Advanced Machine Learning & Perception

Instructor: Tony Jebara

Topic 6

- Standard Kernels
- Unusual Input Spaces for Kernels
- String Kernels
- Probabilistic Kernels
- Fisher Kernels
- Probability Product Kernels
- Mixture Model Probability Product Kernels
- Junction Tree Algorithm for Kernels

SVMs and Kernels

- Recall the SVM dual optimization:

$$L_D : \max \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j k\left(x_i, x_j\right)$$
$$subject\ to\ \ \alpha_i \in \left[0, C\right]\ \ and\ \sum_i y_i \alpha_i = 0$$

- Solve a QP by using the Gram matrix K where $K_{ij} = k\left(x_i, x_j\right)$

$$K = \begin{bmatrix} k\left(x_1, x_1\right) & k\left(x_1, x_2\right) & k\left(x_1, x_3\right) \\ k\left(x_1, x_2\right) & k\left(x_2, x_2\right) & k\left(x_2, x_3\right) \\ k\left(x_1, x_3\right) & k\left(x_2, x_3\right) & k\left(x_3, x_3\right) \end{bmatrix}$$

- Then KKT conditions give us b. The SVM prediction is then:

$$f\left(x\right) = sign\left(\sum_i \alpha_i y_i k\left(x, x_i\right) + b\right)$$

- Kernels replace vector dot products for nonlinear boundary
- Kernels also allow *arbitrary large structured input spaces!*

Standard Kernels on Vectors

- Kernels usually take two input vectors and output a scalar:

- Polynomial Kernel (gives p twists in SVM decision boundary)

$$k\left(x_i, x_j\right) = \left\langle \phi\left(x_i\right), \phi\left(x_j\right) \right\rangle = \left(x_i^T x_j + 1\right)^p$$

- RBF Kernel (phi is infinite dimensional)

$$k\left(x_i, x_j\right) = \left\langle \phi\left(x_i\right), \phi\left(x_j\right) \right\rangle = \exp\left(-1/2\sigma^2 \left\| x_i - x_j \right\|^2 \right)$$

- Hyperbolic Tangent Kernel

$$k\left(x_i, x_j\right) = \left\langle \phi\left(x_i\right), \phi\left(x_j\right) \right\rangle = \tanh\left(\kappa x_i^T x_j - \delta\right)$$

- But why just deal with vector inputs?
- We can apply kernels between any type of input:
  graphs, strings, probabilities, structured spaces, etc.!

String Kernels

- Inputs are two strings: X = "aardvark"

  X' = "accent"

- Want $k(X,X') = \text{scalar value} = \phi(X)^T\phi(X')$

- One choice for features $\phi(X)$ is the number of times each substrings of length 1, 2 and 3 appears in X

$\phi(X)$ = [#a #b #c #d #e #f … #y #z #aa #ab #ac …]

= [3 0 0 1 0 0 … 0 0 1 0 0 …]

$\phi(X')$ = [1 0 2 0 1 0 … 0 0 0 0 1 …]

- Can do this efficiently via dynamic programming

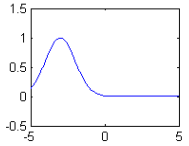
String Kernels

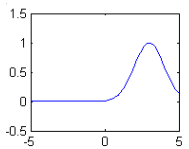
- Want $k(X,X') = \text{scalar value} = \phi(X)^T\phi(X')$
- Explicit Kernel can be slow because of many substrings s in our final vocabulary
- $\phi_s(X)$ = number of times substring s appears in X
- $\phi_a(\text{aardvark}) = 3$ $\phi_{ar}(\text{aardvark}) = 2$
- Implicit Kernel is more efficient

$k(X,X')$: for each substring s of X, count how often s appears in X'

…via dynamic programming in time proportional to the product of the lengths of X and X'. This computes the dot product much more quickly!

Probability Kernels

- Inputs are two probabilities: $p = p(x|\theta)$ = 

  $p' = p(x|\theta')$ = 

- Want k(p,p') = scalar value
- How to design this?
- Recall that a kernel can be converted into a distance:

$$D\left(p_i, p_j\right) = \sqrt{k\left(p_i, p_i\right) - 2k\left(p_i, p_j\right) + k\left(p_j, p_j\right)}$$

- What is a natural distance between two probabilities?
- Kullback-Leibler Divergence!

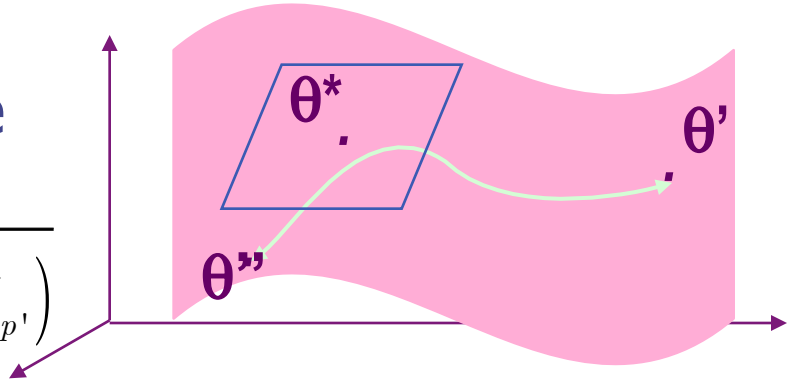
$$D\left(p_i \| p_j\right) = \int p_i\left(x\right) \log \frac{p_i\left(x\right)}{p_j\left(x\right)} dx$$

...caveats?

