

Discriminant function is now generalized linear additive

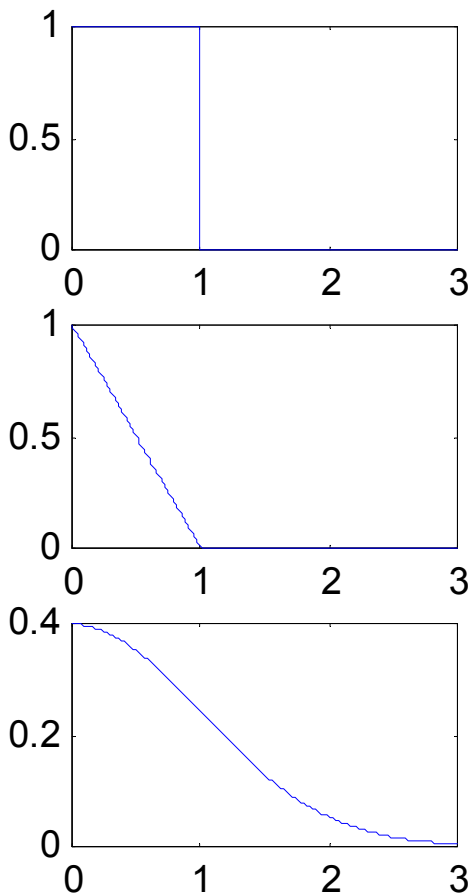
Radial Basis Function (RBF)

- RBF is a *kernel* function that is symmetric w. r. t. origin. Hence its variable is *r* that is the norm-distance from origin.
- Examples of RBF

$$\varphi_1(r) = \begin{cases} 1 & 0 \le r \le c; \\ 0 & r > c. \end{cases}$$

$$\varphi_2(r) = \begin{cases} 1 - r/c & 0 \le r \le c; \\ 0 & r > c. \end{cases}$$

$$\varphi_3(r) = \frac{1}{\sqrt{2\pi\sigma}} e^{-r^2/(2\sigma^2)}$$



Interpolation Problem Formulation

Radial Basis function for interpolation:

Given $\{\boldsymbol{x}_i;\ 1 \le i \le K\}$ and $\{d_i;\ 1 \le i \le K\}$, find a function $F(\boldsymbol{x})$ that satisfies the interpolation condition:

$$F(\boldsymbol{x}_i) = d_i \qquad 1 \le i \le K \quad (1)$$

One possible choice of $F(\boldsymbol{x})$ is a radial basis function of the following form:

$$F(x) = \sum_{i=1}^{K} w_i \varphi(\| x - x_i \|) \quad (2)$$

where $\{\boldsymbol{x}_i;\ 1 \le i \le K\}$ are the centers of the radial basis functions.

Solving Radial Basis Coefficients

- Substitute (1) into (2), we obtain a set of linear system of equations: $M\,w = d$ (3)

  where $M = [M(i,j),\ 1 \le i, j, \le K]$ is the *interpolation matrix*, $M(i,j) = \varphi(||\boldsymbol{x}_i - \boldsymbol{x}_j||)$, $w = [w_1, w_2, \bullet\bullet\bullet, w_K]^T$, and $d = [d_1, d_2, \bullet\bullet\bullet, d_K]^T$
- Given M and $\boldsymbol{d}$, assuming the N centers are distinct, w can be solved as: $w = M^{-1}d$ if M is non-singular. If the $\varphi(r) = (r^2 + c^2)^{-1/2}$, or $\varphi(r) = exp(-r^2/(2s^2))$, it can further be shown that M is also positive definite

An Example

Let F(–1) = 0.2, F(–0.5) = 0.5, and F(1) = –0.5.

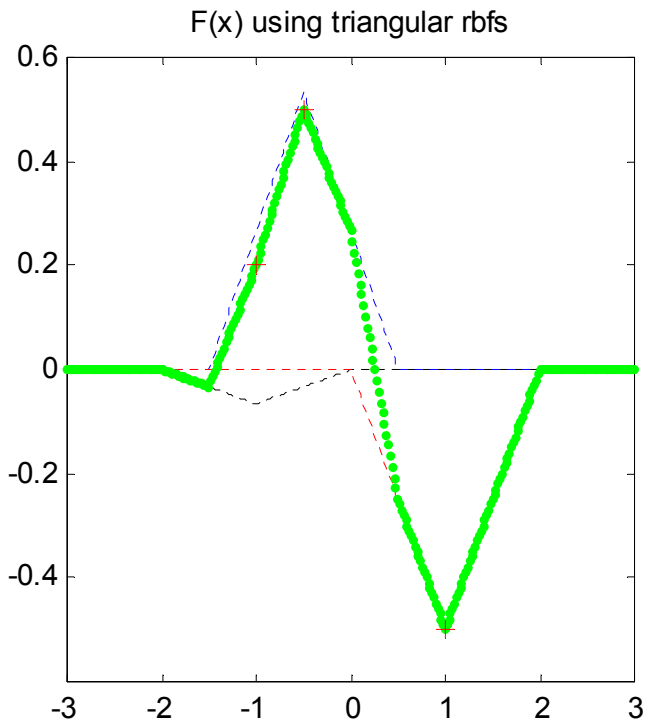
Use a triangular radial basis function j(r) = (1–r)[u(r) –u(r –1)]

