Kernels Between Distributions & Sets

Tony Jebara
Risi Kondor
Machine Learning Lab
Columbia University
June 2003
Outline

Vectors vs. Sets of Vectors
Generative Models of Sets
Kernels on Sets
Kernels on Probability Models
Hellinger & Bhattacharyya

Probability Product Kernels:
  Exponential Family
  Bernoulli & Multinomial
  Gaussian
  Kernelized Gaussian
Mixture Models
Hidden Markov Models
Bayesian Networks
Sampling Methods
Non-Vector Data

Part 1: Generative Models on Sets

Part 2: Kernels on Sets & Distributions
Part 1:

Generative Models on Sets of Vectors

(...as opposed to Vectors)
Modeling Data as Vectors & Vectorization

Many learning methods expect datum as vector

\[ \vec{x}_t = \begin{bmatrix} 0.7 & -2 & 3 & 6 \end{bmatrix}^T \]

But, vectorizing an object into vector is dangerous

**Images**: morph, rotate, translate, zoom...

**Audio**: pitch changes, ambient acoustics...

**Video**: motion, camera view, angles...

**Gene Data**: proteins fold, insertions, deletions...

Want Invariance: factor out certain variations (translations)
Want Linearity: model desired variations (identity) linearly
But, above variations are highly nonlinear in vector representation

i.e. image translation:

\[ \vec{x}_t = T^t \vec{x}_0 \]
Alternative: Avoid Vectorizing, Use “Bag of Vectors”

Since vectorization so nonlinear, avoid it from outset
View a datum as “Bag of Vectors” instead of single Vector
i.e. grayscale image = Set of Vectors or Bag of Pixels
(N pixels, each is a D=3 XYI tuple)

Why? Image Vectorization dumps DxN dimensional vector by lexicographic ordering. Over many images, ordering stays constant, only intensities vary. All variation captured only by intensities changes
Why Bags of Pixels or Sets of Vectors?

Vectorization / Rasterization: uses index in image to sort pixels into large vector. Dataset only shows variations in I entries of large vector (spatial is nonlinear)

\[
\vec{x}_1 = \left\{ (X^1, Y^1, I_1^1), (X^2, Y^2, I_1^2), \ldots, (X^N, Y^N, I_1^N) \right\}
\]

\[
\vec{x}_2 = \left\{ (X^1, Y^1, I_2^1), (X^2, Y^2, I_2^2), \ldots, (X^N, Y^N, I_2^N) \right\}
\]

If we knew “optimal” correspondence:
could sort pixels in the bag into large vector more appropriately. Dataset shows jointly linear variations in X, Y and I entries

\[
\vec{x}_1 = \left\{ (X^5_1, Y^5_1, I^5_1), (X^8_1, Y^8_1, I^8_1), \ldots, (X^2_1, Y^2_1, I^2_1) \right\}
\]

\[
\vec{x}_2 = \left\{ (X^3_2, Y^3_2, I^3_2), (X^4_2, Y^4_2, I^4_2), \ldots, (X^9_2, Y^9_2, I^9_2) \right\}
\]
Why Bags of Pixels or Sets of Vectors?

As vector images, linear changes & eigenvectors are additions and deletions of intensities (awkward). Translating, raising eyebrows, etc. involve erasing & redrawing.

In bag of pixels (vectorized only after optimal correspondence) see linear changes and eigenvectors are morphings, warpings, jointly spatial and intensity change.

But, we don’t know optimal correspondence, must learn it.
Bag Representation     Permutation     Manifold

Assume order unknown. “Set of Vectors” or “Bag of Pixels”
Get permutational invariance (order doesn’t matter)

Can’t represent invariance by single ‘X’ vector point in DxN space
since we don’t know the ordering

Get permutation invariance by ‘X’ spanning all possible reorderings
Multiply X by unknown A matrix (permutation or doubly-stochastic)
Invariant Paths as Matrix Operators on Vectors

Move vector $X$ along a manifold by multiplying by a matrix: $AX$

Restrict A to be permutation matrix
Resulting manifold of configurations is an “orbit” if A is a group
Or, for smooth manifold, A is doubly-stochastic matrix

*Endow* each image in dataset with its own transformation matrix A
Each image is now a bag or manifold: \[ \left\{ A_1 X_1, \ldots, A_T X_T \right\} \]
Modeling a Dataset of Invariant Manifolds