Fisher Kernels

- KL isn't symmetric distance, probabilities lie on a manifold!
- So we must approximate it…
- Try quadratic distance measure like Mahalanobis distance

$$D\left(p''\middle\|p'\right) \approx \sqrt{\left(U_{p''} - U_{p'}\right)^T I_{\theta^*}^{-1}\left(U_{p''} - U_{p'}\right)}$$



- Linearize manifold at θ^* (some maximum likelihood point)
- Get distance between p and p' via a distance from θ to θ'
- The right kernel to go with the Mahalanobis distance is:

$$k\left(p'', p'\right) = U_{p''} I_{\theta^*}^{-1} U_{p'}$$

$$where\ U_{p''} = \nabla_{\theta''} \log p\left(x \mid \theta\right)\ \ and\ \ U_{p'} = \nabla_{\theta'} \log p\left(x \mid \theta\right)$$

- Matrix I is the Fisher information

$$I_{\theta^*}\left(i,j\right) = \int p\left(x \mid \theta^*\right) \frac{\partial \log p\left(x \mid \theta\right)}{\partial \theta_i} \frac{\partial \log p\left(x \mid \theta\right)}{\partial \theta_j} dx$$

Other Possible Divergences

The Kullback-Leibler Divergence is asymmetric and tricky...

All divergences have: $D\left(p,q\right) \geq 0, D\left(p,q\right) = 0 \, iff \, p = q$

Consider KL, Jensen-Shannon and Symmetrized KL...

$$KL\left(p,q\right) = \int p\left(x\right) \log p\left(x\right) q^{-1}\left(x\right) dx$$

$$JS\left(p,q\right) = \tfrac{1}{2} KL\left(p, \tfrac{p+q}{2}\right) + \tfrac{1}{2} KL\left(q, \tfrac{p+q}{2}\right)$$

$$SKL\left(p,q\right) = \tfrac{1}{2} KL\left(p,q\right) + \tfrac{1}{2} KL\left(q,p\right)$$

sqrt(JS) is a true distance metric (Endres & Schindelin, 2003)

Hellinger divergence is symmetric & can be written as kernel!

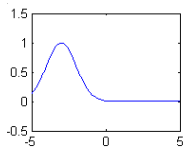
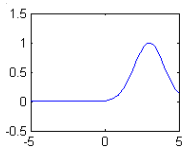
$$H\left(p,q\right) = \tfrac{1}{4} \sqrt{\int \left(\sqrt{p\left(x\right)} - \sqrt{q\left(x\right)}\right)^2 dx} = \sqrt{k\left(p,p\right) - 2k\left(p,q\right) + k\left(q,q\right)}$$

...what kernel is this?

Bhattacharyya Kernel

A kernel between distributions that gives Hellinger divergence

$$k(p,q) = \int \sqrt{p(x)}\sqrt{q(x)}\,dx$$

$p(x)$   $q(x)$

A slight generalization is the *Probability Product Kernel*

$$k(p,q) = \int \left(p(x)\right)^{\rho}\left(q(x)\right)^{\rho}\,dx$$

Setting ρ=1/2 gives the *Bhattacharyya Kernel*

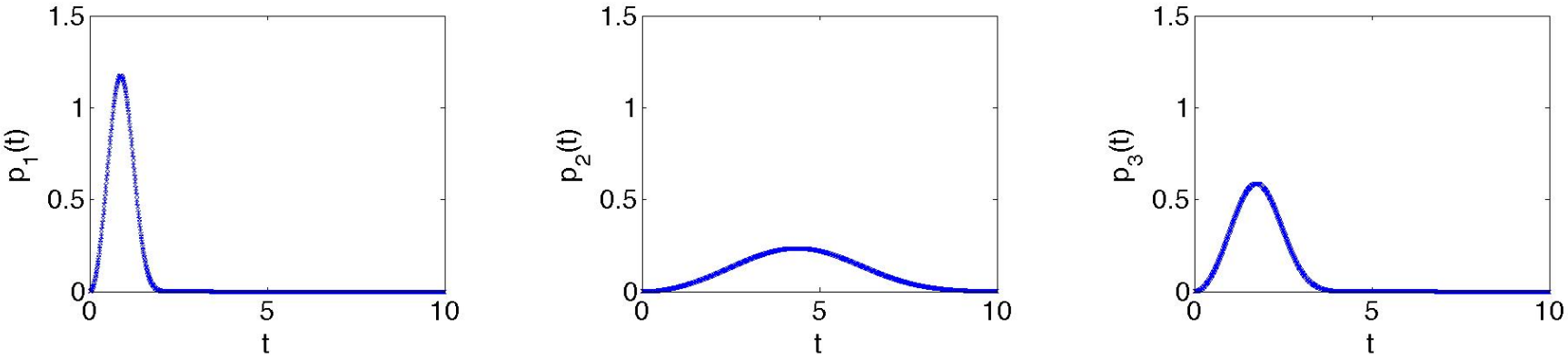
$$k(p,q) = \int \sqrt{p(x)}\sqrt{q(x)}\,dx$$

Setting ρ=1 gives the *Expected Likelihood Kernel*

$$k(p,q) = E_{p(x)}\left[q(x)\right] = E_{q(x)}\left[p(x)\right]$$

Probability Product Kernel

Imagine we are given inputs which are probabilities and labels
Dataset = $\{(p_1,y_1), (p_2,y_2),\ldots, (p_n,y_n)\}$



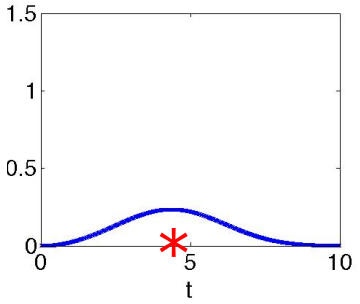
E.g. each input i is a brand of car
p_i is its brakepad failure probability over time
y_i is binary, did government approve it for US roads?