u(r) = 1 if r ≥ 0 and = 0 if r < 0.

$$F(x) = w_1\varphi(|x+1|) + w_2\varphi(|x+0.5|) + w_3\varphi(|x-1|)$$

$$\underbrace{\begin{bmatrix} 1 & .5 & 0 \\ .5 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{M}} \underbrace{\begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}}_{\mathbf{w}} = \underbrace{\begin{bmatrix} 0.2 \\ 0.5 \\ -0.5 \end{bmatrix}}_{\mathbf{d}}$$

Solve for $\mathbf{w} = \begin{bmatrix} \frac{-1}{15} & \frac{8}{15} & \frac{-1}{2} \end{bmatrix}$



rbfexample1.m

Example Continued

- Use Gaussian RBFs:

$$\varphi(\| x - x_i \|) = \frac{1}{\sqrt{2\pi}} \exp\left(-|x - x_i|^2 / 2\right)$$

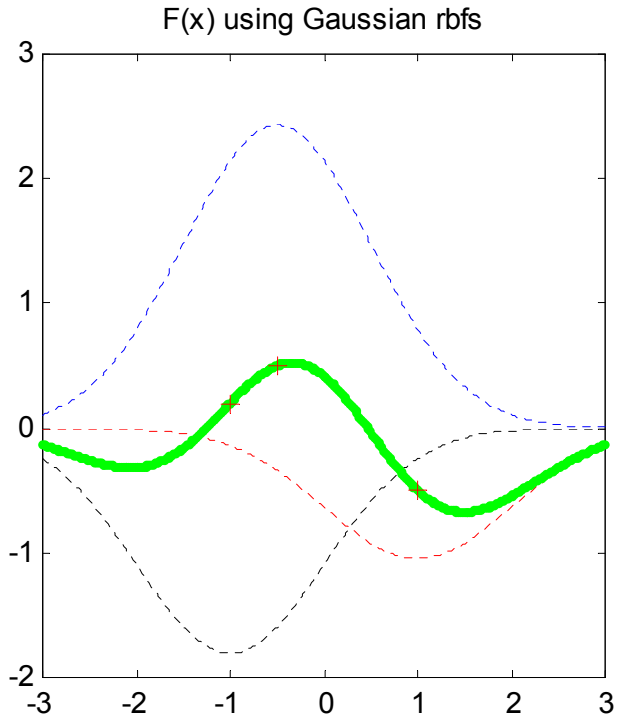
$$\frac{1}{\sqrt{2\pi}} \begin{bmatrix} 1 & e^{-|-1+0.5|^2/2} & e^{-|-1-1|^2/2} \\ e^{-|-0.5+1|^2/2} & 1 & e^{-|-0.5-1|^2/2} \\ e^{-|1+1|^2/2} & e^{-|1+0.5|^2/2} & 1 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} 0.2 \\ 0.5 \\ -0.5 \end{bmatrix}$$

