Probabilistic Modelling, Machine Learning, and the Information Revolution

Zoubin Ghahramani

Department of Engineering
University of Cambridge, UK

zoubin@eng.cam.ac.uk
http://learning.eng.cam.ac.uk/zoubin/

MIT CSAIL 2012

An Information Revolution?

- We are in an era of abundant data:
  - **Society:** the web, social networks, mobile networks, government, digital archives
  - **Science:** large-scale scientific experiments, biomedical data, climate data, scientific literature
  - **Business:** e-commerce, electronic trading, advertising, personalisation

- We need tools for modelling, searching, visualising, and understanding large data sets.

Modelling Tools

Our modelling tools should:

- Faithfully represent **uncertainty** in our model structure and parameters and **noise** in our data
- Be automated and **adaptive**
- Exhibit **robustness**
- **Scale well** to large data sets

Probabilistic Modelling

- A model describes data that one could observe from a system
- If we use the mathematics of probability theory to express all forms of uncertainty and noise associated with our model...
- ...then *inverse probability* (i.e. Bayes rule) allows us to infer unknown quantities, adapt our models, make predictions and learn from data.

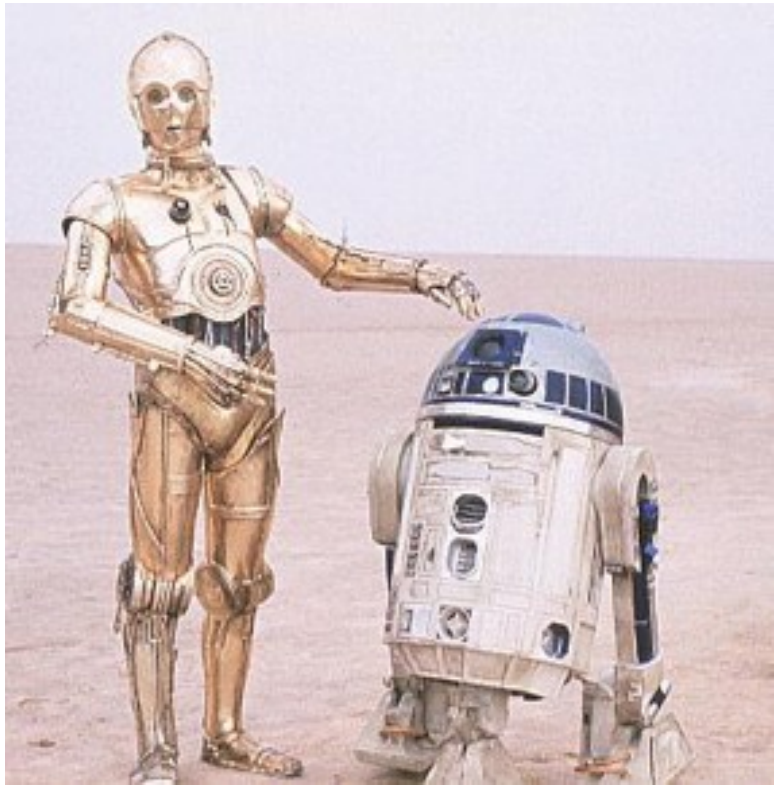
Bayes Rule

$$P(\text{hypothesis}|\text{data}) = \frac{P(\text{data}|\text{hypothesis})P(\text{hypothesis})}{P(\text{data})}$$

Rev'd Thomas Bayes (1702–1761)

- Bayes rule tells us how to do inference about hypotheses from data.
- Learning and prediction can be seen as forms of inference.

How do we build thinking machines?



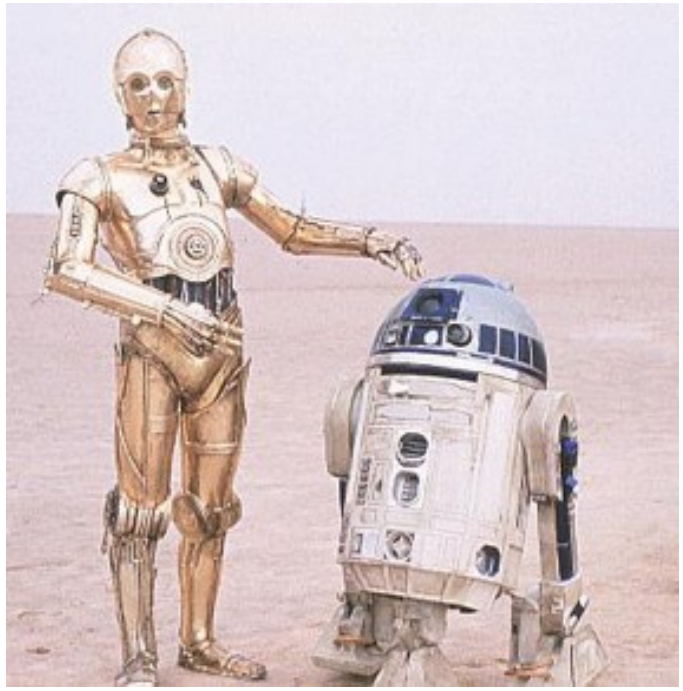
Representing Beliefs in Artificial Intelligence

Consider a robot. In order to behave intelligently the robot should be able to represent beliefs about propositions in the world:

"my charging station is at location (x,y,z)"

"my rangefinder is malfunctioning"

"that stormtrooper is hostile"



We want to represent the **strength** of these beliefs numerically in the brain of the robot, and we want to know what rules (calculus) we should use to manipulate those beliefs.

Representing Beliefs II

Let's use $b(x)$ to represent the strength of belief in (plausibility of) proposition x.

$0 \leq b(x) \leq 1$		
$b(x) = 0$	x	is definitely **not true**
$b(x) = 1$	x	is definitely **true**
$b(x\|y)$		strength of belief that x is true given that we know y is true

Cox Axioms (Desiderata):

- Strengths of belief (degrees of plausibility) are represented by real numbers
- Qualitative correspondence with common sense
- Consistency
  - If a conclusion can be reasoned in more than one way, then every way should lead to the same answer.
  - The robot always takes into account all relevant evidence.
  - Equivalent states of knowledge are represented by equivalent plausibility assignments.

Consequence: Belief functions (e.g. $b(x)$, $b(x|y)$, $b(x, y)$) must satisfy the rules of probability theory, including Bayes rule.

(Cox 1946; Jaynes, 1996; van Horn, 2003)

The Dutch Book Theorem

Assume you are willing to accept bets with odds proportional to the strength of your beliefs. That is, $b(x) = 0.9$ implies that you will accept a bet:

$$\begin{cases} x & \text{is true} & \text{win} & \geq \$1 \\ x & \text{is false} & \text{lose} & \$9 \end{cases}$$

Then, unless your beliefs satisfy the rules of probability theory, including Bayes rule, there exists a set of simultaneous bets (called a "Dutch Book") which you are willing to accept, and for which **you are guaranteed to lose money, no matter what the outcome**.

The only way to guard against Dutch Books to to ensure that your beliefs are coherent: i.e. satisfy the rules of probability.

Bayesian Machine Learning

Everything follows from two simple rules:

Sum rule: $P(x) = \sum_y P(x, y)$

Product rule: $P(x, y) = P(x)P(y|x)$

$$P(\theta|\mathcal{D}, m) = \frac{P(\mathcal{D}|\theta, m)P(\theta|m)}{P(\mathcal{D}|m)}$$

$P(\mathcal{D}\|\theta, m)$	likelihood of parameters θ in model m
$P(\theta\|m)$	prior probability of θ
$P(\theta\|\mathcal{D}, m)$	posterior of θ given data $\mathcal{D}$

Prediction:

$$P(x|\mathcal{D}, m) = \int P(x|\theta, \mathcal{D}, m)P(\theta|\mathcal{D}, m)d\theta$$

Model Comparison:

$$P(m|\mathcal{D}) = \frac{P(\mathcal{D}|m)P(m)}{P(\mathcal{D})}$$

$$P(\mathcal{D}|m) = \int P(\mathcal{D}|\theta, m)P(\theta|m)\, d\theta$$

Modeling vs toolbox views of Machine Learning

- **Machine Learning seeks to learn models of data**: define a space of possible models; learn the parameters and structure of the models from data; make predictions and decisions

- **Machine Learning is a toolbox of methods for processing data**: feed the data into one of many possible methods; choose methods that have good theoretical or empirical performance; make predictions and decisions

Bayesian Nonparametrics

Why...

- **Why Bayesian?**

  Simplicity (of the framework)

- **Why nonparametrics?**

  Complexity (of real world phenomena)

Parametric vs Nonparametric Models

- *Parametric models* assume some finite set of parameters θ. Given the parameters, future predictions, x, are independent of the observed data, $\mathcal{D}$:

$$P(x|\theta, \mathcal{D}) = P(x|\theta)$$

  therefore θ capture everything there is to know about the data.

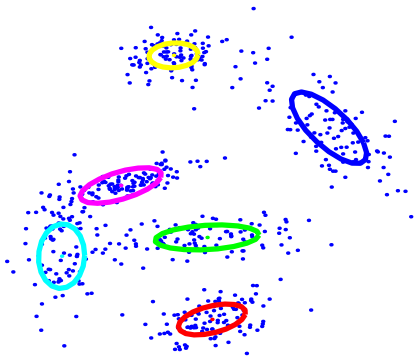
- So the complexity of the model is bounded even if the amount of data is unbounded. This makes them not very flexible.

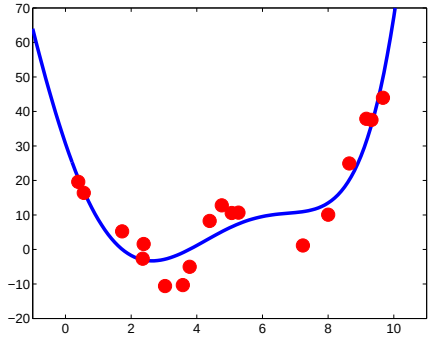
- *Non-parametric models* assume that the data distribution cannot be defined in terms of such a finite set of parameters. But they can often be defined by assuming an *infinite dimensional* θ. Usually we think of θ as a *function*.

- The amount of information that θ can capture about the data $\mathcal{D}$ can grow as the amount of data grows. This makes them more flexible.

Why nonparametrics?

- flexibility
- better predictive performance
- more realistic





All successful methods in machine learning are essentially nonparametric[1]:

- kernel methods / SVM / GP
- deep networks / large neural networks
- k-nearest neighbors, ...

[1] or highly scalable!

Overview of nonparametric models and uses

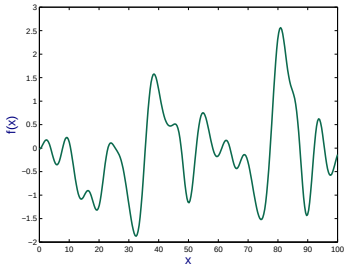
Bayesian nonparametrics has many uses.

Some modelling goals and *examples* of associated nonparametric Bayesian models:

Modelling goal	**Example process**
Distributions on functions	Gaussian process
Distributions on distributions	Dirichlet process
	Polya Tree
Clustering	Chinese restaurant process
	Pitman-Yor process
Hierarchical clustering	Dirichlet diffusion tree
	Kingman's coalescent
Sparse binary matrices	Indian buffet processes
Survival analysis	Beta processes
Distributions on measures	Completely random measures
...	...