Example: assume model is PCA, learn 2D subspace of 3D data
Permutation indicates points can move independently along paths
Find PCA after moving to form ‘tight’ 2D subspace
More generally, move along manifolds to improve fit of any model (PCA, SVM, probability density, etc.)
Explicitly Handling Permutations (AISTAT 2003)

Borrow from SVM: regularization cost + linear constraints on model
Here have: modeling cost + linear constraints on transformations
Estimate $A = \{A_1, \ldots, A_T\}$ transformation parameters
and $\Theta$ model parameters (PCA, Gaussian, SVM)

Cost function on matrices $A$ emerges from $\Theta$ modeling criterion
Min Convex Cost with Convex Hull of Constraints (Unique!)

$$\min_A C(A_1, \ldots, A_T)$$

subject to: $\sum_{i,j} A_{ij}^t Q_{td}^j + b_{td} \geq 0 \ \forall t, d$

Since $A$ matrices are soft permutation matrices (doubly-stochastic) we have:

$$\sum_i A_{ij}^t = 1 \ \sum_j A_{ij}^t = 1 \ \ A_{ij}^t \geq 0$$
Cost $C(A)$? Gaussian Mean

Maximum Likelihood Gaussian Mean Model:

$$l(A, \mu) = \sum_t \log N(A_t X_t; \mu, I)$$
$$\hat{\mu} = \frac{1}{T} \sum_t A_t X_t$$

$$l(A, \hat{\mu}) = -\frac{T D}{2} \log 2\pi - \frac{1}{2} \sum_t \|A_t X_t - \hat{\mu}\|^2$$

$$C(A) = -l(A, \hat{\mu}) = \text{trace}(\text{Cov}(AX))$$

Theorem 1: $C(A)$ is convex in $A$ (Convex Program)

Can solve via a quadratic program on the $A$ matrices

Minimizing the trace of a covariance tries to pull the data spherically towards a common mean
Cost $C(A)$? Gaussian Mean & Covariance

$$l(A, \mu, \Sigma) = \sum_t \log N(A_t X_t; \mu, \Sigma)$$

$$\hat{\mu} = \frac{1}{T} \sum_t A_t X_t \quad \hat{\Sigma} = \frac{1}{T} \sum_t (A_t X_t - \hat{\mu}) (A_t X_t - \hat{\mu})^T$$

$$l(A, \hat{\mu}, \hat{\Sigma}) = -T \frac{D}{2} \log 2\pi - T \frac{1}{2} \log |\hat{\Sigma}| - \frac{T}{2} \sum_t (A_t X_t - \hat{\mu})^T \hat{\Sigma}^{-1} (A_t X_t - \hat{\mu})$$

$$C(A) = -l(A, \hat{\mu}, \hat{\Sigma}) = |\text{Cov}(AX)|$$

**Theorem 2:** Regularized log determinant of a covariance is convex

Equivalently, minimize

$$\log |\text{Cov}(AX) + \varepsilon I| + \varepsilon \text{tr} \left( \text{Cov}(AX) \right)$$

**Theorem 3:** Cost not quadratic but can be upper bounded by quad

Iteratively solve quadratic program with variational bound

$$\log |S| \leq \text{trace} \left( S_0^{-1} S \right) + \log |S_0| - \text{trace} \left( S_0^{-1} S_0 \right)$$

Minimizing determinant flattens data into a pancake of low volume
Cost C(A)? Fisher Discriminant

Find linear discriminant model ‘w’ that maximizes between / within class scatter

\[
\max_w \frac{w^T U w}{w^T S w}
\]

\[
U = (\mu_+ - \mu_-)(\mu_+ - \mu_-)^T
\]

\[
S = \Sigma_+ + \Sigma_-
\]

For discriminative invariance learning, estimate transformation matrices to:

- increase between-class scatter (numerator)
- reduce within class scatter (denominator)

\[
C'(A) = \left| \Sigma_+ + \Sigma_- - \lambda (\mu_+ - \mu_-)(\mu_+ - \mu_-)^T \right|
\]

Minimizing this tries to permute data to make classification easy
Cost Function C(A) Interpretations