$$w = \begin{bmatrix} -1.8051 & 2.4323 & -1.0454 \end{bmatrix}$$

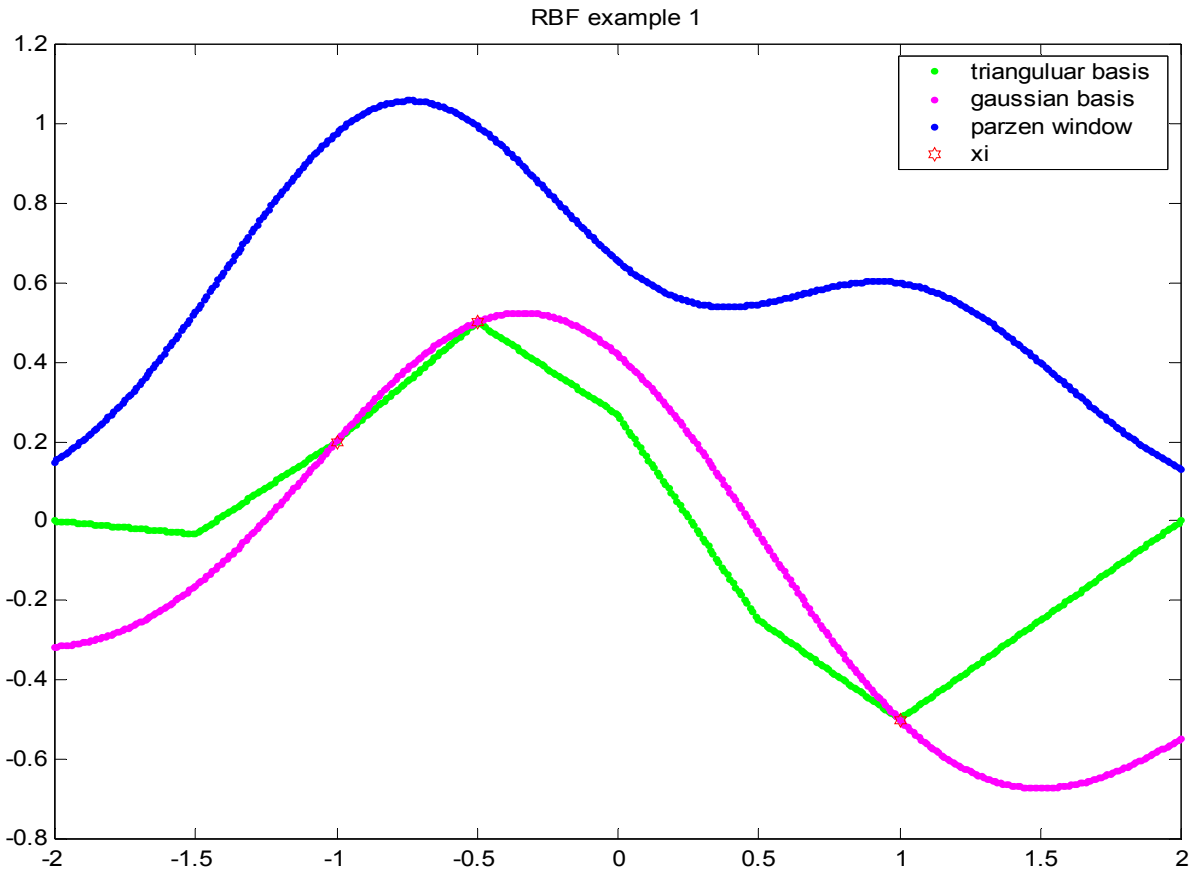
- Parzen window: No weighting, and no target values of F(x) needed

$$P(x) = \frac{1}{3h_3} \sum_{i=1}^{3} \varphi\left(\frac{x - x_i}{h_3}\right) \qquad h_3 = \frac{1}{\sqrt{3}}$$

$$= \frac{1}{\sqrt{3}} \begin{bmatrix} e^{-3|x+1|^2/2} & e^{-3|x+0.5|^2/2} & e^{-3|x-1|^2/2} \end{bmatrix}$$



Example (Comparison)



Radial Basis Network (Type II)

$$F(x) = \sum_{j=1}^{J} w_j G(x; t_j)$$

Instead of x_i, use virtual data points t_j in the solution of $F(x)$. Define

$$\mathbf{G} = \begin{bmatrix} G(x_1,t_1) & G(x_1,t_2) & \cdots & G(x_1,t_J) \\ G(x_2,t_1) & G(x_2,t_2) & \cdots & G(x_2,t_J) \\ \vdots & \vdots & \ddots & \vdots \\ G(x_K,t_1) & G(x_K,t_2) & \cdots & G(x_K,t_J) \end{bmatrix}_{K\times J}$$

$$\mathbf{G}_0 = \begin{bmatrix} G(t_1,t_1) & G(t_1,t_2) & \cdots & G(t_1,t_J) \\ G(t_2,t_1) & G(t_2,t_2) & \cdots & G(t_2,t_J) \\ \vdots & \vdots & \ddots & \vdots \\ G(t_J,t_1) & G(t_J,t_2) & \cdots & G(t_J,t_J) \end{bmatrix}_{J\times J}$$

Substitute each x_i into eq.