Gaussian and Dirichlet Processes

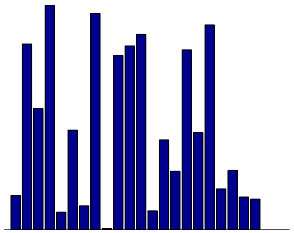
- Gaussian processes define a distribution on functions



$$f \sim \mathrm{GP}(\cdot|\mu, c)$$

where μ is the mean function and c is the covariance function.
We can think of GPs as "infinite-dimensional" Gaussians

- Dirichlet processes define a distribution on distributions



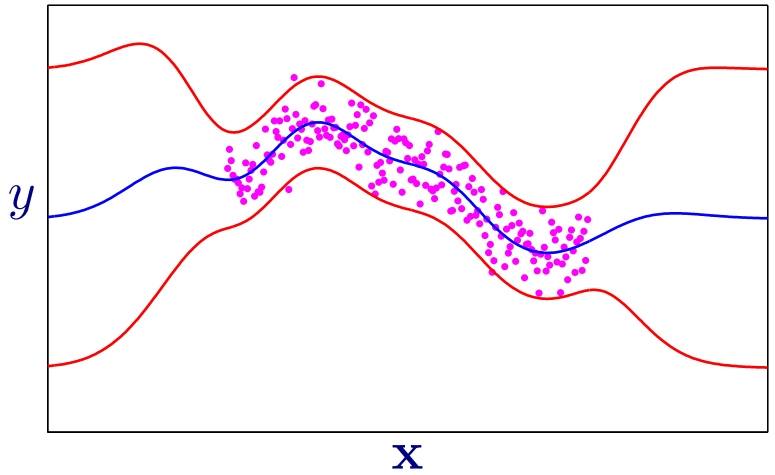
$$G \sim \mathrm{DP}(\cdot|G_0, \alpha)$$

where $\alpha > 0$ is a scaling parameter, and G_0 is the base measure.
We can think of DPs as "infinite-dimensional" Dirichlet distributions.

Note that both f and G are infinite dimensional objects.

Nonlinear regression and Gaussian processes

Consider the problem of nonlinear regression:
You want to learn a function f with error bars from data $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$



A Gaussian process defines a distribution over functions $p(f)$ which can be used for Bayesian regression:

$$p(f|\mathcal{D}) = \frac{p(f)p(\mathcal{D}|f)}{p(\mathcal{D})}$$

Let $\mathbf{f} = (f(x_1), f(x_2), \ldots, f(x_n))$ be an n-dimensional vector of function values evaluated at n points $x_i \in \mathcal{X}$. Note, $\mathbf{f}$ is a random variable.

Definition: $p(f)$ is a Gaussian process if for *any* finite subset $\{x_1, \ldots, x_n\} \subset \mathcal{X}$, the marginal distribution over that subset $p(\mathbf{f})$ is multivariate Gaussian.

Gaussian Processes and SVMs

Support Vector Machines and Gaussian Processes

We can write the SVM loss as: $\min_{\mathbf{f}} \ \frac{1}{2}\mathbf{f}^\top \mathbf{K}^{-1}\mathbf{f} + C\sum_i (1 - y_i f_i)_+$

We can write the negative log of a GP likelihood as: $\frac{1}{2}\mathbf{f}^\top \mathbf{K}^{-1}\mathbf{f} - \sum_i \ln p(y_i|f_i) + c$

Equivalent? No.

With Gaussian processes we:

- Handle **uncertainty** in unknown function $\mathbf{f}$ by averaging, not minimization.
- Compute $p(y = +1|\mathbf{x}) \neq p(y = +1|\hat{\mathbf{f}}, \mathbf{x})$.
- Can **learn the kernel parameters** automatically from data, no matter how flexible we wish to make the kernel.
- Can **learn the regularization parameter** C without cross-validation.
- Can incorporate **interpretable** noise models and priors over functions, and can sample from prior to get intuitions about the model assumptions.
- We can combine **automatic feature selection** with learning using ARD.

Easy to use Matlab code: `http://www.gaussianprocess.org/gpml/code/`

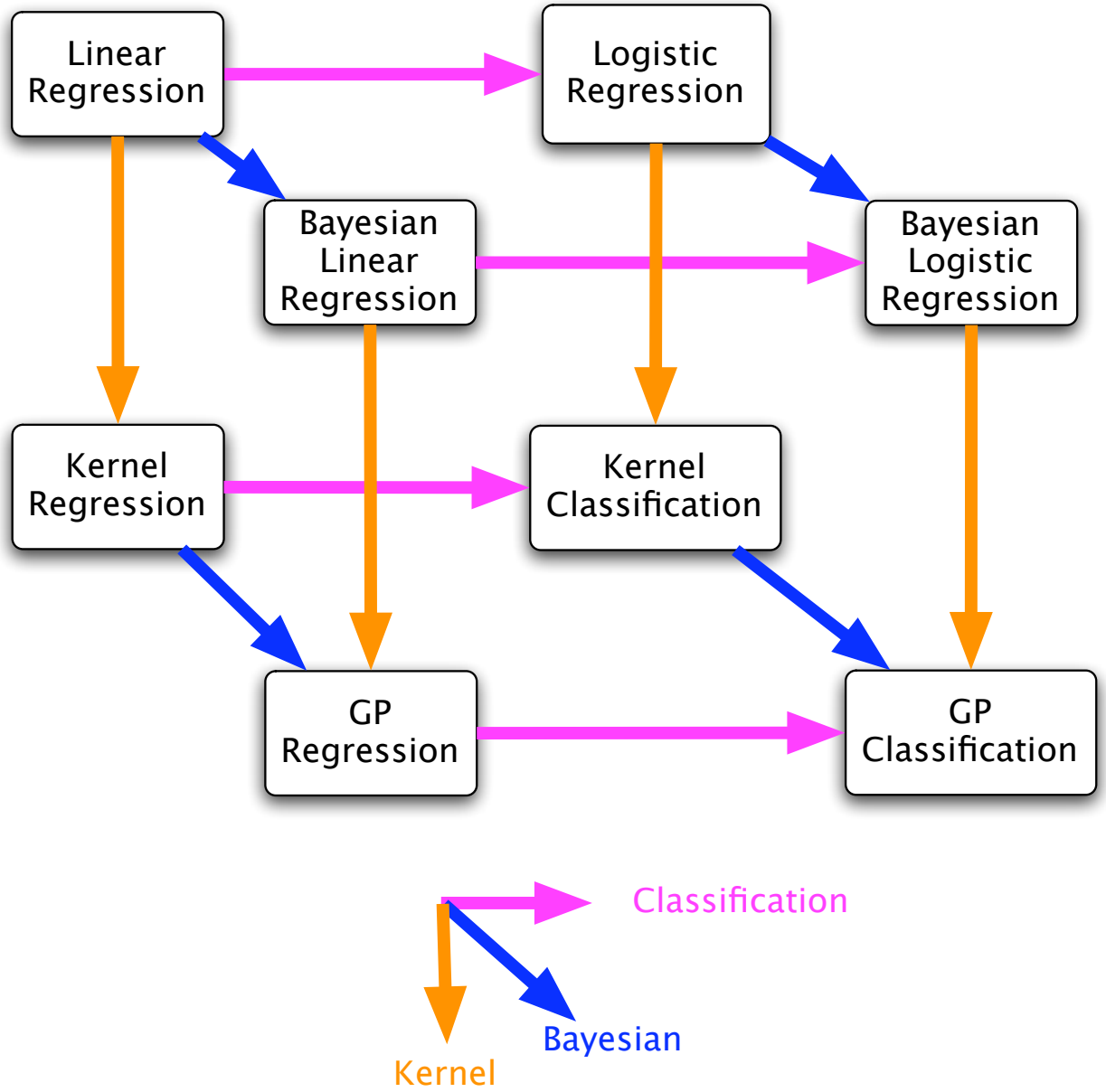
Some Comparisons

Table 1: Test errors and predictive accuracy (smaller is better) for the GP classifier, the support vector machine, the informative vector machine, and the sparse pseudo-input GP classifier.

Data set			**GPC**		**SVM**		**IVM**			**SPGPC**		
name	train:test	dim	err	nlp	err	#sv	err	nlp	M	err	nlp	M
synth	250:1000	2	0.097	*0.227*	0.098	98	0.096	*0.235*	150	**0.087**	*0.234*	4
crabs	80:120	5	0.039	*0.096*	0.168	67	0.066	*0.134*	60	**0.043**	*0.105*	10
banana	400:4900	2	0.105	*0.237*	0.106	151	**0.105**	*0.242*	200	0.107	*0.261*	20
breast-cancer	200:77	9	0.288	*0.558*	**0.277**	122	0.307	*0.691*	120	0.281	*0.557*	2
diabetes	468:300	8	0.231	*0.475*	**0.226**	271	0.230	*0.486*	400	0.230	*0.485*	2
flare-solar	666:400	9	0.346	*0.570*	**0.331**	556	0.340	*0.628*	550	0.338	*0.569*	3
german	700:300	20	0.230	*0.482*	0.247	461	0.290	*0.658*	450	**0.236**	*0.491*	4
heart	170:100	13	0.178	*0.423*	**0.166**	92	0.203	*0.455*	120	0.172	*0.414*	2
image	1300:1010	18	0.027	*0.078*	0.040	462	**0.028**	*0.082*	400	0.031	*0.087*	200
ringnorm	400:7000	20	0.016	*0.071*	0.016	157	0.016	*0.101*	100	**0.014**	*0.089*	2
splice	1000:2175	60	0.115	*0.281*	**0.102**	698	0.225	*0.403*	700	0.126	*0.306*	200
thyroid	140:75	5	0.043	*0.093*	0.056	61	0.041	*0.120*	40	**0.037**	*0.128*	6
titanic	150:2051	3	0.221	*0.514*	**0.223**	118	0.242	*0.578*	100	0.231	*0.520*	2
twonorm	400:7000	20	0.031	*0.085*	0.027	220	0.031	*0.085*	300	**0.026**	*0.086*	2
waveform	400:4600	21	0.100	*0.229*	0.107	148	0.100	*0.232*	250	**0.099**	*0.228*	10

From (Naish-Guzman and Holden, 2008), using exactly same kernels.

A picture


Linear Regression
Logistic Regression
Bayesian Linear Regression
Bayesian Logistic Regression
Kernel Regression
Kernel Classification
GP Regression
GP Classification
Classification
Bayesian
Kernel

Outline

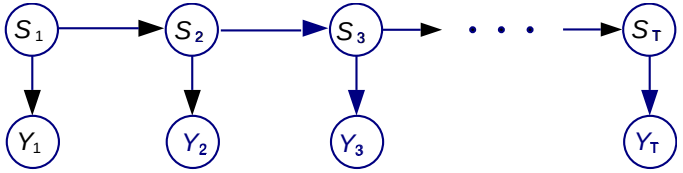
Bayesian nonparametrics applied to models of other structured objects:

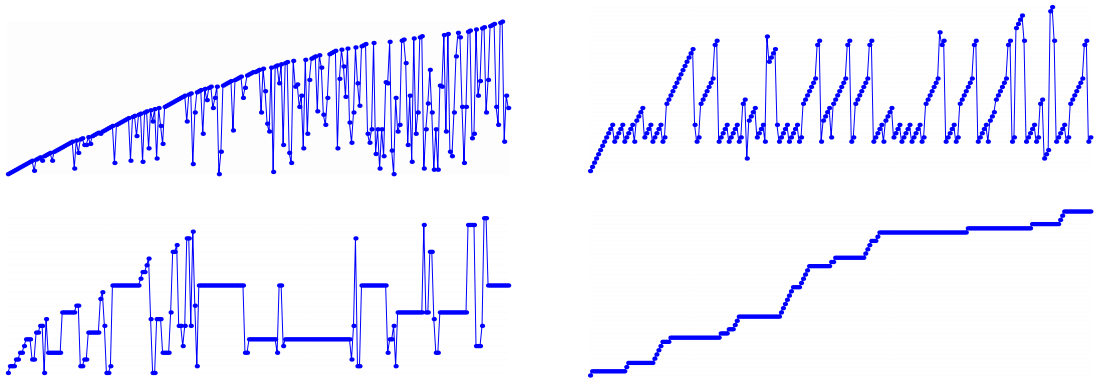
- Time Series
- Sparse Matrices
- Deep Sparse Graphical Models
- Hierarchies
- Covariances
- Network Structured Regression

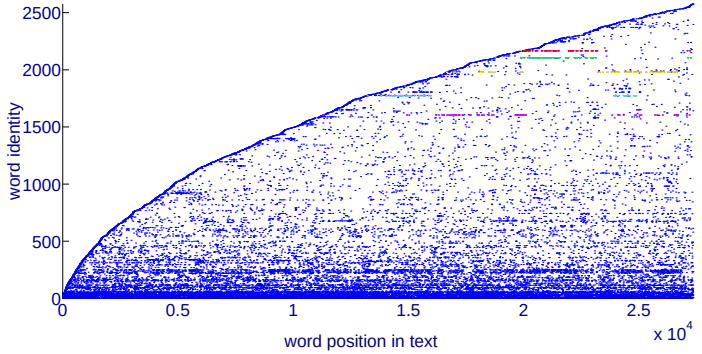
Infinite hidden Markov models (iHMMs)

Hidden Markov models (HMMs) are widely used sequence models for speech recognition, bioinformatics, text modelling, video monitoring, etc. HMMs can be thought of as *time-dependent mixture models*.