Maximum Likelihood Mean
Permute data towards common mean

Maximum Likelihood Mean & Covariance
Permute data towards flat subspace
Pushes energy into few eigenvectors
Great as pre-processing before PCA

Fisher Discriminant
Permute data towards two flat
subspaces while repelling away
from each other’s means
Practical Optimization of Quadratic Programs

Quadratic Programming used for all C(A) since:
- Gaussian Mean quadratic
- Gaussian Covariance upper boundable by quadratic
- Fisher Discriminant upper boundable by quadratic

\[ \text{trace}(MS) = \frac{1}{T} \sum_{mpnqij} A_{ij}^{mn} A_{ij}^{pq} X_i^j M_{ij}^{pm} X_i^n - \frac{1}{T^2} \sum_{mpnqij} A_{ij}^{mn} A_{ij}^{pq} X_i^j M_{ij}^{pm} X_i^n \]

Use Sequential Minimal Optimization
axis parallel optimization, pick axes to update,
ensure constraints not violated

Soft permutation matrix 4 constraints
or 4 entries at a time

\[ A_t^{mn}, A_t^{mq}, A_t^{pn}, A_t^{pq} \]
Digits: Image = Bag of XY Vectors

20 Images of ‘3’ and ‘9’
Each is 70 (x,y) dots
No order on the ‘dots’

PCA compress with same number of Eigenvectors

Convex Program first estimates the permutation better reconstruction

Original

PCA

Permuted PCA
Linear Interpolation

Intermediate images are smooth morphs

Points nicely corresponded

Spatial morphing versus ‘redrawing’

No ghosting
Single Person Faces: Image = Bag of XYI Pixels

2000 XYI Pixels: Compress to 20 dims
Improve squared error of PCA by Almost 3 orders of magnitude x10^3
Multi-Person Faces: Bag-of-Pixels Eigenvectors

+- Scaling on Eigenvector

Top 5 Eigenvectors

All just linear variations in bag of XYI pixels

Vectorization nonlinear needs huge # of eigenvectors
Multi-Person Faces: Bag-of-Pixels Eigenvectors

 +/- Scaling on Eigenvector

Next 5 Eigenvectors
Classification: Distances & Affinity between Two Bags?

For nearest neighbor classification: need distances between two bags.

For SVM classification: need kernel or affinity between two bags.

Could solve permutations, find closest point between two manifolds:

But, can we avoid computing optimal permutations (work)? Implicitly compute distances or affinities?
Implicitly Handling Permutations: Kernels on PDFs

Idea:
1) Each bag has IID vectors in it
2) Model each bag using a probability density (e.g. Gaussian)
3) Build kernel classifier by measuring affinity between PDFs
Part 2: Kernelizing ...

Kernels on Sets of Vectors & Kernels on PDFs
Kernels from Generative Models

Combines Generative & Discriminative Tools

Use a Kernel derived from generative model

Fisher Kernels: Jaakkola & Haussler
Convolution / Transducer Kernels: Haussler, Watkins, Cortes
Diffusion Kernels: Kondor, Lafferty
Heat Kernels: Lafferty, Lebanon

Model distribution of points
(one, some or all)

Density helps compute affinities
& kernels in machine (SVM)

\[ K(\chi, \chi') \]
Fisher Kernels & Kullback-Leibler Divergence

Fisher Kernel: approximate distance between two generative models on statistical manifold using Kullback-Leibler

\[ KL(p \parallel p') = \int p \log \frac{p}{p'} \, dx \]

Approximate KL by quadratic form local tangent space at ML estimate

\[ KL \approx \frac{1}{2} (\theta - \theta')^T \mathbf{I}_{\theta^*} (\theta - \theta') \]

From distance get affinity via Fisher Info & gradients

\[ K(\chi, \chi') \approx U_x I_{\theta^*}^{-1} U_{x'} \]
\[ U_x = \nabla_{\theta^*} \log p(\chi | \theta) \]

Heat Kernel: (Lafferty & Lebanon) better geodesic distance & affinity, Only solvable for multinomial (sphere) or covariance (hyperbolic)
Hellinger Divergence & Bhattacharyya (COLT’03)

KL not symmetric, needs approximation, lose interesting nonlinearities

\[ KL(p \| p') = \int p \log \frac{p}{p'} \longrightarrow K(\chi, \chi') \approx U \chi I_{\theta^*}^{-1} U_{\chi'} \]

Instead: START with nice divergence, avoid approximation

Other choices & bounds for divergence (Topsoe, Dragomir):
   Triangular, L1, Hellinger, Harmonic, Variational, Csiszar’s f-Div, ...

