Gaussian Processes for Machine Learning

Chris Williams

Institute for Adaptive and Neural Computation
School of Informatics, University of Edinburgh, UK

August 2007

Overview

1. What is machine learning?
2. Gaussian Processes for Machine Learning
3. Multi-task Learning

1. What is Machine Learning?

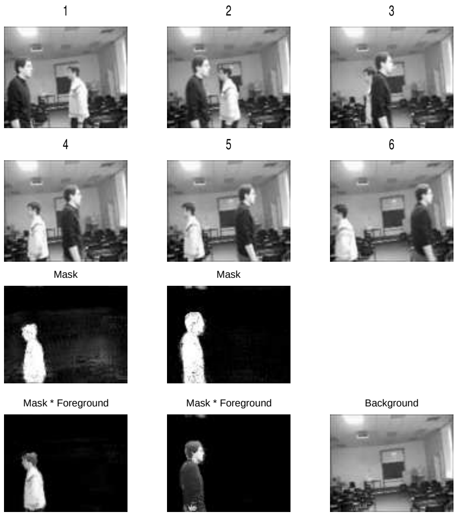
- The goal of machine learning is to build computer systems that can adapt and learn from their experience. (Dietterich, 1999)
- Machine learning usually refers to changes in systems that perform tasks associated with artificial intelligence (AI). Such tasks involve recognition, diagnosis, planning, robot control, prediction, etc. (Nilsson, 1996)
- Some reasons for adaptation:
  - Some tasks can be hard to define except via examples
  - Adaptation can improve a human-built system, or track changes over time
- Goals can be autonomous machine performance, or enabling humans to learn from data (data mining)

Roots of Machine Learning

- Statistical pattern recognition, adaptive control theory (EE)
- Artificial Intelligence: e.g. discovering rules using decision trees, inductive logic programming
- Brain models, e.g. neural networks
- Psychological models
- Statistics

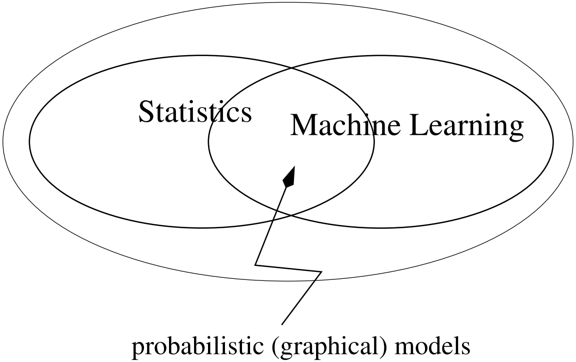
Problems Addressed by Machine Learning

- **Supervised Learning**
  model $p(y|\mathbf{x})$: regression, classification, *etc*
- **Unsupervised Learning**
  model $p(\mathbf{x})$: not just clustering!
- **Reinforcement Learning**
  Markov decision processes, POMDPs, planning.



(Williams and Titisias, 2004)

Machine Learning and Statistics



- Same models, but different problems?
- Not all machine learning methods are based on probabilisic models, e.g. SVMs, non-negative matrix factorization

Some Differences

- Statistics: focus on understanding data in terms of models
- Statistics: interpretability, hypothesis testing
- Machine Learning: greater focus on prediction
- Machine Learning: focus on the analysis of learning algorithms (not just large dataset issues)

Slide from Rob Tibshirani (early 1990s)

NEURAL NETS	STATISTICS
network	model
weights	parameters
learning	fitting
generalization	test set performance
supervised learning	regression/classification
unsupervised learning	density estimation
optimal brain damage	model selection
large grant = \$100,000	large grant= \$10,000
nice place to have a meeting: Snowbird, Utah, French Alps	nice place to have a meeting: Las Vegas in August

2. Gaussian Processes for Machine Learning

- Gaussian processes
- History
- Regression, classification and beyond
- Covariance functions/kernels
- Dealing with hyperparameters
- Theory
- Approximations for large datasets

Gaussian Processes

- A Gaussian process is a stochastic process specified by its mean and covariance functions
- Mean function

$$\mu(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})]$$

  often we take $\mu(\mathbf{x}) \equiv 0 \ \forall \mathbf{x}$
- Covariance function