In an HMM with K states, the transition matrix has $K \times K$ elements. Let $K \to \infty$.

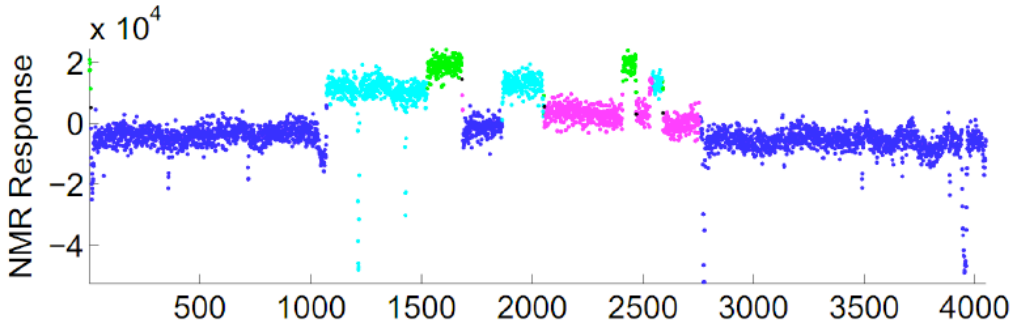


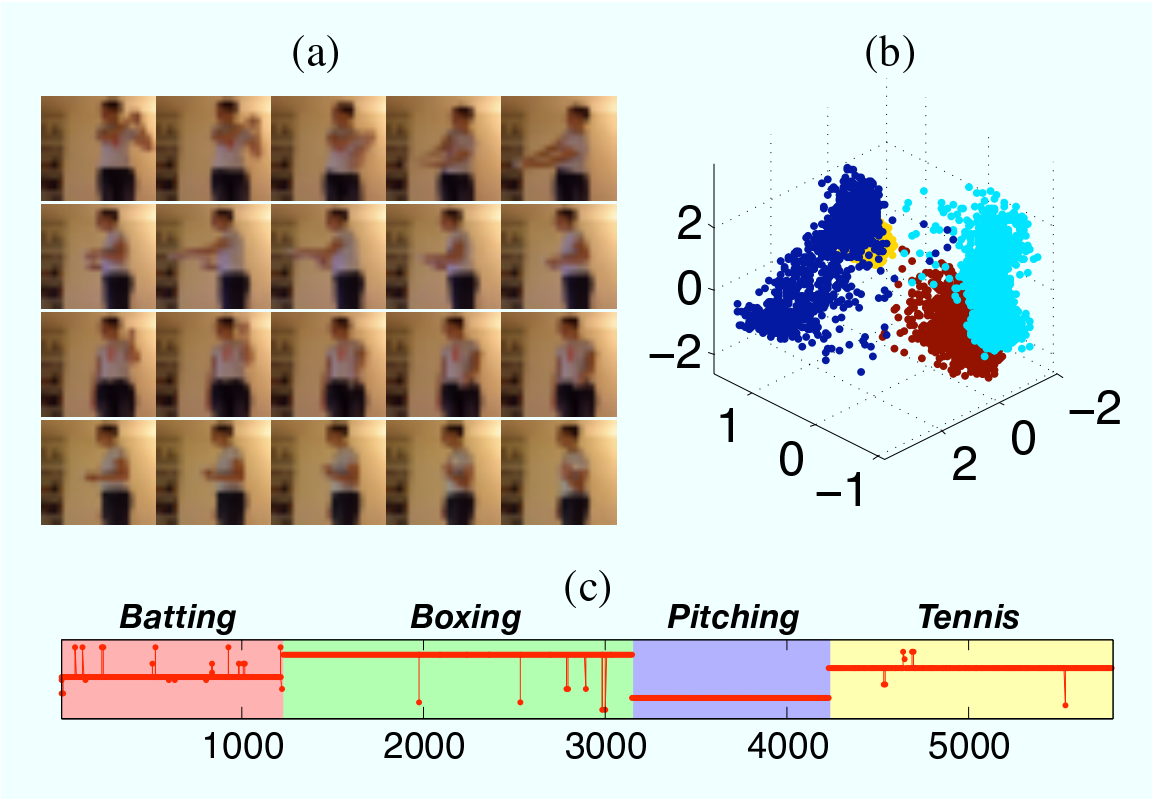




- Introduced in (Beal, Ghahramani and Rasmussen, 2002).
- Teh, Jordan, Beal and Blei (2005) showed that iHMMs can be derived from hierarchical Dirichlet processes, and provided a more efficient Gibbs sampler.
- We have recently derived a much more efficient sampler based on Dynamic Programming (Van Gael, Saatci, Teh, and Ghahramani, 2008). `http://mloss.org/software/view/205/`
- And we have parallel (.NET) and distributed (Hadoop) implementations (Bratieres, Van Gael, Vlachos and Ghahramani, 2010).

Infinite HMM: Changepoint detection and video segmentation

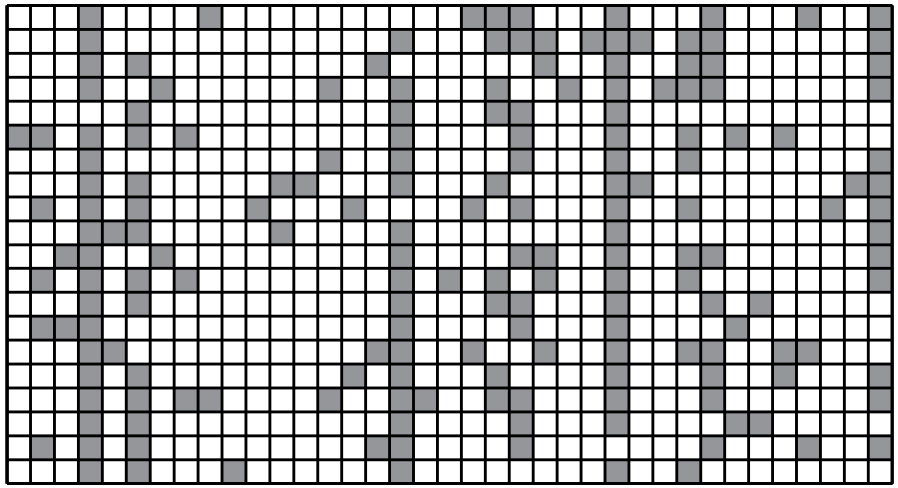




(w/ Tom Stepleton, 2009)

Sparse Matrices

From finite to infinite sparse binary matrices



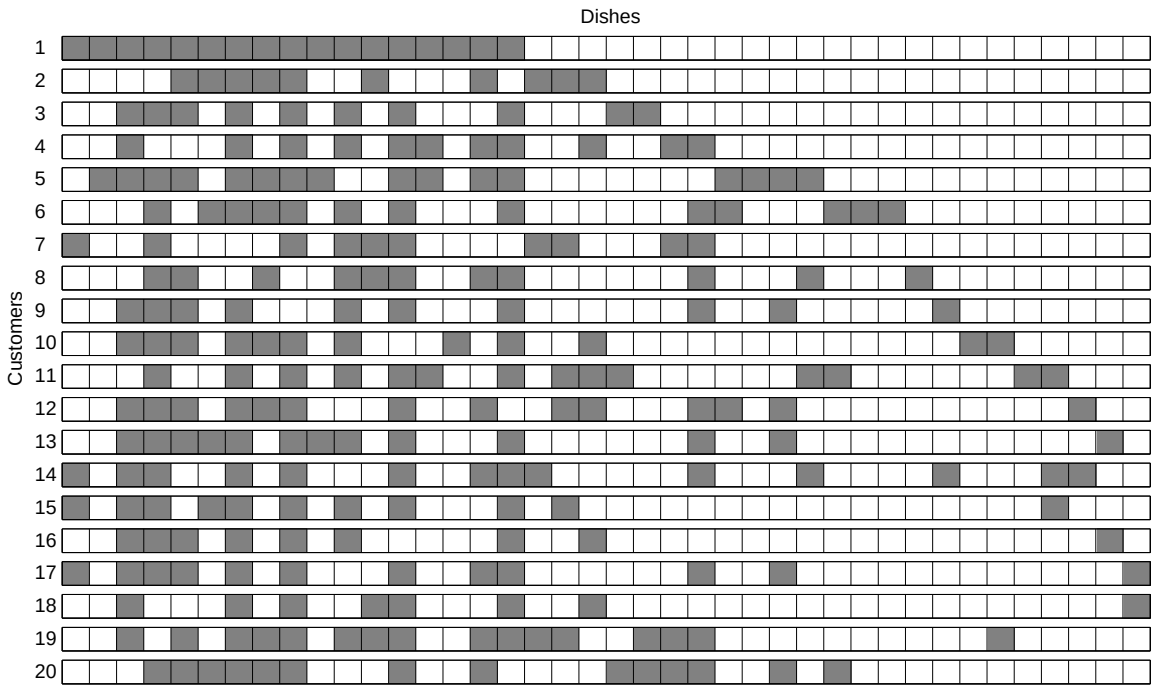
$z_{nk} = 1$ means object n has feature k:

$$z_{nk} \sim \text{Bernoulli}(\theta_k)$$

$$\theta_k \sim \text{Beta}(\alpha/K, 1)$$

- Note that $P(z_{nk} = 1|\alpha) = E(\theta_k) = \frac{\alpha/K}{\alpha/K+1}$, so as K grows larger the matrix gets sparser.
- So if $\mathbf{Z}$ is $N \times K$, the expected number of nonzero entries is $N\alpha/(1+\alpha/K) < N\alpha$.
- Even in the $K \to \infty$ limit, the matrix is expected to have a finite number of non-zero entries.
- $K \to \infty$ results in an Indian buffet process (IBP)

Indian buffet process



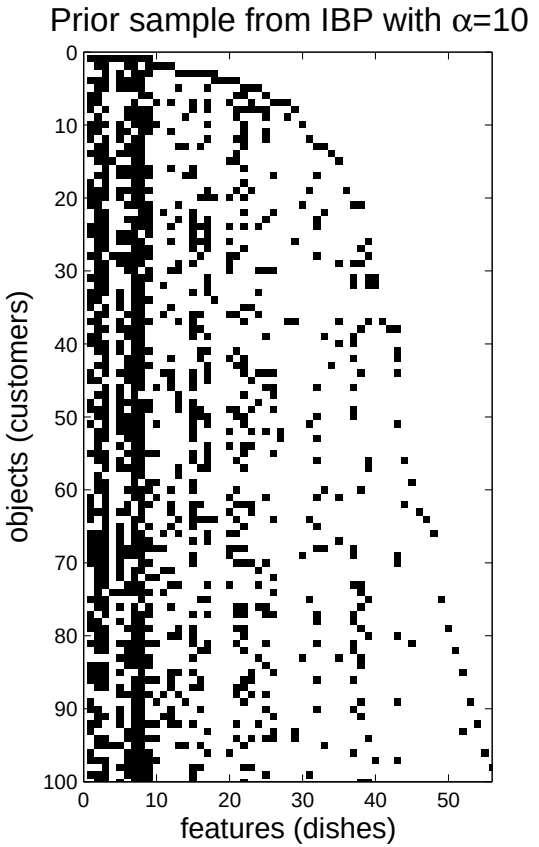
"Many Indian restaurants in London offer lunchtime buffets with an apparently infinite number of dishes"

- First customer starts at the left of the buffet, and takes a serving from each dish, stopping after a Poisson(α) number of dishes as his plate becomes overburdened.
- The n^{th} customer moves along the buffet, sampling dishes in proportion to their popularity, serving himself dish k with probability m_k/n, and trying a Poisson(α/n) number of new dishes.
- The customer-dish matrix, $\mathbf{Z}$, is a draw from the IBP.