$$F(x_i)=d_i$$

we have a new system:

$$(G^TG + IG_o)w = G^Td$$

Thus,

$$w = (G^TG + IG_o)^{-1}G^Td$$

when I = 0,

$$w = G^+d = (G^TG)^{-1}G^Td$$

where G^+ is the pseudo-inverse matrix of G

RBN2 Algorithm Summary

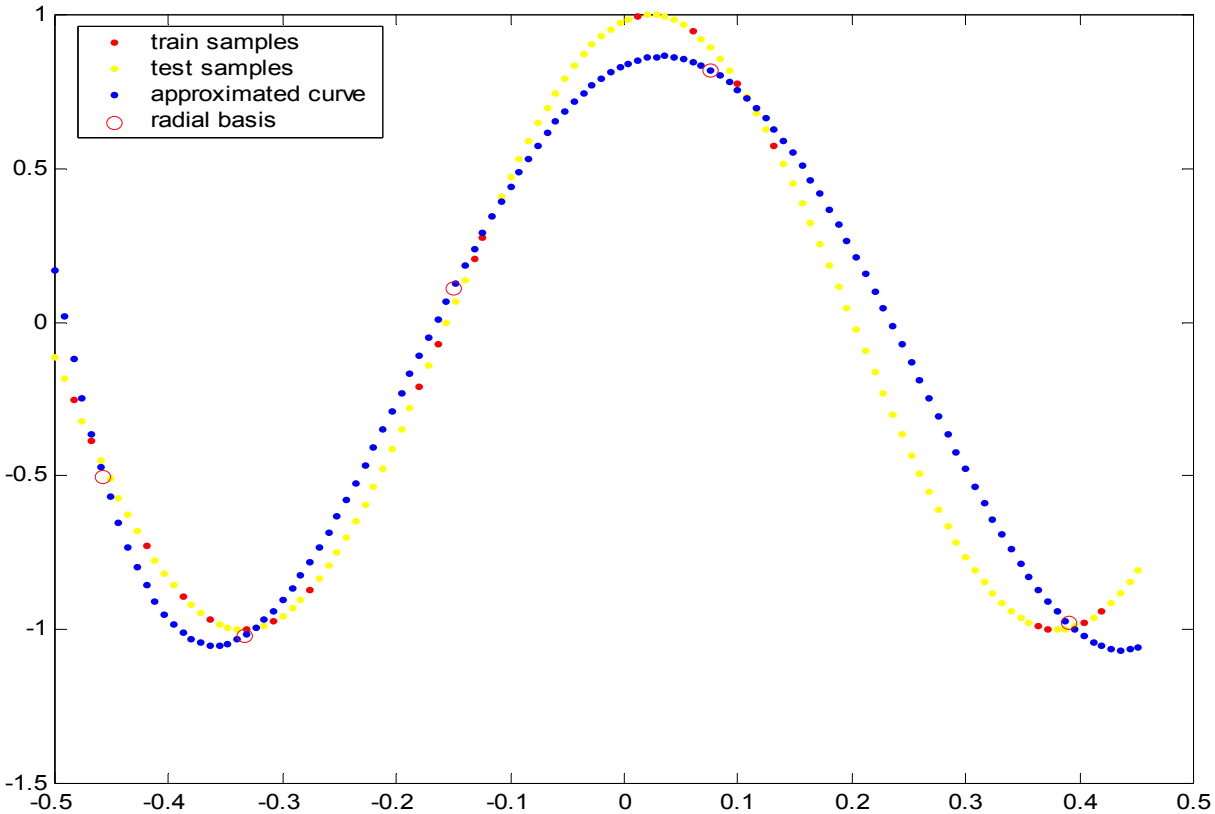
- Given: $\{x_i;\ 1 \le i \le K\}$, d: desired output, J: # of radial neurons
- Cluster $\{x_i\}$ into J clusters, find centers $\{t_j;1 \le\ j \le J\}$, variance σ_j^2 or inverse covariance matrix Σ_j^{-1} are also computed
- Compute G matrix (K by J) and G_0 matrix

  $G_{i,j+1} = exp(-0.5||x(i)-t_j||^2/\sigma_j^2)$ or

  $G_{i,j+1} = exp(-0.5(x(i)-t_j)^T\Sigma_j^{-1}(x(i)-t_j))$

  Solve $w = G^{\dagger}d$ or $(G^TG + IG_0)^{-1}G^Td$
- Above procedure can be refined by fitting the clusters into a Gaussian mixture model and train it with the EM algorithm

Example


train samples
test samples
approximated curve
radial basis
1
0.5
0
-0.5
-1
-1.5
-0.5
-0.4
-0.3
-0.2
-0.1
0
0.1
0.2
0.3
0.4
0.5