Hellinger Divergence

\[ HD(p \| p') = \int (\sqrt{p} - \sqrt{p'})^2 \longrightarrow BA(p, p') = \int \sqrt{p} \sqrt{p'} \, dx \]

Desiderata:

Mercer Kernel? +ve? +ve definite? \ YES
Symmetric? \ YES
Computable for many distributions? \ YES
Interesting nonlinear behavior? \ YES
Probability Product Kernels

Computing the kernel: given inputs $\chi$ and $\chi'$

1) Estimate Densities (i.e. ML):

$\chi \rightarrow p(x) = p(x \mid \theta)$

$\chi' \rightarrow p'(x) = p(x \mid \theta')$

2) Compute Bhattacharyya Affinity:

$K(\chi, \chi') = \int \sqrt{p} \sqrt{p'} \, dx$

Probability Product Kernel:

$K(\chi, \chi') = \int (p)^{\rho} (p')^{\rho} \, dx$

Bhattacharyya Kernel:

$\rho = \frac{1}{2}$

Expected Likelihood Kernel:

$\rho = 1$

$K(\chi, \chi') = E_p[p'] = E_{p'}[p]$
Exponential Family Product Kernels

Many Properties, includes Gaussian Mean, Covariance, Multinomial, Binomial, Poisson, Exponential, Gamma, Bernoulli, Dirichlet, ...

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

Maximum likelihood is straightforward: \[ \frac{\partial}{\partial \theta} K(\theta) = \frac{1}{n} \sum_{n} T(x_n) \]

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

<table>
<thead>
<tr>
<th>Family</th>
<th>( A(X) )</th>
<th>( K(\theta) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gaussian (mean)</td>
<td>(-\frac{1}{2} X^T X - \frac{D}{2} \log(2\pi))</td>
<td>( \frac{1}{2} \theta^T \theta )</td>
</tr>
<tr>
<td>Gaussian (variance)</td>
<td>(-\frac{1}{2} \log(2\pi))</td>
<td>(-\frac{1}{2} \log(\theta))</td>
</tr>
<tr>
<td>Multinomial</td>
<td>( \log(\Gamma(\eta + 1)) - \log(\nu) )</td>
<td>( \eta \log(1 + \sum_{d=1}^{D} \exp(\theta_d)) )</td>
</tr>
<tr>
<td>Exponential</td>
<td>0</td>
<td>(-\log(-\theta))</td>
</tr>
<tr>
<td>Gamma</td>
<td>(- \exp(X) - X)</td>
<td>( \log \Gamma(\theta))</td>
</tr>
<tr>
<td>Poisson</td>
<td>( \log(X!))</td>
<td>( \exp(\theta))</td>
</tr>
</tbody>
</table>
Exponential Family Product Kernels

Compute the Bhattacharyya Kernel for the e-family:

\[ p(x | \theta) = \exp(A(x) + T(x)^T \theta - K(\theta)) \]

Analytic solution for e-family:

\[ K(\chi, \chi') = \int p(x | \theta)^{1/2} p(x | \theta')^{1/2} \, dx \]

\[ = \exp(K(\frac{1}{2} \theta + \frac{1}{2} \theta') - \frac{1}{2} K(\frac{1}{2} \theta) - \frac{1}{2} K(\theta')) \]

Only depends on convex cumulant-generating function \( K(\theta) \)

Meanwhile, Fisher Kernel is always linear in sufficient stats…

\[ U_x = \nabla_{\theta^*} \log p(\chi | \theta) = T(\chi) - \nabla_{\theta^*} K(\theta) \]

\[ U_{\chi} I^{-1}_{\theta^*} U_{\chi'} = (T(\chi) - g)^T I^{-1}_{\theta^*} (T(\chi') - g) \]
Multinomial and Bernoulli Product Kernels

**Bernoulli:** (binary)

\[ p(x | \theta) = \prod_{d=1}^{D} \gamma_x \gamma_d ^{(1 - \gamma_d)^{1-x_d}} \]

\[ K(\chi, \chi ') = \int (p(x))^\rho (p'(x))^{\rho} \, dx \]

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

**Multinomial:** (discrete)

\[ p(x | \theta) = \prod_{d=1}^{D} \alpha_d^{x_d} \]

\[ K(\chi, \chi ') = \sum_{d=1}^{D} (\alpha_d \alpha_{d}')^\rho \]

**For multinomial counts (for N words):**

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

Fisher for Multinomial is linear
Multinomial Product Kernel for Text

**WebKB dataset:** Faculty vs. student web page SVM kernel classifier

20-Fold Cross-Validation, 1641 student & 1124 faculty, ...