(w/ Tom Griffiths 2006; 2011)

Properties of the Indian buffet process

$$P([\mathbf{Z}]|\alpha) = \exp\left\{-\alpha H_N\right\} \frac{\alpha^{K_+}}{\prod_{h>0} K_h!} \prod_{k \leq K_+} \frac{(N-m_k)!(m_k-1)!}{N!}$$



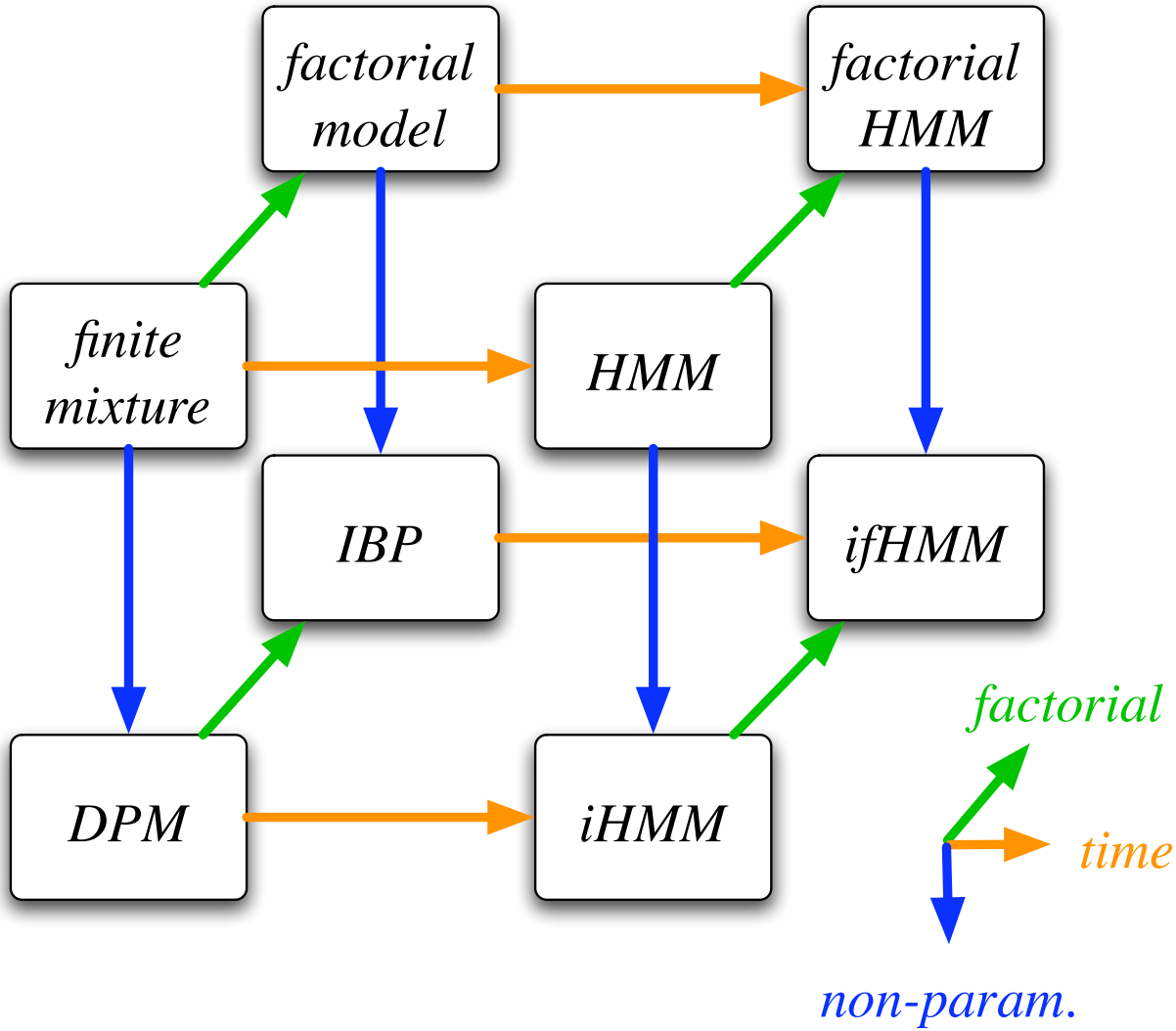
Shown in (Griffiths and Ghahramani 2006, 2011):

- It is infinitely exchangeable.
- The number of ones in each row is Poisson(α)
- The expected total number of ones is αN.
- The number of nonzero columns grows as $O(\alpha \log N)$.

Additional properties:

- Has a stick-breaking representation (Teh, et al 2007)
- Has as its de Finetti mixing distribution the Beta process (Thibaux and Jordan 2007)
- More flexible two and three parameter versions exist (w/ Griffiths & Sollich 2007; Teh and Görür 2010)

The Big Picture:
Relations between some models



Modelling Data with Indian Buffet Processes

Latent variable model: let $\mathbf{X}$ be the $N \times D$ matrix of observed data, and $\mathbf{Z}$ be the $N \times K$ matrix of binary latent features

$$P(\mathbf{X}, \mathbf{Z}|\alpha) = P(\mathbf{X}|\mathbf{Z})P(\mathbf{Z}|\alpha)$$

By combining the IBP with different likelihood functions we can get different kinds of models:

- Models for graph structures (w/ Wood, Griffiths, 2006; w/ Adams and Wallach, 2010)
- Models for protein complexes (w/ Chu, Wild, 2006)
- Models for choice behaviour (Görür & Rasmussen, 2006)
- Models for users in collaborative filtering (w/ Meeds, Roweis, Neal, 2007)
- Sparse latent trait, pPCA and ICA models (w/ Knowles, 2007, 2011)
- Models for overlapping clusters (w/ Heller, 2007)

Nonparametric Binary Matrix Factorization

genes $\times$ patients

users $\times$ movies

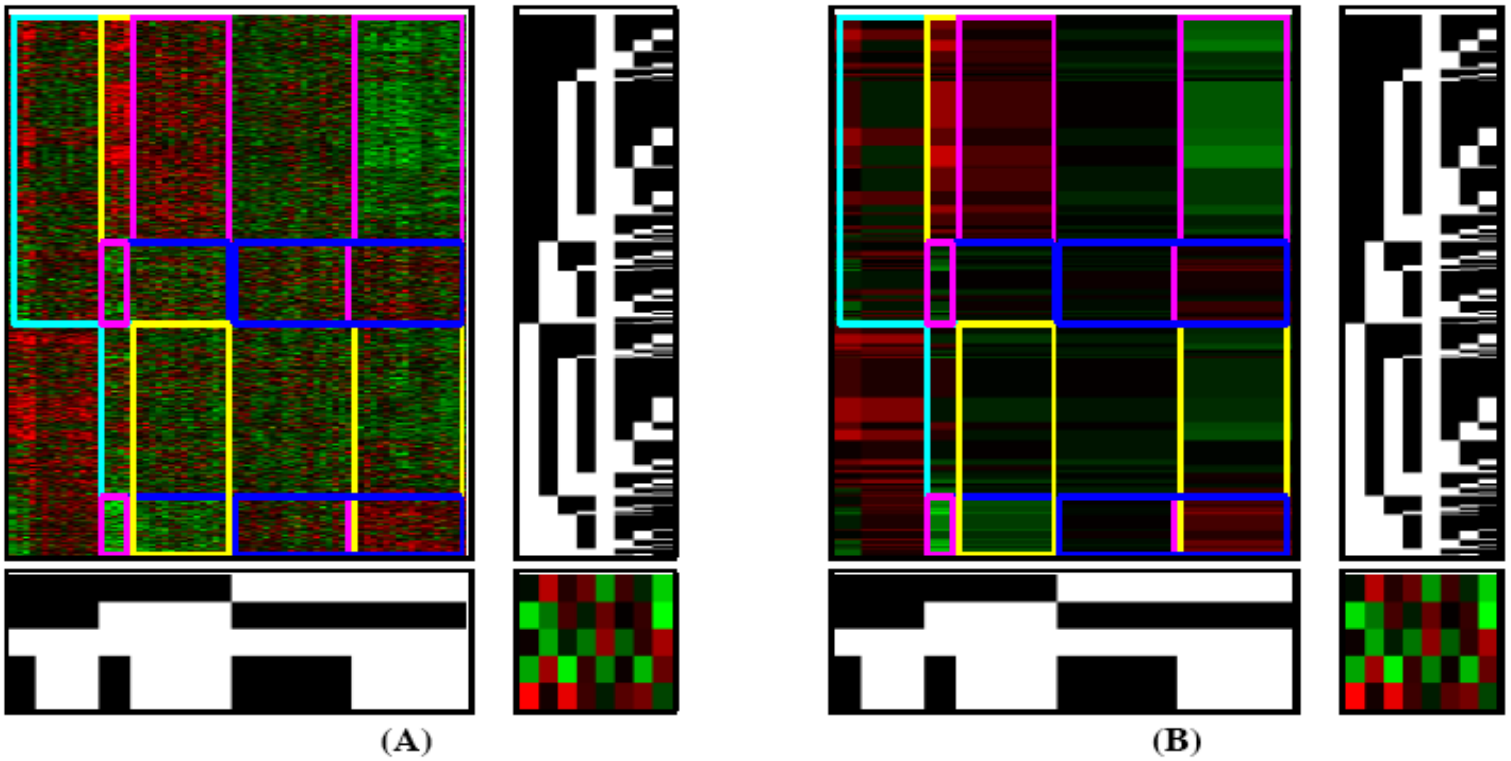
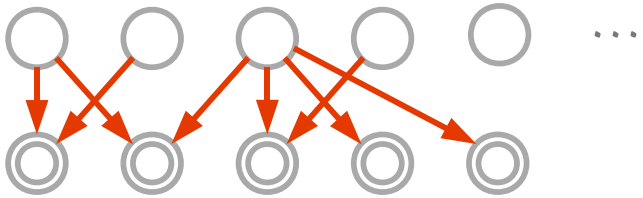
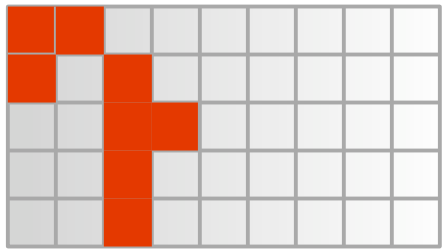


Figure 5: Gene expression results. **(A)** The top-left is $\mathbf{X}$ sorted according to contiguous features in the final $\mathbf{U}$ and $\mathbf{V}$ in the Markov chain. The bottom-left is $\mathbf{V}^\top$ and the top-right is $\mathbf{U}$. The bottom-right is $\mathbf{W}$. **(B)** The same as **(A)**, but the expected value of $\mathbf{X}$, $\hat{\mathbf{X}} = \mathbf{U}\mathbf{W}\mathbf{V}^\top$. We have hilighted regions that have both u_{ik} and v_{jl} on. For clarity, we have only shown the (at most) two largest contiguous regions for each feature pair.

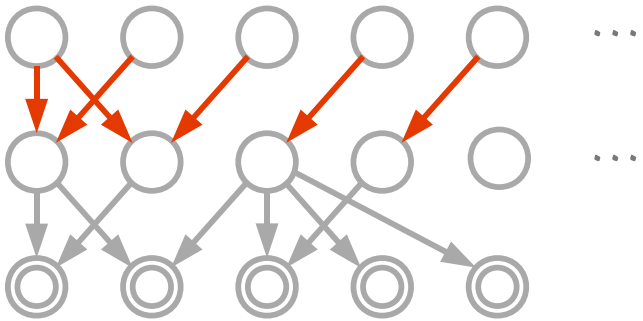
Meeds et al (2007) Modeling Dyadic Data with Binary Latent Factors.