Build SVM with Probability Product Kernels to predict if a brakepad failure distribution will be approved/rejected by gov't

Probability Product Kernel

Imagine we are given input data vectors with labels
Dataset = $\{(x_1,y_1), (x_2,y_2),\ldots, (x_n,y_n)\}$

E.g. each input i is a brand of car
x_i is time when one car of brand i failed
y_i is binary, did government approve it for US roads?



For each i, estimate from
$x_i \rightarrow$ probability $p_i=p(x|\theta_i)$

Dataset = $\{(p_1,y_1), (p_2,y_2),\ldots, (p_n,y_n)\}$

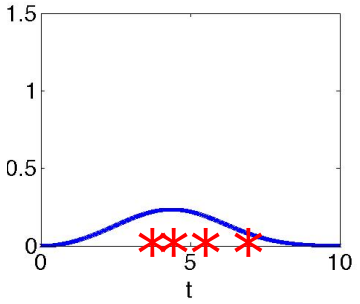
Build SVM with Probability Product Kernels to predict if a brakepad failure distribution will be approved/rejected by gov't

Probability Product Kernel

Imagine we are given input data vector-sets with labels
Dataset = $\{(\chi_1,y_1), (\chi_2,y_2),\ldots, (\chi_n,y_n)\}$

E.g. each input i is a brand of car
χ_i is a set of times when several cars of brand i failed
y_i is binary, did government approve it for US roads?

For each i, estimate from
$\chi_i=\{x_{i1},\ldots,x_{it}\}$ → probability $p_i=p(x|\theta_i)$



Dataset = $\{(p_1,y_1), (p_2,y_2),\ldots, (p_n,y_n)\}$

Build SVM with Probability Product Kernels to predict if a brakepad failure distribution will be approved/rejected by gov't

Gaussian Product Kernels

For Gaussians: $$k\left(p,p'\right)=\int N^{\rho}\left(x\mid\mu\right)N^{\rho}\left(x\mid\mu'\right)dx$$

$$\propto\exp\left(-\left\|\mu-\mu'\right\|^{2}/\tilde{\sigma}\right)$$

(if μ=x and μ'=x', get back RBF kernel ☺)
(Fisher kernel here is just a linear kernel ☹)

For Gaussians with variable covariance:

$$k\left(p,p'\right)=\int N^{\rho}\left(x\mid\mu,\Sigma\right)N^{\rho}\left(x\mid\mu',\Sigma'\right)dx$$

$$\propto\left|\Sigma\right|^{-\rho/2}\left|\Sigma'\right|^{-\rho/2}\left|\bar{\Sigma}\right|^{1/2}\exp\left(-\tfrac{\rho}{2}\mu^{T}\Sigma^{-1}\mu-\tfrac{\rho}{2}\mu'^{T}\Sigma'^{-1}\mu'+\tfrac{1}{2}\bar{\mu}^{T}\bar{\Sigma}\bar{\mu}\right)$$

$$where\ \ \bar{\Sigma}=\left(\rho\Sigma^{-1}+\rho\Sigma'^{-1}\right)^{-1}\ \ and\ \ \bar{\mu}=\rho\Sigma^{-1}\mu+\rho\Sigma'^{-1}\mu'$$

(Fisher kernel here is just a quadratic kernel ☹)

Multinomial Product Kernels

Bernoulli: (binary x)

$$p(x \mid \theta) = \prod_{d=1}^{D} \gamma_d^{x_d} \left(1 - \gamma_d\right)^{1-x_d}$$

$$k(\chi, \chi') = \int \left(p(x)\right)^{\rho} \left(p'(x)\right)^{\rho} dx$$

$$= \prod_{d=1}^{D} \left[\left(\gamma_d \gamma_d'\right)^{\rho} + \left(1 - \gamma_d\right)^{\rho} \left(1 - \gamma_d'\right)^{\rho} \right]$$

Multinomial: (discrete x)