General RBF Network

- Consider a Gaussian RBF model

$$F(x) = \sum_{j=1}^{J} w_j \exp\left(-\frac{(x - t_j)^2}{2\sigma^2} \right)$$

- In RBN-II training, in order to compute $\{w_i\}$, parameters $\{t_j\}$ are determined in advance using *K*-means clustering *and* s^2 is selected initially
- To fit the model better at $F(x_i) = d_i$, these parameters may need fine-tuning

- Additional enhancements include
  - Allowing each basis has its own width parameter s_j, and
  - A bias term is added to compensate for nonzero background value of the function over the support

$$F(x) = \sum_{j=1}^{J} w_j \exp\left(-\frac{(x - t_j)^2}{2\sigma_j^2} \right) + b$$

- While similar to the Gaussian mixture model, $\{w_i\}$ can be negative is the main difference

Training of Generalized RBN

$$F(x)=\sum_{j=1}^{J} w_j \exp\left(-\frac{(x-t_j)^2}{2\sigma_j^2}\right)+b$$

Parameters q = {w_j, t_j, s_j, b} are to minimize the approximation error

$$E(\theta)=\sum_{i=1}^{K}\frac{e_i^2}{2}=\frac{1}{2}\sum_{i=1}^{K}[F(x_i\,|\,\theta)-d_i]^2$$

Specifically, for $1 \le m \le J$

$$\frac{\partial F(x_i\,|\,\theta)}{\partial t_m}=\frac{w_m(x_i-t_m)}{\sigma_m^2}\cdot\exp\left(-\frac{(x_i-t_m)^2}{2\sigma_m^2}\right)$$

$$\frac{\partial F(x_i\,|\,\theta)}{\partial \sigma_m^2}=\frac{w_m(x_i-t_m)^2}{2\sigma_m^4}\cdot\exp\left(-\frac{(x_i-t_m)^2}{2\sigma_m^2}\right)$$

$$\frac{\partial F(x_i\,|\,\theta)}{\partial w_m}=\exp\left(-\frac{(x_i-t_m)^2}{2\sigma_m^2}\right),\quad \text{and}$$

$$\frac{\partial F(x_i\,|\,\theta)}{\partial b}=1$$

The steepest descent gradient method leads to:

$$\theta(n+1)=\theta(n)-\eta\nabla_\theta E(\theta)=\theta(n)-\eta\sum_{i=1}^{K} e_i\cdot\nabla_\theta F(x_i\,|\,\theta)$$

Training …

Note that

$$G_{ij} = \exp\left(-\frac{|x_i - t_j|^2}{2\sigma_j^2}\right)$$

Hence

$$\frac{\partial E(\theta)}{\partial t_m} = \sum_{i=1}^{K}\left(e_i G_{im} \cdot \frac{w_m (x_i - t_m)}{\sigma_m^2}\right)$$

$$\frac{\partial E(\theta)}{\partial \sigma_m^2} = \sum_{i=1}^{K}\left(e_i G_{im} \cdot \frac{w_m (x_i - t_m)^2}{2\sigma_m^4}\right)$$

$$\frac{\partial E(\theta)}{\partial w_m} = \sum_{i=1}^{K} e_i G_{im}, \quad \text{and}$$

$$\frac{\partial E(\theta)}{\partial b} = \sum_{i=1}^{K} e_i$$

Thus, the individual parameters' on-line learning formula are:

$$e_i = \sum_{j=1}^{J} w_j G_{ij} - d_i$$

$$t_m(n+1) = t_m(n) - \eta \cdot \sum_{i=1}^{K}\left(e_i G_{im} \cdot \frac{w_m (x_i - t_m)}{\sigma_m^2}\right)$$