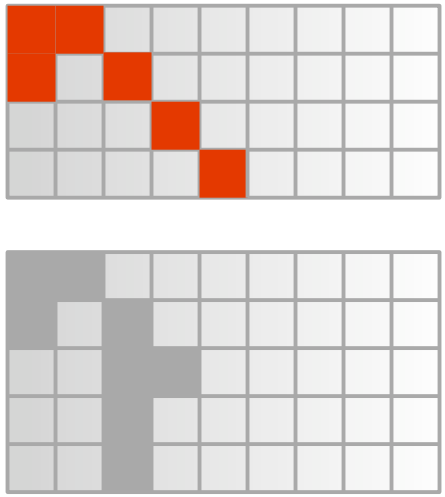
Learning Structure of Deep Sparse Graphical Models



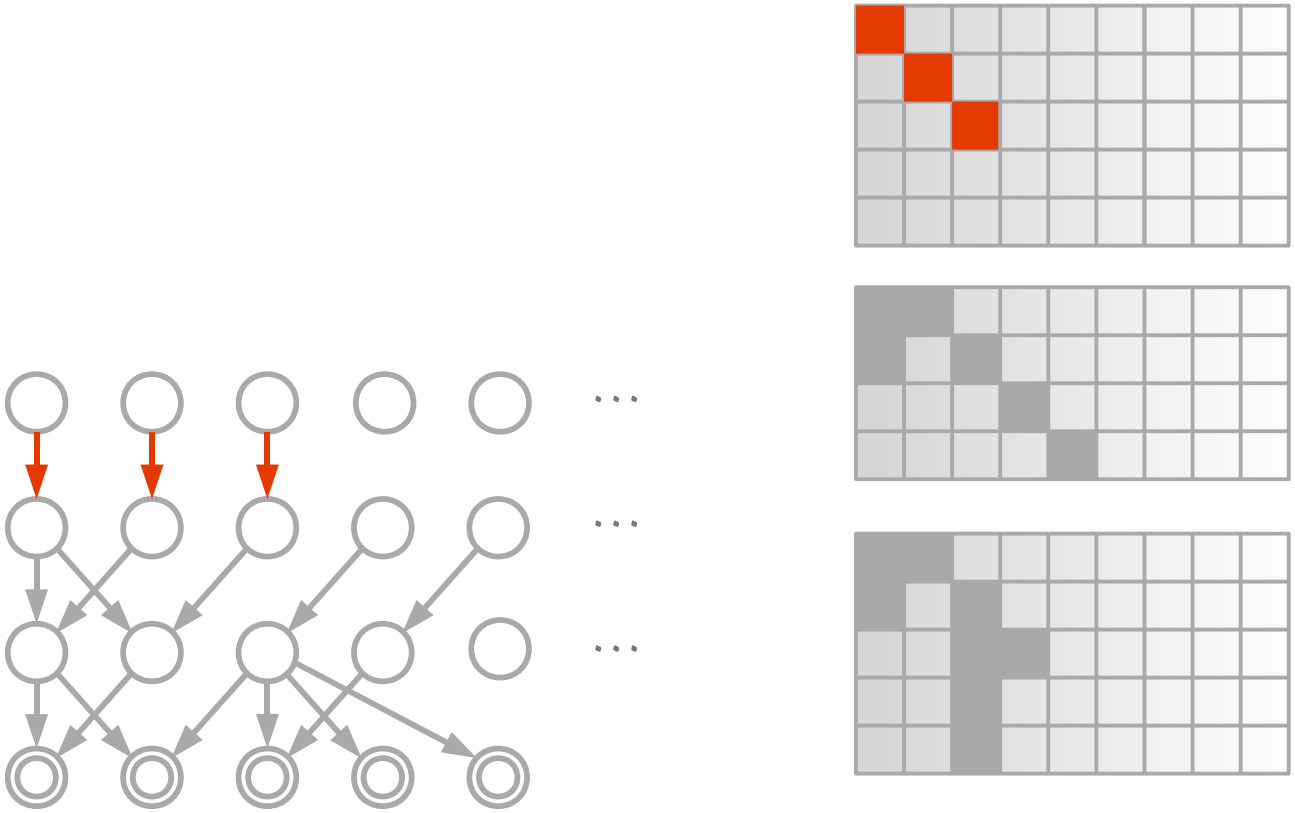


Learning Structure of Deep Sparse Graphical Models

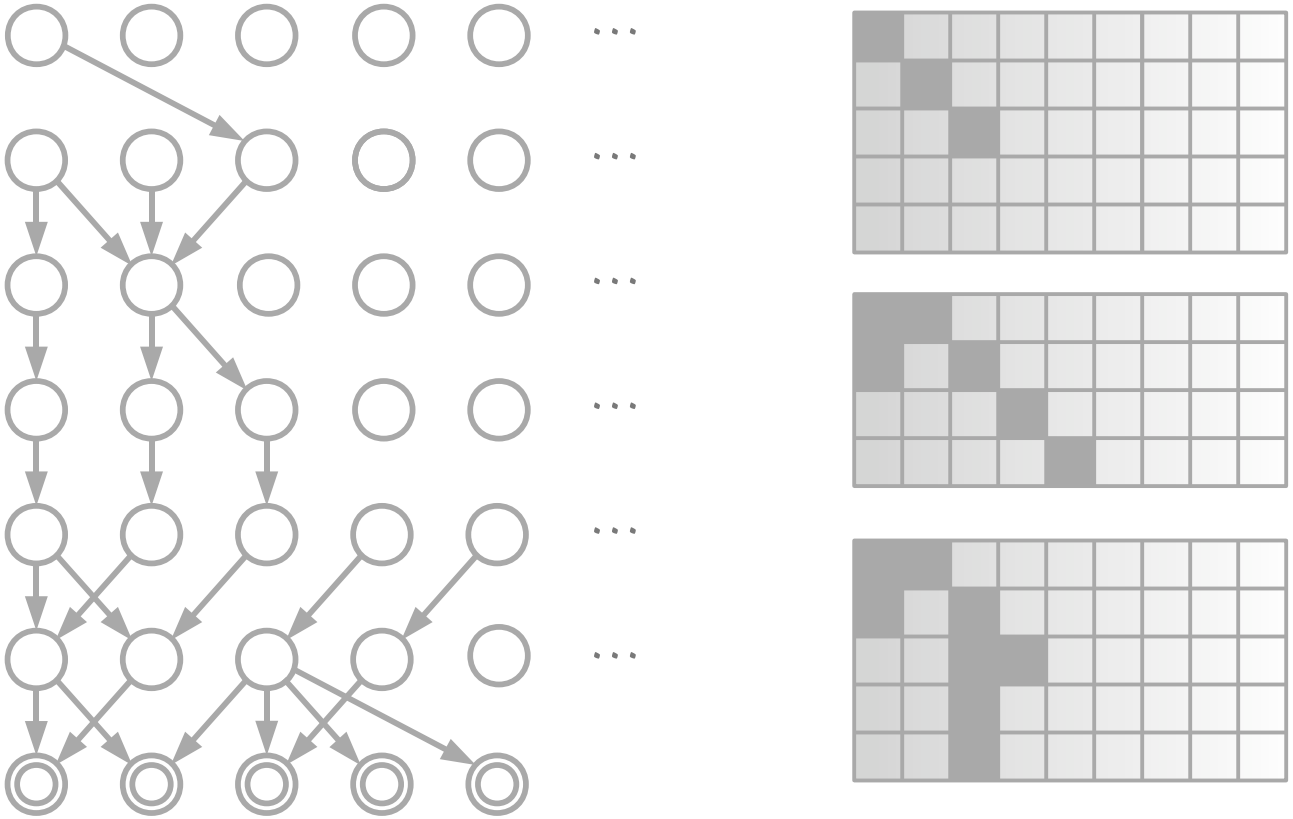




Learning Structure of Deep Sparse Graphical Models



Learning Structure of Deep Sparse Graphical Models

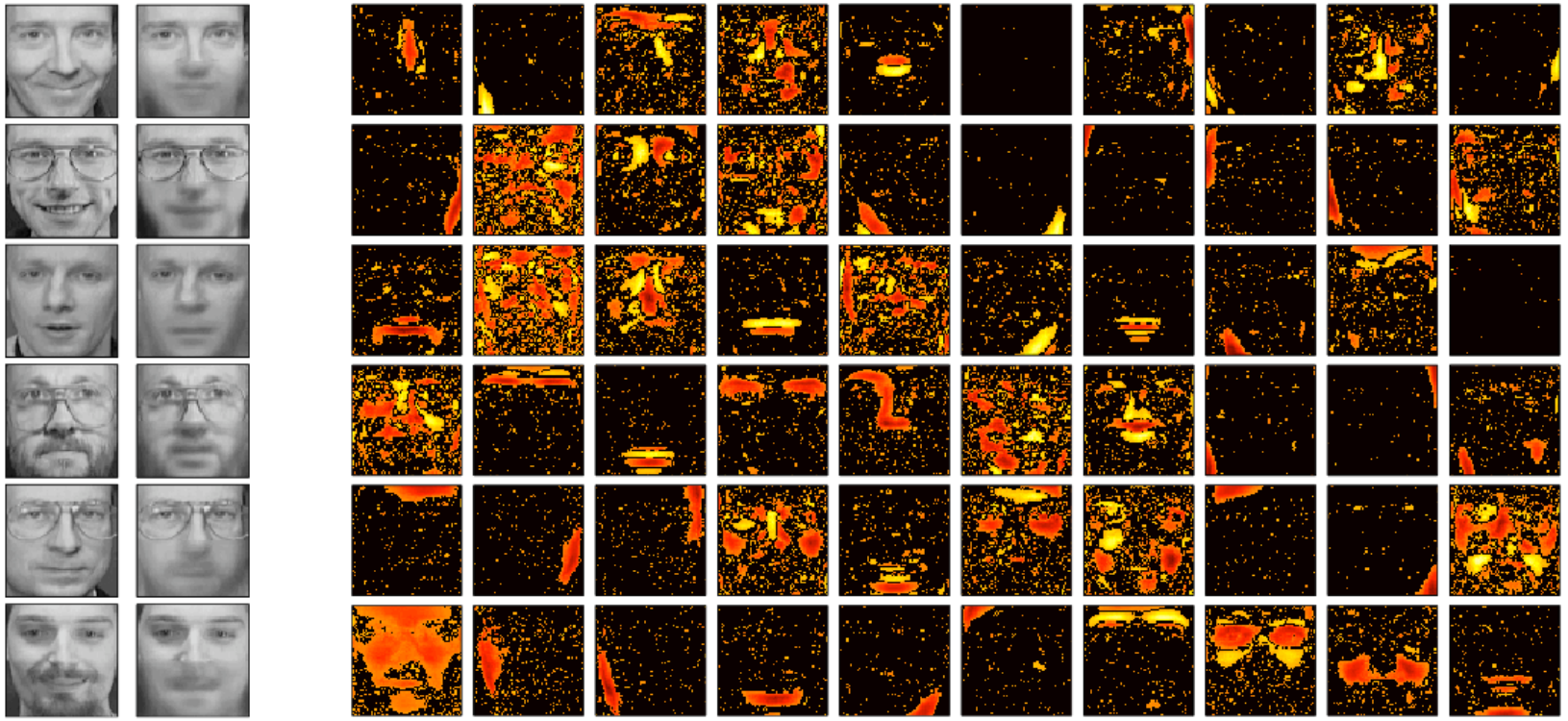


(w/ Ryan P. Adams, Hanna Wallach, 2010)

Learning Structure of Deep Sparse Graphical Models

Olivetti Faces: 350 + 50 images of 40 faces (64×64)
Inferred: 3 hidden layers, 70 units per layer.