Use Bhattacharyya Kernel on multinomial (word frequency)

Training Set Size = 77

Training Set Size = 622
Gaussian Product Kernel

**Gaussian Mean:** (any $\rho$)

If $\mu = \chi$ and $\mu' = \chi'$ get RBF Kernel

Fisher here is linear

**Gaussian Covariance:** (any $\rho$)

$$K(\chi, \chi') = \int N^\rho(x \mid \mu, \Sigma) N^\rho(x \mid \mu', \Sigma') dx$$

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

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

Fisher here is quadratic

But, how do we get a non-degenerate covariance from 1 data point?
Gaussian Product Kernels on Bags of Vectors

Instead of a single $\chi$ & $\chi'$
Construct $\rho$ & $\rho'$ from many $\chi$ & $\chi'$
I.e. use bag of vectors

\[
\begin{align*}
\{\chi_1, \ldots, \chi_N\} & \rightarrow p(x) \\
\{\chi_1', \ldots, \chi_{N'}\} & \rightarrow p'(x) \\
\mu & = E\{\chi_i\} \\
\Sigma & = E\left\{(\chi_i - \mu)(\chi_i - \mu)^T\right\}
\end{align*}
\]

$\sim N(x \mid \mu, \Sigma)$

$\sim N(x \mid \mu', \Sigma')$

\[
K(\chi, \chi') = |\Sigma|^{-\rho/2} |\Sigma'|^{-\rho/2} |\Sigma^\dagger|^{1/2} \exp\left(-\frac{\rho}{2} \mu^T \Sigma^{-1} \mu - \frac{\rho}{2} \mu'^T \Sigma'^{-1} \mu' + \frac{1}{2} \mu'^T \Sigma' \Sigma \mu^\dagger\right)
\]
Kernelized Gaussian Product Kernel (ICML’03)

Bhattacharyya affinity on Gaussians on bags \( \{\chi, \ldots\} \) & \( \{\chi', \ldots\} \)

Invariant: to order of tuples in each bag
But too simple: overlap of two Gaussian distributions on images

Need more detail than mean and covariance of pixels...

Use Kernel Trick again when computing Gaussian mean & covariance

\[
\kappa(\chi, \chi') = \phi(\chi)^T \phi(\chi')
\]

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

\[
\kappa(\chi, \chi') = \exp \left( -\frac{1}{2\sigma^2} \| \chi - \chi' \|^2 \right)
\]

Compute mini-kernel between each pixel in a given image...
Gives kernelized or augmented Gaussian \( \mu \) and \( \Sigma \) via Gram
Kernelized Gaussian Product Kernel

Previously: \[ \mu = E \{ \chi \} \quad \Sigma = E \left\{ (\chi - \mu)(\chi - \mu)^T \right\} \]

Now have: \[ \mu = E \{ \phi(\chi) \} \quad \Sigma = E \left\{ (\phi(\chi) - \mu)(\phi(\chi) - \mu)^T \right\} \]

Still invariant to order of pixels

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

\[
K \left( \phi(\chi) \| \phi(\chi') \right) \propto |\Sigma|^{-\rho/2} |\Sigma'|^{-\rho/2} |\Sigma^\dagger|^1/2 \exp \left( -\frac{\rho}{2} \mu^T \Sigma^{-1} \mu - \frac{\rho}{2} \mu'^T \Sigma'^{-1} \mu' + \frac{1}{2} \mu^T \Sigma^\dagger \mu^\dagger \right)
\]

Puts all dimensions (X,Y,I) on an equal footing
Kernelized Gaussian Product Kernel

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 Kernel

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

SVM: Train on 50, Test on 50

Fisher for Gaussian is Quadratic Kernel

RBF Kernel (red) 67% accuracy

Bhattacharyya (blue) 90% accuracy
Kernelized Gaussian Product Kernel

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 Model Product Kernels

Beyond Exponential Family: Mixtures and Hidden Variables
Easier for $\rho=1$ Expected Likelihood kernel...