$$p(x \mid \theta) = \prod_{d=1}^{D} \alpha_d^{x_d}$$

$$k(\chi, \chi') = \sum_{d=1}^{D} \left(\alpha_d \alpha_d'\right)^{\rho}$$

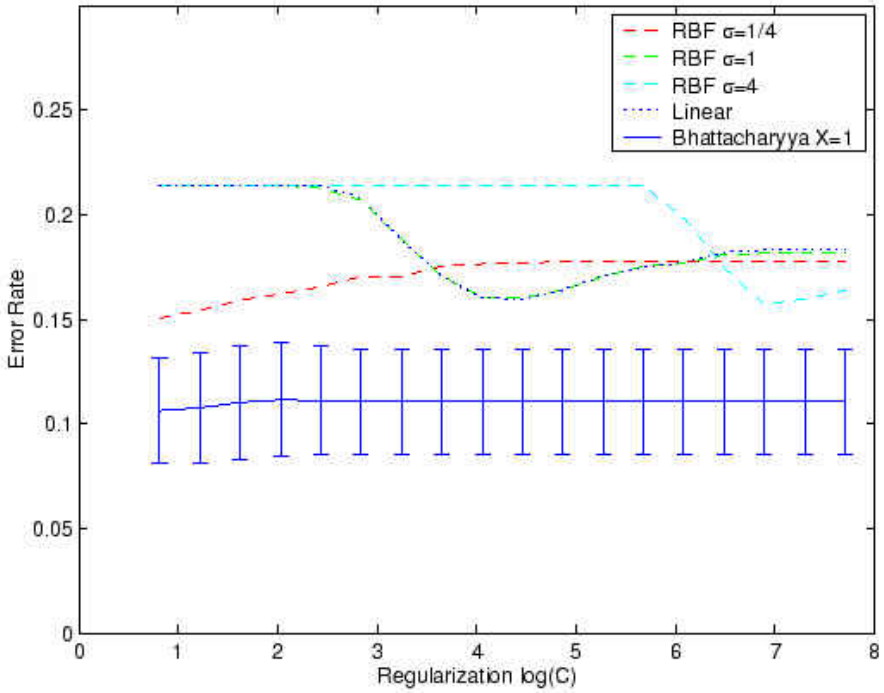
For multinomial counts (for N words per document):

$$k(\chi, \chi') = \left[\sum_{d=1}^{D} \left(\alpha_d \alpha_d'\right)^{1/2} \right]^{N}$$

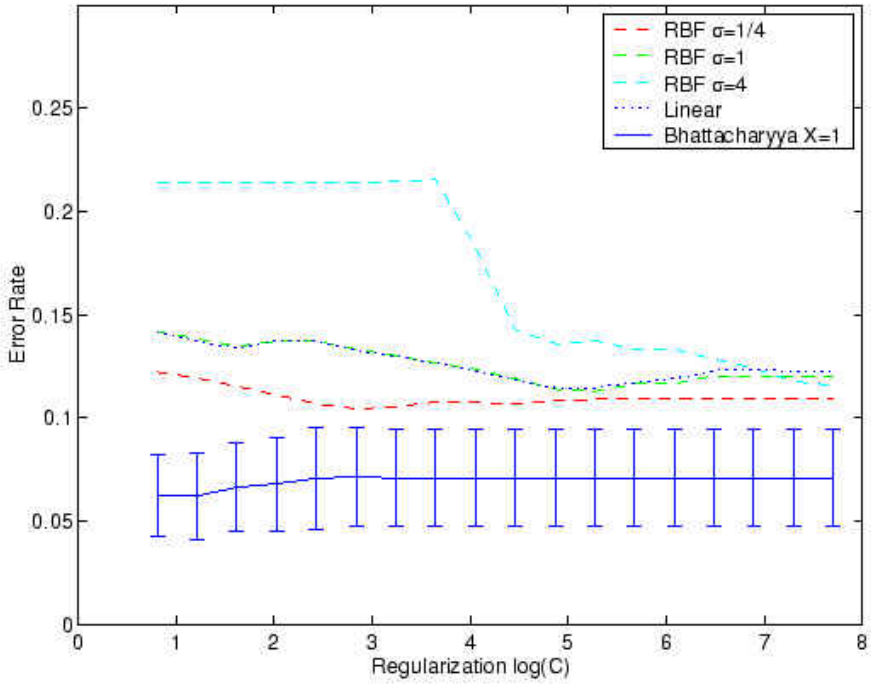
(Fisher for Multinomial is linear ☹)

Multinomial Product Kernels

WebKB dataset: Faculty vs. student web page SVM classifier
20-Fold Cross-Validation, 1641 student & 1124 faculty, ...
Use Bhattacharyya Kernel on multinomial (word frequency)



Training Set Size = 77



Training Set Size = 622

Exponential Family Kernels

Exponential Family: Gaussian, Multinomial, Binomial, Poisson, Inverse Wishart, Gamma, Bernoulli, Dirichlet, …

$$p\left(x \mid \theta\right) = \exp\left[A\left(x\right) + T\left(x\right)^T \theta - K\left(\theta\right)\right]$$

Maximum likelihood is straightforward: $\frac{\partial}{\partial\theta} K\left(\theta\right) = \frac{1}{n}\sum_n T\left(x_n\right)$

All have above form but different A(x), T(x), convex K(q)

Family	$\mathcal{A}(X)$	$\mathcal{K}(\theta)$
Gaussian (mean)	$-\frac{1}{2}X^T X - \frac{D}{2}\log(2\pi)$	$\frac{1}{2}\theta^T\theta$
Gaussian (variance)	$-\frac{1}{2}\log(2\pi)$	$-\frac{1}{2}\log(\theta)$
Multinomial	$\log(\Gamma(\eta+1)) - \log(\nu)$	$\eta\ \log(1+\sum_{d=1}^{D}\exp(\theta_d))$
Exponential	0	$-\log(-\theta)$
Gamma	$-\exp(X) - X$	$\log\Gamma(\theta)$
Poisson	$\log(X!)$	$\exp(\theta)$

Exponential Family Kernels

Compute the Bhattacharyya Kernel for the e-family:

$$p\left(x \mid \theta\right) = \exp\left(A\left(x\right) + T\left(x\right)^T \theta - K\left(\theta\right)\right)$$