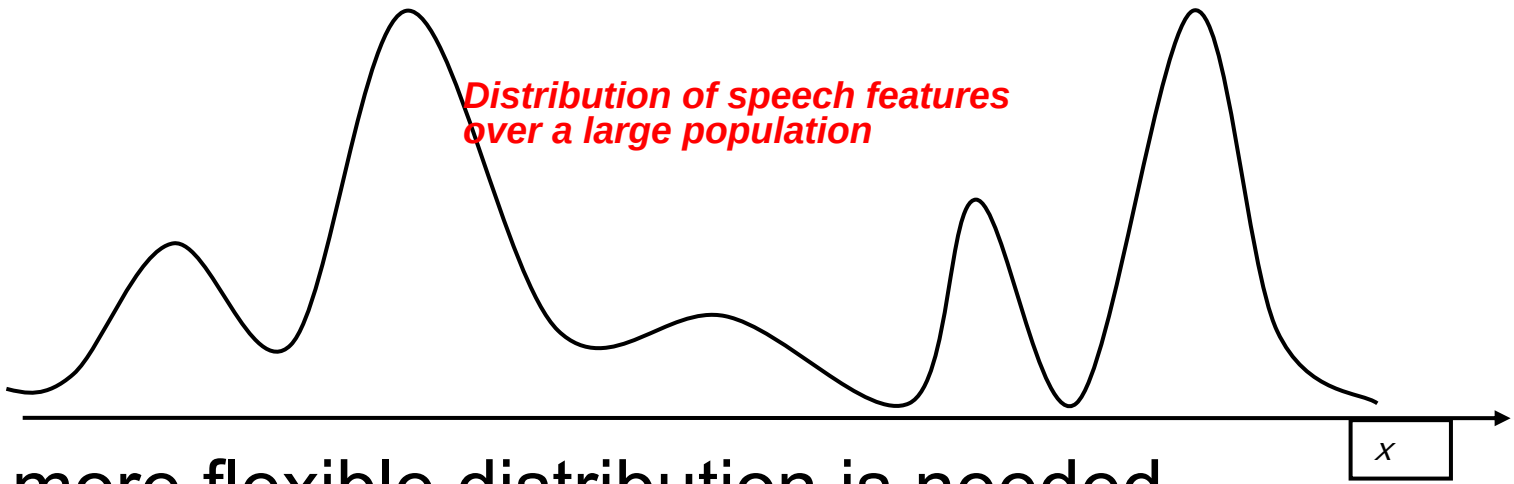
$$\sigma_m^2(n+1) = \sigma_m^2(n) - \eta \cdot \sum_{i=1}^{K}\left(e_i G_{im} \cdot \frac{w_m (x_i - t_m)^2}{2\sigma_m^4}\right)$$

$$w_m(n+1) = w_m(n) - \eta \cdot \sum_{i=1}^{K} e_i G_{im}, \quad \text{and}$$

$$b(n+1) = b(n) - \eta \cdot \sum_{i=1}^{K} e_i$$

Distribution of Real-World Data

- Single Gaussian distribution (either univariate or multivariate) is a single mode distribution.
- In many cases, the distribution has multiple modes



- A more flexible distribution is needed
  - Gaussian Mixture model (GMM)
  - A GMM can be tuned to approximate any distribution

Gaussian Mixture Model (GMM)

- Gaussian mixture model (GMM)
  - Multivariate density: $p(X) = \sum_{k=1}^{K} \omega_k \cdot N(X \mid \mu_k, \Sigma_k)$
  - GMM is a mixture of single Gaussian distribution (each one is called mixand) which have different means and variances
  - ω_k is called mixture weight, prior probability of each mixand. They satisfy $\sum_{k=1}^{K} \omega_k = 1$
- GMM is widely used for speaker recognition, audio classification, audio segmentation, etc.

Estimation of GMM Parameters

- However, GMM parameter estimation is not trivial
- Consider a simple case:
  - have a set of training data X=$\{x_1, x_2, \ldots, x_n\}$
  - use a 2-mixture GMM to model it:

$$p(x) = \frac{0.5}{\sqrt{2\sigma_1^2}} e^{-\frac{(x-\mu_1)^2}{2\sigma_1^2}} + \frac{0.5}{\sqrt{2\sigma_2^2}} e^{-\frac{(x-\mu_2)^2}{2\sigma_2^2}}$$

  - obtain ML estimate of μ_1, σ_1, μ_2, σ_2 from training data.
  - maximization with differential calculus does not work
- Alternative – expectation maximization (EM)

Expectation-Maximization (EM) Algorithm

- EM algorithm is an iterative method of obtaining a maximum likelihood estimate of model parameters for GMM
- EM suits best to the so-called *missing data* problem:
  - Only observe a subset of features, called *observed*, *X*
  - Other features are *missing* or unobserved, denoted as *Y*
  - The complete data *Z={X,Y}.* If given the complete data *Z*, it is usually easy to obtain ML estimation of model parameters
  - How to do ML estimation based on observed *X* only??
- EM consists of two steps: Iterate until convergence
  - Expectation (E-step): the expectation is with respect of the missing data *Y*, using the current estimate of the unknown parameters and conditioned upon the observed *X*
  - Maximization (M-step): provides a new estimate of unknown parameters (better than the initial ones) in terms of increasing the above expectation → increasing likelihood function of observed

EM Algorithm (I): E-step

- E-step: form an auxiliary function

$$Q(\theta;\theta^{(i)}) = E_Y\left[\ln\ p(X,Y\mid\theta)\ \mid\ X,\theta^{(i)}\ \right]$$