Reconstructions and Features:

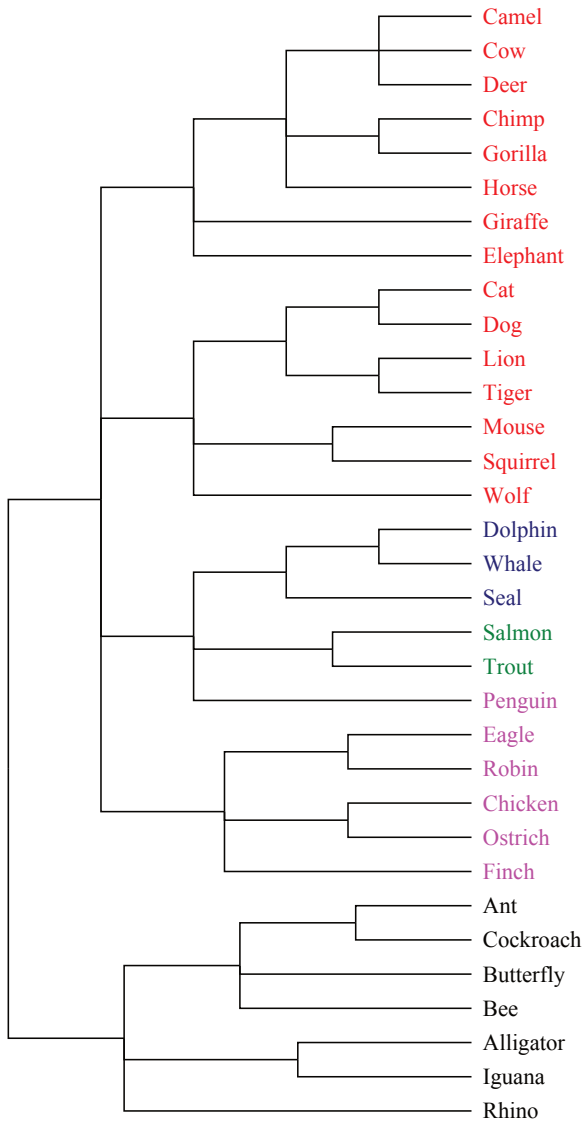


Learning Structure of Deep Sparse Graphical Models

Fantasies and Activations:

Hierarchies

- true hierarchies
- parameter tying
- visualisation and interpretability



Dirichlet Diffusion Trees (DDT)

(Neal, 2001)

In a DPM, parameters of one mixture component are independent of other components – this lack of structure is potentially undesirable.

A DDT is a generalization of DPMs with hierarchical structure between components.

To generate from a DDT, we will consider data points $x_1, x_2, \ldots$ taking a random walk according to a Brownian motion Gaussian diffusion process.

- $x_1(t) \sim$ Gaussian diffusion process starting at origin ($x_1(0) = 0$) for unit time.
- $x_2(t)$ also starts at the origin and follows x_1 but diverges at some time τ, at which point the path followed by x_2 becomes independent of x_1's path.
- $a(t)$ is a divergence or hazard function, e.g. $a(t) = 1/(1-t)$. For small dt:

$$P(x_i \text{ diverges at time } \tau \in (t, t+dt)) = \frac{a(t)dt}{m}$$

  where m is the number of previous points that have followed this path.
- If x_i reaches a branch point between two paths, it picks a branch in proportion to the number of points that have followed that path.

Dirichlet Diffusion Trees (DDT)

Generating from a DDT:

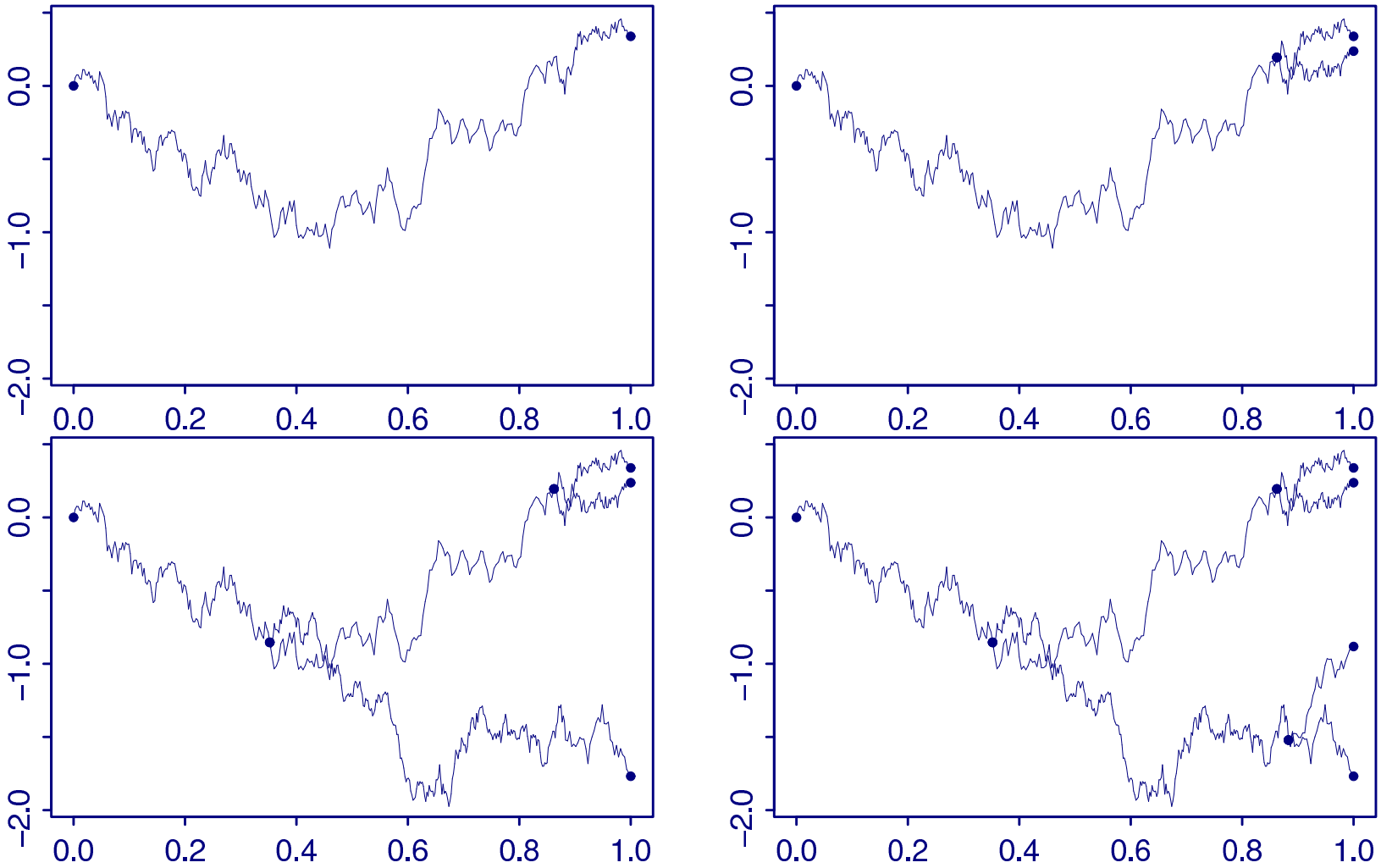


Figure from (Neal 2001)

Pitman-Yor Diffusion Trees

Generalises a DDT, but at a branch point, the probability of following each branch is given by a Pitman-Yor process:

$$P(\text{following branch } k) = \frac{b_k - \alpha}{m + \theta},$$
$$P(\text{diverging}) = \frac{\theta + \alpha K}{m + \theta},$$

to maintain exchangeability the probability of diverging also has to change.

- naturally extends DDTs ($\theta = \alpha = 0$) to arbitrary non-binary branching
- infinitely exchangeable over data
- prior over structure is the most general Markovian consistent and exchangeable distribution over trees (McCullagh et al 2008)

(w/ Knowles 2011)

Pitman-Yor Diffusion Tree: Results

$N_{\text{train}} = 200, N_{\text{test}} = 28, D = 10$ Adams et al. (2008)

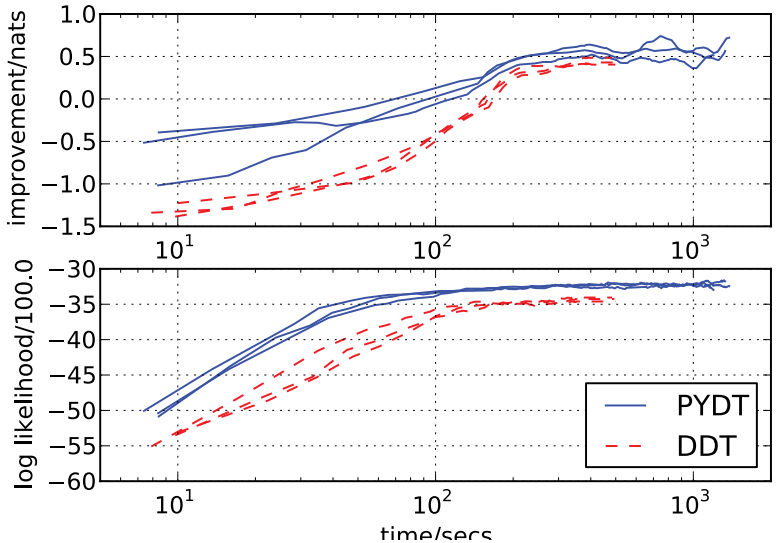
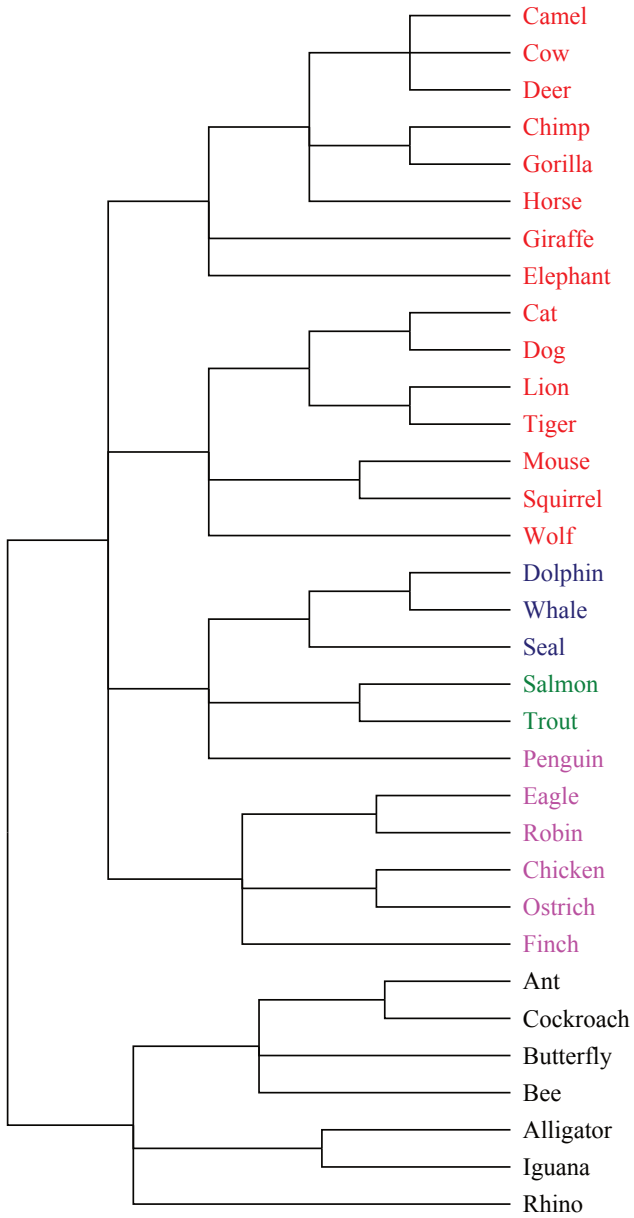