Analytic solution for e-family:

$$k\left(p, p'\right) = \int p\left(x \mid \theta\right)^{1/2} p\left(x \mid \theta'\right)^{1/2} dx$$
$$= \exp\left(K\left(\tfrac{1}{2}\theta + \tfrac{1}{2}\theta'\right) - \tfrac{1}{2}K\left(\tfrac{1}{2}\theta\right) - \tfrac{1}{2}K\left(\theta'\right)\right)$$

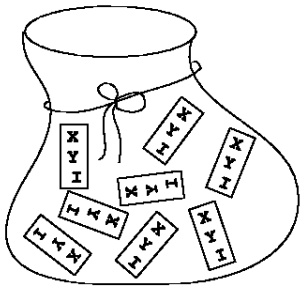
Only depends on convex cumulant-generating function K(q)

Meanwhile, Fisher Kernel is always linear in sufficient stats...

$$U_x = \nabla_{\theta^*} \log p\left(\chi \mid \theta\right) = T\left(\chi\right) - \nabla_{\theta^*} K\left(\theta\right)$$
$$U_\chi I_{\theta^*}^{-1} U_{\chi'} = \left(T\left(\chi\right) - g\right)^T I_{\theta^*}^{-1} \left(T\left(\chi'\right) - g\right)$$

Gaussian Product Kernels

Instead of a single x & x'
Construct p & p' from many x & x' samples



Use bag of vectors

$$\left\{\chi_1,\ldots,\chi_N\right\} \rightarrow p\left(x\right) = N\left(x \mid \mu,\Sigma\right)$$

$$\mu = \frac{1}{N}\sum_i \chi_i,\ \Sigma = \frac{1}{N}\sum_i \left(\chi_i - \mu\right)\left(\chi_i - \mu\right)^T$$

$$\left\{\chi_1',\ldots,\chi_{N'}'\right\} \rightarrow p'\left(x\right) = N\left(x \mid \mu',\Sigma'\right)$$

$$\mu' = \frac{1}{N}\sum_i \chi_i',\ \Sigma' = \frac{1}{N}\sum_i \left(\chi_i' - \mu\right)\left(\chi_i' - \mu\right)^T$$

$$k\left(p,p'\right) = \left|\Sigma\right|^{-\rho/2}\left|\Sigma'\right|^{-\rho/2}\left|\bar{\Sigma}\right|^{1/2}\exp\left(-\frac{\rho}{2}\mu^T\Sigma^{-1}\mu - \frac{\rho}{2}\mu'^T\Sigma'^{-1}\mu' + \frac{1}{2}\bar{\mu}^T\bar{\Sigma}\bar{\mu}\right)$$

Kernelized Gaussian Product

Bhattacharyya affinity on Gaussians on bags {x,...} & {x′,...}
Invariant to order of tuples in each bag
But too simple, just overlap of Gaussians on images


Need more detail than mean and covariance of pixels…
Use another Kernel to find Gaussian in Hilbert space

$$\kappa\left(\chi,\chi'\right)=\phi\left(\chi\right)^{T}\phi\left(\chi'\right)$$

Never compute outerproducts, use kernel, i.e. infinite RBF:

$$\kappa\left(\chi,\chi'\right)=\exp\left(-{}^{1}\!/_{2\sigma^{2}}\left\|\chi-\chi'\right\|^{2}\right)$$

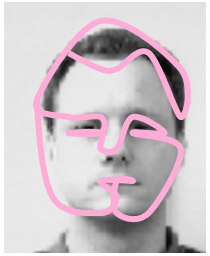
Compute mini-kernel between each pixel in a given image…
Gives kernelized or augmented Gaussian μ and Σ via Gram

Kernelized Gaussian Product

Previously: $\mu = E\{\chi\} \qquad \Sigma = E\left\{(\chi-\mu)(\chi-\mu)^T\right\}$

Now: $\mu = E\{\phi(\chi)\} \qquad \Sigma = E\left\{(\phi(\chi)-\mu)(\phi(\chi)-\mu)^T\right\}$

Compute Hilbert Gaussian's mean & covariance of each image
bag or image is N x N pixel Gram matrix using kappa kernel
Use kernel PCA to regularize infinite dimensional RBF Gaussian

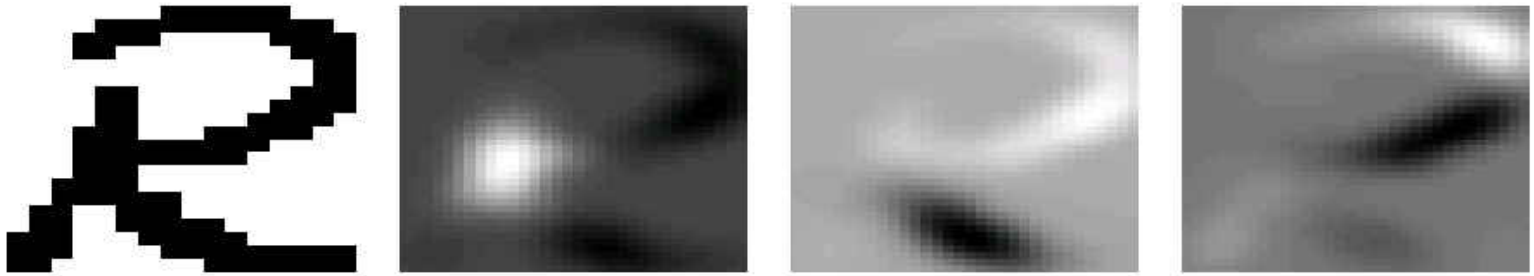


$$\begin{bmatrix} \kappa(x_1,x_1) & \dots & \kappa(x_1,x_N) \\ \vdots & & \vdots \\ \kappa(x_N,x_1) & \dots & \kappa(x_N,x_N) \end{bmatrix} \begin{bmatrix} \kappa(x_1,x_1) & \dots & \kappa(x_1,x_M) \\ \vdots & & \vdots \\ \kappa(x_M,x_1) & \dots & \kappa(x_M,x_M) \end{bmatrix}$$