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

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

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

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

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

Use $M \times N$ subkernel evaluations from our previous repertoire
Hidden Markov Model Product Kernels

Hide

Hidden Markov Models: (sequences)

\[ p(x) = \sum_s p(s_0) p(x_0 | s_0) \prod_{t=1}^T p(s_t | s_{t-1}) p(x_t | s_t) \]

# of hidden configurations large

\[ \# \text{ configs} = |s|^T \]

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

\[ = \sum_{s} \sum_{U} p(s) p'(U) \int p(x | S) p'(x | U) \, dx \]

\[ = \sum_{s} \sum_{U} p(s) p'(U) K_{s,U}(x, x') \]

Do we need to consider raw cross-product of hidden variables? \( O(|s|^T \times |u|^T) \)
Hidden Markov Model Product Kernels

Do we need to consider cross-product of hiddens? NO!

\[
K(\chi, \chi') = \sum_s \sum_u \prod_t p(s_t | s_{t-1}) p'(u_t | u_{t-1}) \int p(x_t | s_t) p'(x_t | u_t) \, dx_t \\
= \sum_{s,u} \prod_t p(s_t | s_{t-1}) p'(u_t | u_{t-1}) K_{s,u} \\
= \sum_{s,u} p(s_0) p'(u_0) \psi(s_0, u_0) \prod_t p(s_t | s_{t-1}) p'(u_t | u_{t-1}) \psi(s_t, u_t)
\]

Take advantage of structure in HMMs via Bayesian network

Only compute subkernels for common parents

Evaluate total of \( O\left( T |s| |u| \right) \) subkernels

Form +ve clique potential functions, sum via junction tree algorithm
Hidden Markov Model Product Kernels

Protein Dataset: 480 protein sequences from SCOP dataset
Each ~200 discrete symbols (alphabet ~ 20)
Train 2-state HMM on each sequence (over-fitting)
SVM two class problem: family 2.1.1.4 vs. family 2.1
Training: 120 +ve, 120 -ve Testing: 120 +ve, 120 -ve
Bayesian Network Product Kernels

Probability product over common sample space between any pair of Latent Bayesian Networks:

\[
p(x) = \prod_{i=1}^{D} p(x_i | pa_i)
\]

\[
K(\chi_v, \chi_v') = \sum_s \sum_u p(s) p'(u) \int p(x_v | s) p'(x_v | u) dx_v
\]

Compute subkernels for over all latent parents

\[
K_{pa_i, pa_i'} = \int p(x_i | pa_i) p'(x_i | pa_i') dx_i
\]

Computations grow tractably with enlarged clique size of joint parents. Won’t get loopy if original graphs non-loopy.
Intractable Distributions Product Kernels (Sampling)

What if non-parametric or intractable (loopy) distributions?

**Sampling: approximate**

\[ K(\chi, \chi') = \int p(x) p'(x) \, dx \]

By definition, generative models can:
1) Generate a Sample
2) Compute Likelihood of a Sample

Thus, approximate probability product via sampling:

\[
K(\chi, \chi') = K(p, p') = \frac{\beta}{N} \sum_{x_i \sim p(x)} p'(x_i) + \frac{1-\beta}{N'} \sum_{x_i' \sim p'(x)} p(x_i')
\]

Beta controls how much sampling from each distribution...
Discussion

Use Generative Models & Sets in SVMs and Kernel Machines via Hellinger, Bhattacharyya, Expected Likelihood (Less Kernel Voodoo)

Avoid Approximation, Mercer Kernel, Symmetric, Nonlinear Behavior, and Computable for many distributions:
- Exponential Family, Bernoulli, Multinomial, Gaussian,
- Kernelized Gaussian, Mixture Models, HMMs
- Latent Bayes Nets, Sampling Methods, ...

Future Work: Getting aggregated maximum likelihood solution to influence kernel

\[ K(\chi, \chi') = \int (p)^\rho (p_{ML})^T (p')^\rho \, dx \]