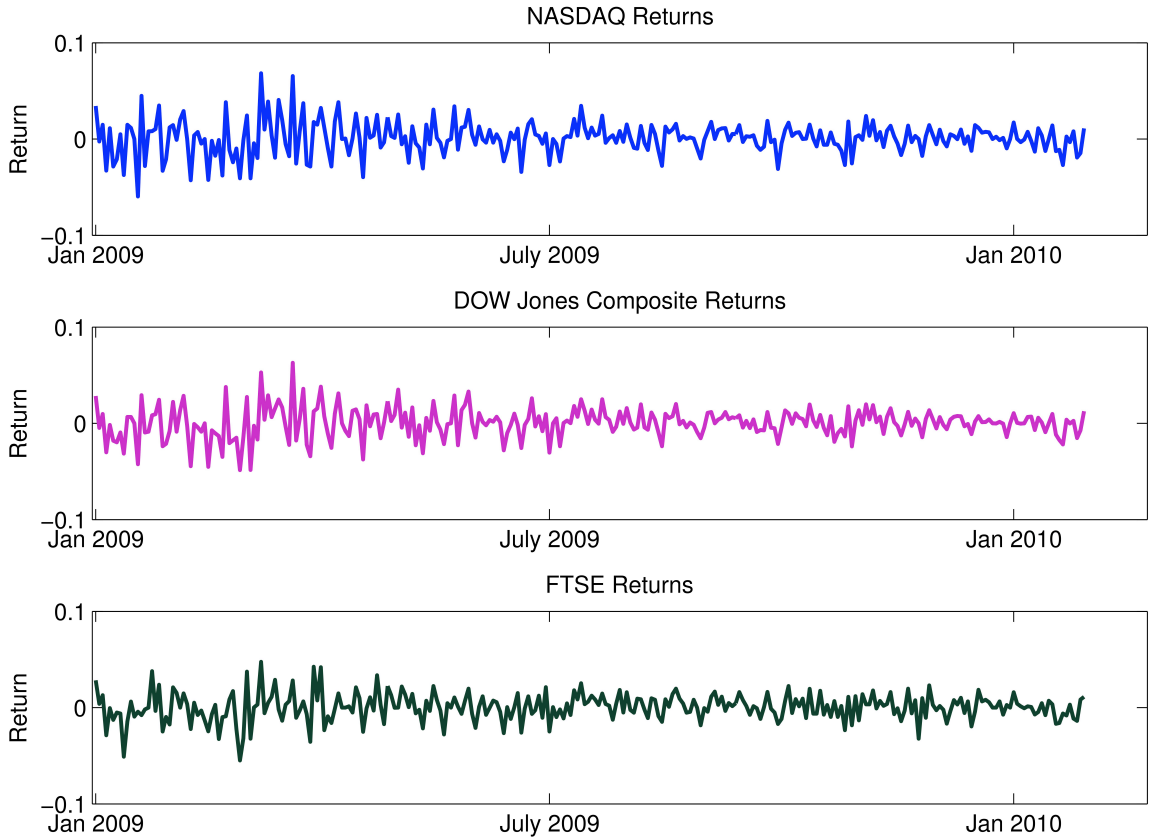
Figure: Density modeling of the $D = 10, N = 200$ macaque skull measurement dataset of Adams et al. (2008). *Top*: Improvement in test predictive likelihood compared to a kernel density estimate. *Bottom*: Marginal likelihood of current tree. The shared x-axis is computation time in seconds.



Covariance Matrices

Covariance Matrices

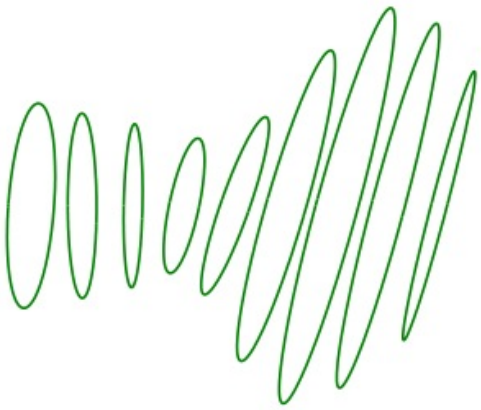
Consider the problem of modelling a covariance matrix Σ that can change as a function of time, $\Sigma(t)$, or other input variables $\Sigma(x)$. This is a widely studied problem in *Econometrics*.



Models commonly used are multivariate GARCH, and multivariate stochastic volatility models, but these only depend on t, and generally don't scale well.

Generalised Wishart Processes for Covariance modelling

Modelling time- and spatially-varying covariance matrices. Note that covariance matrices have to be symmetric positive (semi-)definite.



If $\mathbf{u}_i \sim \mathcal{N}$, then $\Sigma = \sum_{i=1}^{\nu} \mathbf{u}_i \mathbf{u}_i^\top$ is s.p.d. and has a Wishart distribution.

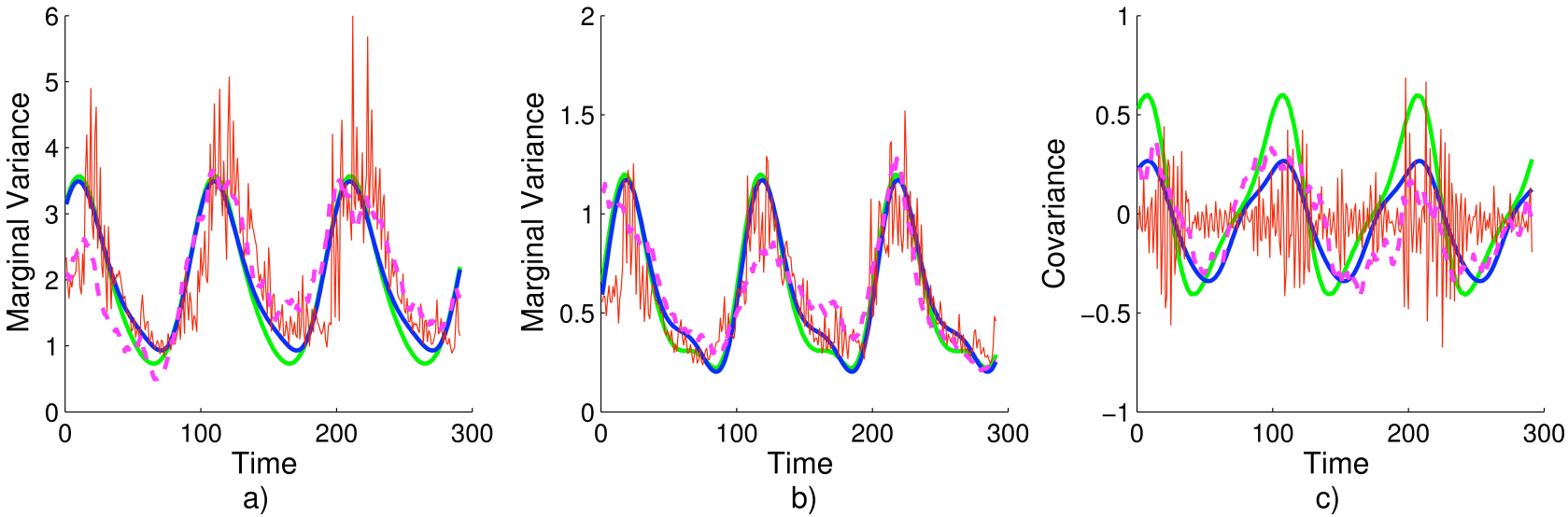
We are going to generalise Wishart distributions to be dependent on time or other inputs, making a nonparametric Bayesian model based on Gaussian Processes (GPs).

So if $\mathbf{u}_i(t) \sim \mathcal{GP}$, then $\Sigma(t) = \sum_{i=1}^{\nu} \mathbf{u}_i(t) \mathbf{u}_i(t)^\top$ defines a **Wishart process**.

This is the simplest form, many generalisations are possible.
Also closely linked to **Copula processes**.

(w/ Andrew Wilson, 2010, 2011)

Generalised Wishart Process Results



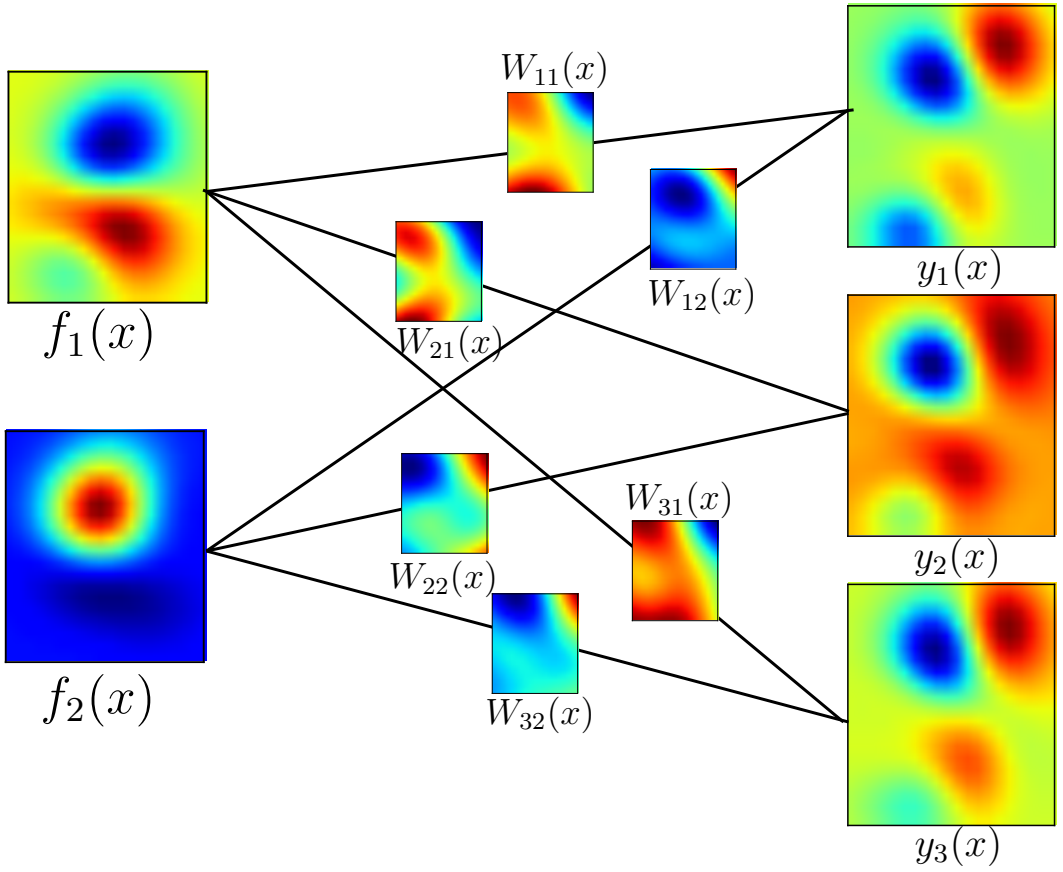
Legend: Truth, GWP, WP, MGARCH

The GWP significantly outperforms its competitors (in MSE and likelihood) on simulated and financial data, even in lower dimensions (<5) and on data that is especially suited to GARCH.

On 5D equity index data, using a GWP with a squared exponential covariance function, forecast log likelihoods are:
GWP: 2930, WP: 1710, BEKK MGARCH: 2760.