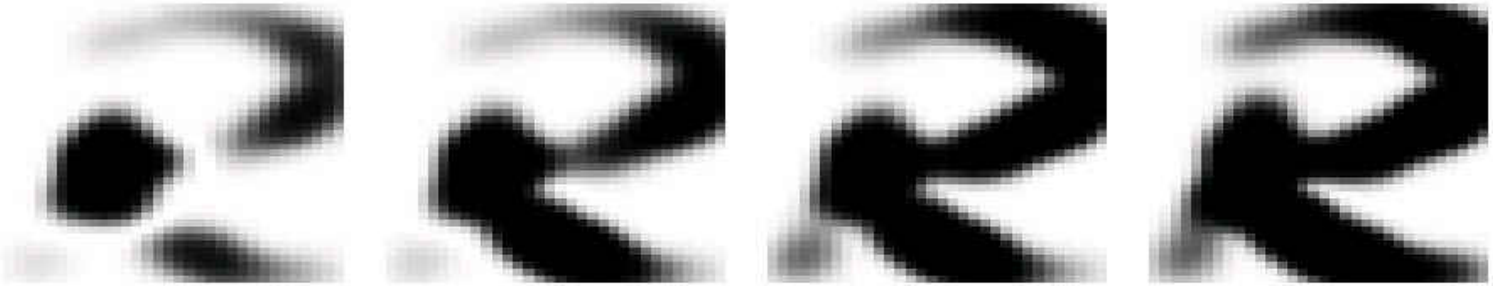
$$k\left(\phi(\chi)\middle\|\phi(\chi')\right) \propto \left|\Sigma\right|^{-\rho/2}\left|\Sigma'\right|^{-\rho/2}\left|\bar{\Sigma}\right|^{1/2}\exp\left(-\tfrac{\rho}{2}\mu^T\Sigma^{-1}\mu-\tfrac{\rho}{2}\mu'^T\Sigma'^{-1}\mu'+\tfrac{1}{2}\bar{\mu}^T\bar{\Sigma}\bar{\mu}\right)$$

Puts all dimensions (X,Y,I) on an equal footing

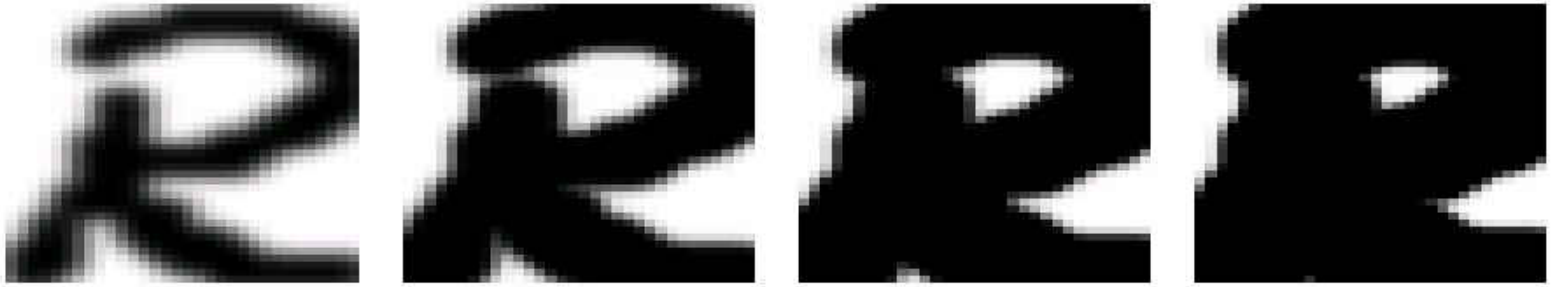
Kernelized Gaussian Product



Letter ‘R’ with 3 KPCA Components of RBF Kernel



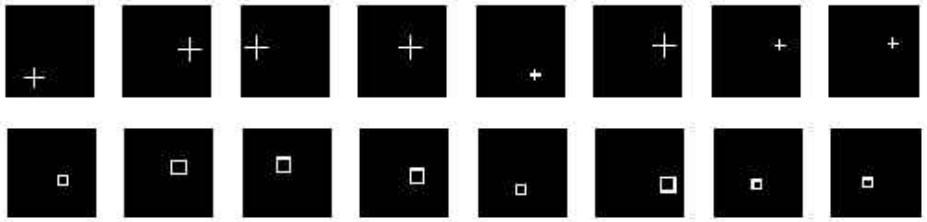
Reconstruction of Letter ‘R’ with 1-4 KPCA with RBF Kernel