  - The expectation of log-likelihood function of complete data is calculated based on the current estimate of unknown parameter, conditioned on the observed data.
  - $Q(\theta;\theta^{(i)})$ is a function of θ with $\theta^{(i)}$ assumed fixed.
  - If missing data Y is continuous:

$$Q(\theta;\theta^{(i)}) = \int_{\Lambda_Y} \ln\ p(X,Y\mid\theta)\cdot p(Y\mid X,\theta^{(i)})\,\mathrm{d}Y$$

  - If missing data Y is discrete:

$$Q(\theta;\theta^{(i)}) = \sum_{Y} \ln\ p(X,Y\mid\theta)\cdot p(Y\mid X,\theta^{(i)})$$

EM Algorithm (II): M-step

- M-step: choose a new estimate $\theta^{(i+1)}$ which maximizes $Q(\theta;\theta^{(i)})$

$$\theta^{(i+1)} = \arg\max_{\theta} Q(\theta;\theta^{(i)})$$

  - $\theta^{(i+1)}$ increases likelihood value $p(X|\theta)$ than $\theta^{(i)}$.
- Replace $\theta^{(i)}$ with $\theta^{(i+1)}$ and iterate until convergence.
- EM algorithm guarantees that the log-likelihood of the observed data $p(X|\theta)$ will increase monotonically.
- EM algorithm may converge to a local maximum or global maximum. And convergence rate is reasonably good.
- Applications of the EM algorithm:
  - ML estimation of some complicated models, e.g., GMM, HMM, …
  - Active noise cancellation (ANC)
  - Spread-spectrum multi-user communication

ML Estimation of Multivariate GMM (I)

- Assume a data set $D=\{X_1,X_2,\ldots,X_T\}$ (a set of vectors)
- Want to model the data by using multivariate GMM:

$$p(X)=\sum_{k=1}^{K}\omega_k\cdot N(X\mid\mu_k,\Sigma_k) \qquad (\text{with}\sum_{k=1}^{K}\omega_k=1)$$

- Problem: use data set D to estimate GMM model parameters, including $\omega(k)$, $\mu(k)$, $\Sigma(k)$ $(k=1,2,\ldots,K)$.
- If we know the label of mixand *lt* from which each data *Xt* come from, the estimation is extremely easy.
- Since the mixand label is not available in training set, we treat it as missing data:
  - Observed data: $D=\{X_1,X_2,\ldots,X_T\}$. Missing data: $L=\{l_1, l_2, \ldots, l_T\}$.
  - *Complete data:* $\{D,L\}=\{X_1, l_1, X_2, l_2, \ldots, X_T, l_T\}$

MLE of Multivariate GMM (II) : E-Step

$$Q(\{\omega_k,\mu_k,\Sigma_k\}\,|\,\{\omega_k^{(i)},\mu_k^{(i)},\Sigma_k^{(i)}\}) = \sum_L \ln p(D,L\,|\{\omega_k,\mu_k,\Sigma_k\})\cdot p(L\,|\,D,\{\omega_k^{(i)},\mu_k^{(i)},\Sigma_k^{(i)}\})$$

$$= C + \sum_L \left[\sum_{t=1}^{T}[\ln\omega_{l_t} - \frac{n}{2}\ln|\Sigma_{l_t}| - \frac{1}{2}\cdot(X_t-\mu_{l_t})^t\Sigma_{l_t}^{-1}(X_t-\mu_{l_t})]\right]\cdot\prod_{t=1}^{t} p(l_t\,|\,X_t,\{\omega_k^{(i)},\mu_k^{(i)},\Sigma_k^{(i)}\})$$

$$= C + \sum_{t=1}^{T}\sum_{k=1}^{K}[\ln\omega_k - \frac{n}{2}\ln|\Sigma_k| - \frac{1}{2}\cdot(X_t-\mu_k)^t\Sigma_k^{-1}(X_t-\mu_k)]\cdot p(l_t=k\,|\,X_t,\omega_k^{(i)},\mu_k^{(i)},\Sigma_k^{(i)})$$

- The partition function (occupation probability):

$$p(l_t=k\,|\,X_t,\{\omega_k^{(i)},\mu_k^{(i)},\Sigma_k^{(i)}\}) = \frac{\omega_k^{(i)}\cdot N(X_t\,|\,\mu_k^{(i)},\Sigma_k^{(i)})}{\sum_{k=1}^{K}\omega_k^{(i)}\cdot N(X_t\,|\,\mu_k^{(i)},\Sigma_k^{(i)})}$$