Gaussian process regression networks

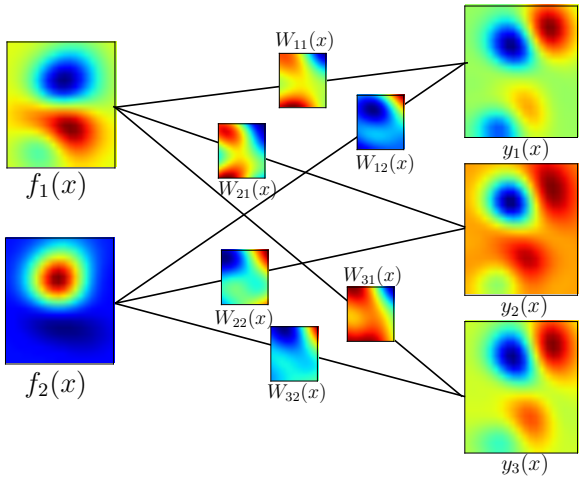
A model for multivariate regression which combines structural properties of Bayesian neural networks with the nonparametric flexibility of Gaussian processes



$$\mathbf{y}(x) = W(x)[\mathbf{f}(x) + \sigma_f \epsilon] + \sigma_y \mathbf{z}$$

(w/ Andrew Wilson, David Knowles, 2011)

Gaussian process regression networks: properties



- multi-output GP with *input-dependent correlation* structure between the outputs
- naturally accommodates *nonstationarity*, heteroskedastic noise, spatially varying lengthscales, signal amplitudes, etc
- has a *heavy-tailed* predictive distribution
- *scales well* to high-dimensional outputs by virtue of being a factor model
- if the input is time, this makes a very flexible *stochastic volatility model*
- efficient inference without costly inversions of large matrices using *elliptical slice sampling* MCMC or variational Bayes

Gaussian process regression networks: results

GENE (50D)	Average SMSE	Average MSLL
SET 1:		
GPRN (VB)	0.3356 ± 0.0294	$\mathbf{-0.5945 \pm 0.0536}$
GPRN (MCMC)	$\mathbf{0.3236 \pm 0.0311}$	-0.5523 ± 0.0478
LMC	0.6909 ± 0.0294	-0.2687 ± 0.0594
CMOGP	0.4859 ± 0.0387	-0.3617 ± 0.0511
SLFM	0.6435 ± 0.0657	-0.2376 ± 0.0456
SET 2:		
GPRN (VB)	0.3403 ± 0.0339	$\mathbf{-0.6142 \pm 0.0557}$
GPRN (MCMC)	$\mathbf{0.3266 \pm 0.0321}$	-0.5683 ± 0.0542
LMC	0.6194 ± 0.0447	-0.2360 ± 0.0696
CMOGP	0.4615 ± 0.0626	-0.3811 ± 0.0748
SLFM	0.6264 ± 0.0610	-0.2528 ± 0.0453

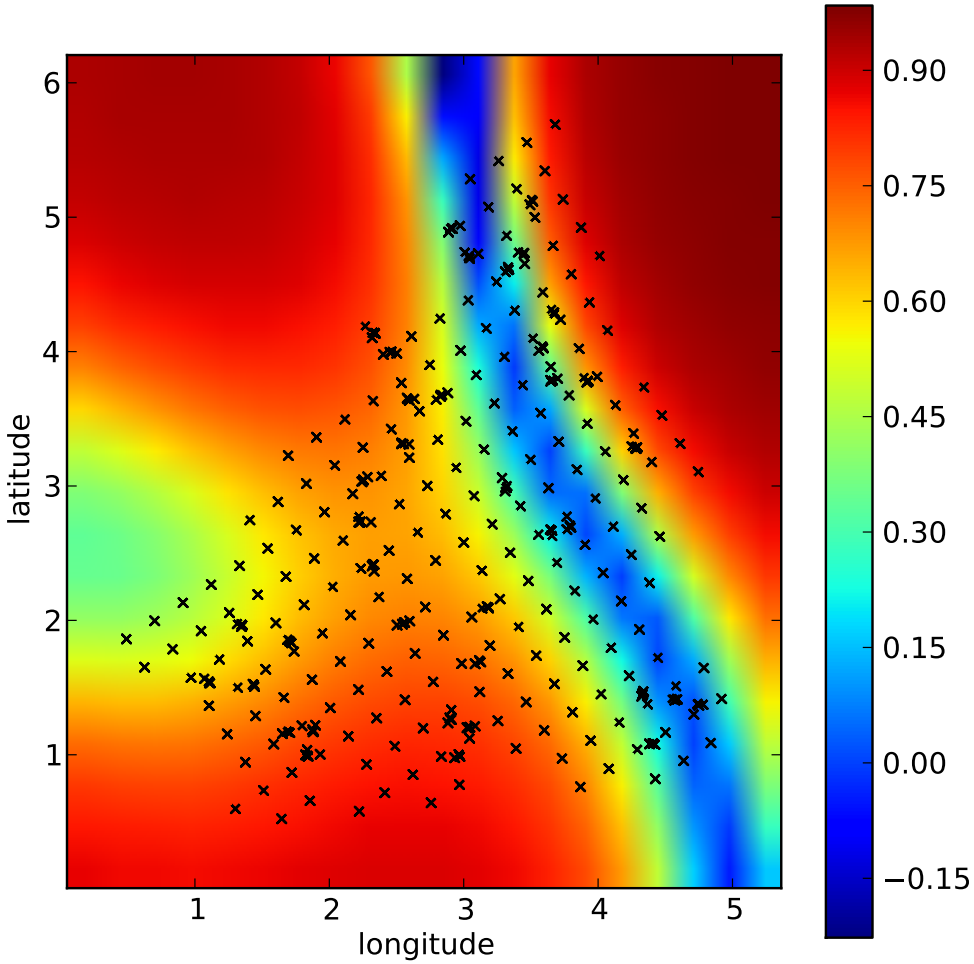
GENE (1000D)	Average SMSE	Average MSLL
GPRN (VB)	$\mathbf{0.3473 \pm 0.0062}$	$\mathbf{-0.6209 \pm 0.0085}$
GPRN (MCMC)	0.4520 ± 0.0079	-0.4712 ± 0.0327
MFITC	0.5469 ± 0.0125	-0.3124 ± 0.0200
MPITC	0.5537 ± 0.0136	-0.3162 ± 0.0206
MDTC	0.5421 ± 0.0085	-0.2493 ± 0.0183

JURA	Average MAE	Training Time (secs)
GPRN (VB)	$\mathbf{0.4040 \pm 0.0006}$	3781
GPRN* (VB)	0.4525 ± 0.0036	4560
SLFM (VB)	0.4247 ± 0.0004	1643
SLFM* (VB)	0.4679 ± 0.0030	1850
SLFM	0.4578 ± 0.0025	792
Co-kriging	0.51	
ICM	0.4608 ± 0.0025	507
CMOGP	0.4552 ± 0.0013	784
GP	0.5739 ± 0.0003	74

EXCHANGE	Historical MSE	$\mathcal{L}$ Forecast
GPRN (VB)	3.83×10^{-8}	**2073**
GPRN (MCMC)	6.120×10^{-9}	2012
GWP	$\mathbf{3.88 \times 10^{-9}}$	2020
WP	$\mathbf{3.88 \times 10^{-9}}$	1950
MGARCH	3.96×10^{-9}	2050
Empirical	4.14×10^{-9}	2006

EQUITY	Historical MSE	$\mathcal{L}$ Forecast
GPRN (VB)	0.978×10^{-9}	2740
GPRN (MCMC)	$\mathbf{0.827 \times 10^{-9}}$	2630
GWP	2.80×10^{-9}	**2930**
WP	3.96×10^{-9}	1710
MGARCH	6.69×10^{-9}	2760
Empirical	7.57×10^{-9}	2370

Gaussian process regression networks: results



Predicted correlations between cadmium and zinc

Summary

- Probabilistic modelling and Bayesian inference are two sides of the same coin
- Bayesian machine learning treats learning as a probabilistic inference problem
- Bayesian methods work well when the models are flexible enough to capture relevant properties of the data
- This motivates non-parametric Bayesian methods, e.g.:
  - Gaussian processes for **regression and classification**
  - Infinite HMMs for **time series** modelling
  - Indian buffet processes for **sparse matrices** and latent feature modelling
  - Pitman-Yor diffusion trees for **hierarchical clustering**
  - Wishart processes for **covariance modelling**
  - Gaussian process regression networks for **multi-output regression**

Thanks to

Ryan Adams
Harvard

Tom Griffiths
Berkeley

David Knowles
Cambridge

Andrew Wilson
Cambridge

`http://learning.eng.cam.ac.uk/zoubin`

`zoubin@eng.cam.ac.uk`

Some References

- Adams, R.P., Wallach, H., Ghahramani, Z. (2010) Learning the Structure of Deep Sparse Graphical Models. AISTATS 2010.
- Griffiths, T.L., and Ghahramani, Z. (2006) Infinite Latent Feature Models and the Indian Buffet Process. NIPS **18**:475–482.
- Griffiths, T.L., and Ghahramani, Z. (2011) The Indian buffet process: An introduction and review. *Journal of Machine Learning Research* **12**(Apr):1185–1224.
- Knowles, D.A. and Ghahramani, Z. (2011) Nonparametric Bayesian Sparse Factor Models with application to Gene Expression modelling. *Annals of Applied Statistics* **5**(2B):1534-1552.
- Knowles, D.A. and Ghahramani, Z. (2011) Pitman-Yor Diffusion Trees. In *Uncertainty in Artificial Intelligence (UAI 2011)*.
- Meeds, E., Ghahramani, Z., Neal, R. and Roweis, S.T. (2007) Modeling Dyadic Data with Binary Latent Factors. NIPS **19**:978–983.
- Wilson, A.G., and Ghahramani, Z. (2010, 2011) Generalised Wishart Processes. arXiv:1101.0240v1. and UAI 2011
- Wilson, A.G., Knowles, D.A., and Ghahramani, Z. (2011) Gaussian Process Regression Networks. arXiv.

Appendix

Support Vector Machines

Consider soft-margin Support Vector Machines:

$$\min_{\mathbf{w}} \ \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i (1 - y_i f_i)_+$$

where $()_+$ is the hinge loss and $f_i = f(\mathbf{x}_i) = \mathbf{w} \cdot \mathbf{x}_i + w_0$. Let's kernelize this:

$$\mathbf{x}_i \to \boldsymbol{\phi}(\mathbf{x}_i) = k(\cdot, \mathbf{x}_i), \qquad \mathbf{w} \to f(\cdot)$$

By reproducing property: $\langle k(\cdot, \mathbf{x}_i), f(\cdot)\rangle = f(\mathbf{x}_i)$.

By representer theorem, solution: $f(\mathbf{x}) = \sum_i \alpha_i k(\mathbf{x}, \mathbf{x}_i)$

Defining $\mathbf{f} = (f_1, \dots f_N)^T$ note that $\mathbf{f} = \mathbf{K}\boldsymbol{\alpha}$, so $\boldsymbol{\alpha} = \mathbf{K}^{-1}\mathbf{f}$

Therefore the regularizer $\frac{1}{2}\|\mathbf{w}\|^2 \to \frac{1}{2}\|f\|^2_{\mathcal{H}} = \frac{1}{2}\langle f(\cdot), f(\cdot)\rangle_{\mathcal{H}} = \frac{1}{2}\boldsymbol{\alpha}^\top \mathbf{K}\boldsymbol{\alpha} = \frac{1}{2}\mathbf{f}^\top \mathbf{K}^{-1}\mathbf{f}$

So we can rewrite the kernelized SVM loss as:

$$\boxed{\min_{\mathbf{f}} \ \frac{1}{2}\mathbf{f}^\top \mathbf{K}^{-1}\mathbf{f} + C\sum_i (1 - y_i f_i)_+}$$

Posterior Inference in IBPs

$$P(\mathbf{Z}, \alpha|\mathbf{X}) \propto P(\mathbf{X}|\mathbf{Z})P(\mathbf{Z}|\alpha)P(\alpha)$$

Gibbs sampling: $P(z_{nk} = 1|\mathbf{Z}_{-(nk)}, \mathbf{X}, \alpha) \propto P(z_{nk} = 1|\mathbf{Z}_{-(nk)}, \alpha)P(\mathbf{X}|\mathbf{Z})$

- If $m_{-n,k} > 0$, $P(z_{nk} = 1|\mathbf{z}_{-n,k}) = \frac{m_{-n,k}}{N}$
- For infinitely many k such that $m_{-n,k} = 0$: Metropolis steps with truncation* to sample from the number of new features for each object.
- If α has a Gamma prior then the posterior is also Gamma $\rightarrow$ Gibbs sample.

Conjugate sampler: assumes that $P(\mathbf{X}|\mathbf{Z})$ can be computed.

Non-conjugate sampler: $P(\mathbf{X}|\mathbf{Z}) = \int P(\mathbf{X}|\mathbf{Z}, \theta)P(\theta)d\theta$ cannot be computed, requires sampling latent θ as well (e.g. approximate samplers based on (Neal 2000) non-conjugate DPM samplers).

Slice sampler: works for non-conjugate case, is not approximate, and has an *adaptive truncation level* using an IBP stick-breaking construction (Teh, et al 2007) see also (Adams et al 2010).

Deterministic Inference: variational inference (Doshi et al 2009a) parallel inference (Doshi et al 2009b), beam-search MAP (Rai and Daume 2011), power-EP (Ding et al 2010)