Reconstruction of Letter ‘R’ with 3 KPCA with RBF Kernel + Smoothing

Kernelized Gaussian Product

100 40x40 monochromatic images of crosses & squares translating & scaling

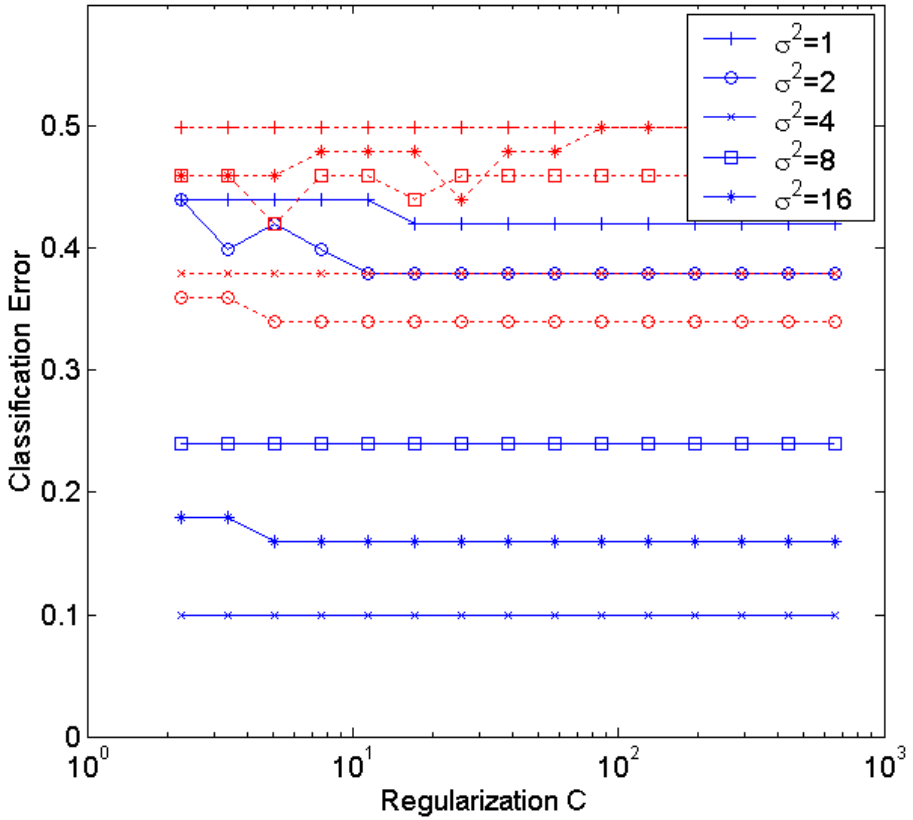


SVM: Train 50, Test 50

Fisher for Gaussian is Quadratic Kernel

RBF Kernel (red) 67% accuracy

Bhattacharyya (blue) 90% accuracy

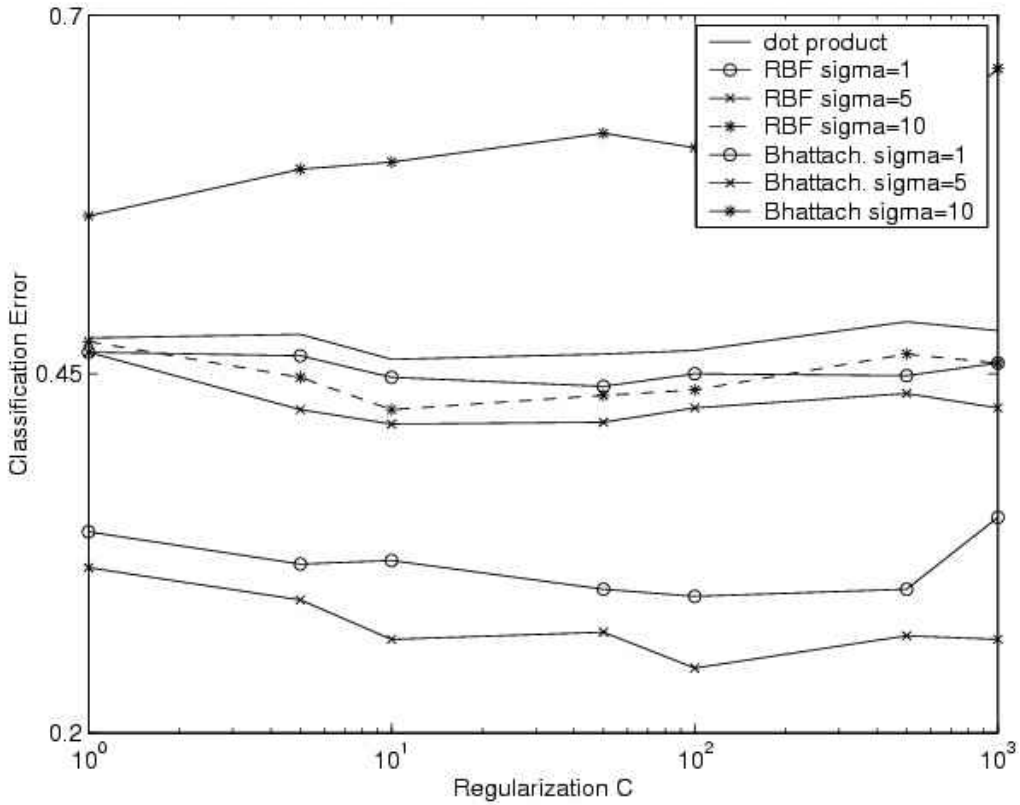


Kernelized Gaussian Product

SVM Classification of NIST digit images 0,1,…,9
Sample each image to get bag of 30 (X,Y) pixels
Train on random 120, test on random 80