MLE of Multivariate GMM (III): M-Step

$$\frac{\partial Q}{\partial \mu_k} = 0 \Rightarrow \mu_k^{(i+1)} = \frac{\sum_{t=1}^{T} X_t \cdot p(l_t = k \mid X_t, \omega_k^{(i)}, \mu_k^{(i)}, \Sigma_k^{(i)})}{\sum_{t=1}^{T} p(l_t = k \mid X_t, \omega_k^{(i)}, \mu_k^{(i)}, \Sigma_k^{(i)})}$$

$$\frac{\partial Q}{\partial \Sigma_k} = 0 \Rightarrow \Sigma_k^{(i+1)} = \frac{\sum_{t=1}^{T} (X_t - \mu_k^{(i)})^t \cdot (X_t - \mu_k^{(i)}) \cdot p(l_t = k \mid X_t, \omega_k^{(i)}, \mu_k^{(i)}, \Sigma_k^{(i)})}{\sum_{t=1}^{T} p(l_t = k \mid X_t, \omega_k^{(i)}, \mu_k^{(i)}, \Sigma_k^{(i)})}$$

$$\frac{\partial}{\partial \omega_k}[Q - \lambda(\sum_{k=1}^{K} \omega_k - 1)] = 0 \Rightarrow \omega_k = \frac{\sum_{t=1}^{T} p(l_t = k \mid X_t, \omega_k^{(i)}, \mu_k^{(i)}, \Sigma_k^{(i)})}{\sum_{k=1}^{K}\sum_{t=1}^{T} p(l_t = k \mid X_t, \omega_k^{(i)}, \mu_k^{(i)}, \Sigma_k^{(i)})} = \frac{\sum_{t=1}^{T} p(l_t = k \mid X_t, \omega_k^{(i)}, \mu_k^{(i)}, \Sigma_k^{(i)})}{T}$$

MLE of Multivariate GMM (IV): Iteration

- Iterative ML estimation of GMM
  - Initiation: choose $\{\omega_k^{(0)}, \mu_k^{(0)}, \sum_k^{(0)}\}$. Usually use vector clustering algorithm (such as LBG) to cluster all data into *K* clusters. Each cluster is used to train for one Gaussian mixand
  - *i=0*;
  - Use EM algorithm to refine model estimation

$$\{\omega_k^{(i)}, \mu_k^{(i)}, \sum_k^{(i)}\} \Rightarrow \{\omega_k^{(i+1)}, \mu_k^{(i+1)}, \sum_k^{(i+1)}\}$$

  - *i*++, go back until convergence

Applications of GMM

- GMM is widely used to model speech or audio signals. In many cases, we use diagonal covariance matrices in GMM
- Speaker recognition:
  - Collect some speech signals from all known speakers
  - Train a GMM for each known speaker by using his/her voice
  - Match with all trained GMM's and determine the speaker's identity
- Audio classification:
  - Classify a continuous audio/video stream (from radio or TV) into some homogeneous segments: anchor's speech, in-field interview, telephone interview, music, commercial ads, sports, etc.
  - Train a GMM based on training data for each audio class
  - Use trained GMM's to scan an unknown audio stream to segment it

Example: Speaker Verification (I)

- Speaker verification: User first claims a user ID, the system records some voice sample and try to answer YES/NO to the question "Is the person the claimed user or not?"
- Speaker verification: if a person claims to be the user *A*,
  - Observation: a segment of voice ➔ feature vectors *X*

  H_0: *X is from the claimed user A*

  H_1: *X is NOT from the claimed user A*
- Data modeling: commonly use GMM for both H_0 and H_1
  - Mixture number depends on the amount of available data, usually from *16* to *256*

  For simplicity, each Gaussian is assumed to be diagonal.

  For each known user *a* registered in the system, we must estimate two GMM's Λ_a and Λ_a for its H_0 and H_1

Example: Speaker Verification (II)

- Model estimation:
  - For Λ_a in *H0*: collect some training samples from the known user and train it based on the ML criterion
  - How to perform ML estimation for GMM?
  - How about $\overline{\Lambda}_a$ in *H1*?
    - Anti-speaker model: Train it based on training data collected for all other known users (except *a*). (ML estimation)
    - Training it based on training data from some "cohort" speakers who are confusing with the current speaker *a.* (how to choose cohort speaker?)
    - For simplicity, use the same background model $\overline{\Lambda}$ for all known users. $\overline{\Lambda}$ is trained based on all users' training data.

Summary

- Today's Class
  - Kernel Methods (Chapter 6)
- Next Classes
  - Basis Expansion (Chapter 5)
- Exercises: make sure you know the topics discussed and how to do all the exercises suggested in Chapters 6 &12
- Reading Assignments
  - HTF, Chapters 6 & 5