bag-of-vectors
Bhattacharyya
outperforms
standard RBF
due to built-in
invariance

Fisher Kernel for
Gaussian is quadratic

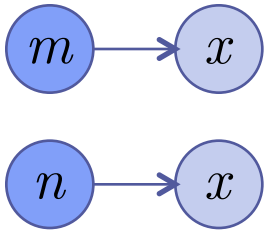


Mixture Product Kernels

Beyond Exponential Family: Mixtures and Hidden Variables Need ρ=1 Expected Likelihood kernel…

$$\chi \rightarrow p(x) = \sum_{m=1}^{M} p(m)p(x \mid m)$$

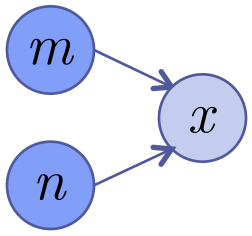
$$\chi' \rightarrow p'(x) = \sum_{n=1}^{N} p'(n)p'(x \mid n)$$



$$k(\chi,\chi') = \int p(x)p'(x)dx$$

$$= \sum_{m=1}^{M}\sum_{n=1}^{N} p(m)p'(n)\int p(x \mid m)p'(x \mid n)dx$$

$$= \sum_{m=1}^{M}\sum_{n=1}^{N} p(m)p'(n)c_{m,n}$$



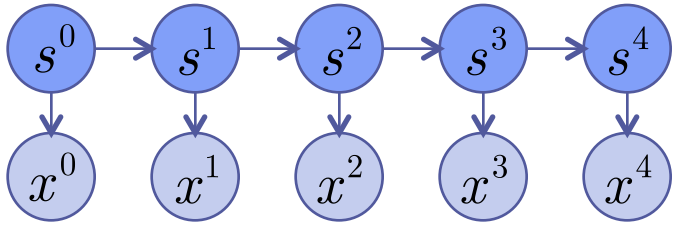
Use M*N subkernel evaluations from our previous repertoire

HMM Product Kernels

Hidden Markov Models: (sequences)

$$p(x) = \sum_S p(s_0)\, p(x_0 \mid s_0) \prod_{t=1}^{T} p(s_t \mid s_{t-1})\, p(x_t \mid s_t)$$

# of hidden configurations large



$$\#configs = |s|^T$$

Kernel: $k(\chi, \chi') = \int p(x)\, p'(x)\, dx$

$$= \sum_S \sum_U p(S)\, p'(U) \int p(x \mid S)\, p'(x \mid U)\, dx$$

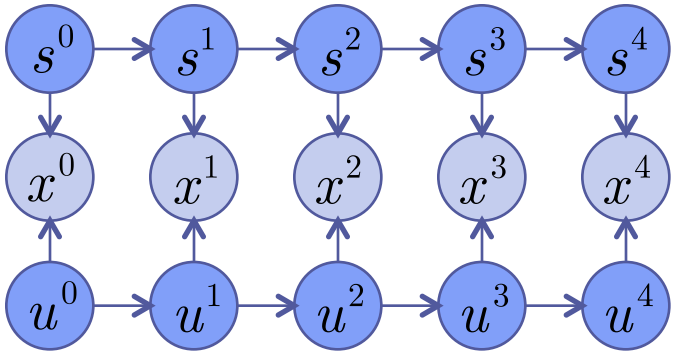
$$= \sum_S \sum_U p(S)\, p'(U)\, c_{S,U}$$

☹ computes cross-product of all hidden variables $O\left(|s|^T \times |u|^T\right)$

HMM Product Kernels

$$k(\chi,\chi') = \sum_S \sum_U \prod_t p(s_t \mid s_{t-1}) p'(u_t \mid u_{t-1}) \int p(x_t \mid s_t) p'(x_t \mid u_t) dx_t$$
$$= \sum_{S,U} \prod_t p(s_t \mid s_{t-1}) p'(u_t \mid u_{t-1}) c_{s_t,u_t}$$
$$= \sum_{S,U} p(s_0) p'(u_0) \psi(s_0,u_0) \prod_t p(s_t \mid s_{t-1}) p'(u_t \mid u_{t-1}) \psi(s_t,u_t)$$

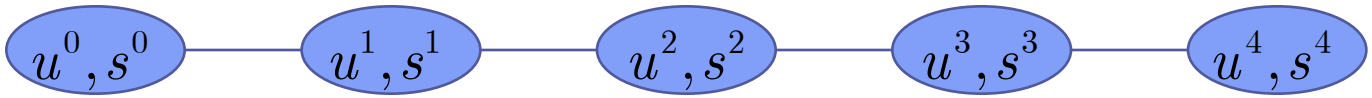
Take advantage of structure in HMMs via Bayesian network



Only compute subkernels for common parents

Evaluate total of $O(T|s||u|)$ subkernels

Form clique potential fn's, sum via junction tree algorithm



Sampling Product Kernels

Can approximate probability product kernel via sampling

By definition, generative models can:

1) Generate a Sample
2) Compute Likelihood of a Sample

Thus, approximate probability product via sampling:

$$k\left(\chi,\chi'\right)=k\left(p,p'\right)=\int p\left(x\right)p'\left(x\right)dx$$

$$\approx \frac{\beta}{N}\sum_{\substack{x_i\sim p\left(x\right)\\ i=1\ldots N}} p'\left(x_i\right)+\frac{1-\beta}{N'}\sum_{\substack{x_i'\sim p'\left(x\right)\\ i=1\ldots N'}} p\left(x_i'\right)$$

Beta controls how much sampling from each distribution...