$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(f(\mathbf{x}) - \mu(\mathbf{x}))(f(\mathbf{x}') - \mu(\mathbf{x}'))]$$

- A Gaussian process prior over functions can be thought of as a Gaussian prior on the coefficients $\mathbf{w} \sim N(0, \Lambda)$ where

$$Y(\mathbf{x}) = \sum_{i=1}^{N_F} w_i \phi_i(\mathbf{x})$$

In many interesting cases, $N_F = \infty$

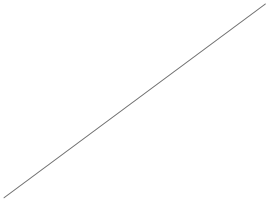
- Can choose ϕ's as eigenfunctions of the kernel $k(\mathbf{x}, \mathbf{x}')$ wrt $p(\mathbf{x})$ (Mercer)

$$\int k(\mathbf{x}, \mathbf{y}) p(\mathbf{x}) \phi_i(\mathbf{x})\, d\mathbf{x} = \lambda_i \phi_i(\mathbf{y})$$

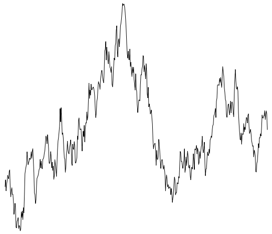
- (For stationary covariance functions and Lebesgue measure we get instead

$$\int k(\mathbf{x} - \mathbf{x}') e^{-2\pi i \mathbf{s}\cdot\mathbf{x}} d\mathbf{x} = S(\mathbf{s}) e^{-2\pi i \mathbf{s}\cdot\mathbf{x}'}$$

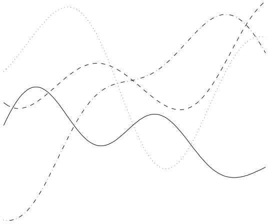
where $S(\mathbf{s})$ is the power spectrum)



$$k(x,x') = \sigma_0^2 + \sigma_1^2 x x'$$



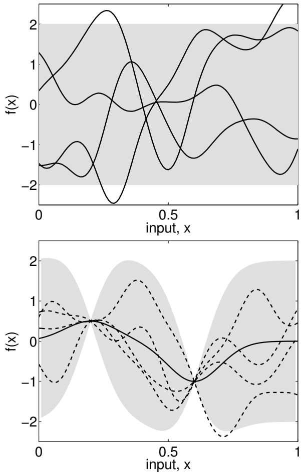
$$k(x,x') = \exp -|x - x'|$$



$$k(x,x') = \exp -(x - x')^2$$

Prediction with Gaussian Processes

- A non-parametric prior over functions
- Although GPs can be infinite-dimensional objects, prediction from a finite dataset is $O(n^3)$



Gaussian Process Regression

Dataset $\mathcal{D} = (\mathbf{x}_i, y_i)_{i=1}^{n}$, Gaussian likelihood $p(y_i|f_i) \sim N(0, \sigma^2)$

$$\bar{f}(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}, \mathbf{x}_i)$$

where

$$\boldsymbol{\alpha} = (K + \sigma^2 I)^{-1}\mathbf{y}$$

$$\mathrm{var}(f(\mathbf{x})) = k(\mathbf{x}, \mathbf{x}) - \mathbf{k}^T(\mathbf{x})(K + \sigma^2 I)^{-1}\mathbf{k}(\mathbf{x})$$

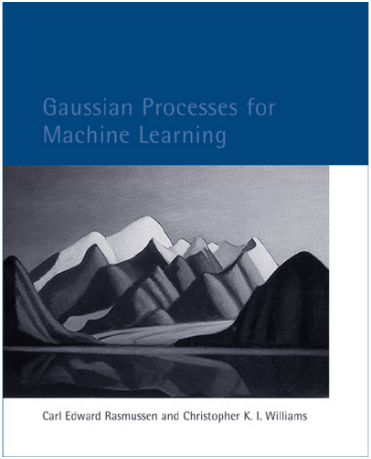
in time $O(n^3)$, with $\mathbf{k}(\mathbf{x}) = (k(\mathbf{x}, \mathbf{x}_1), \ldots, \mathbf{k}(\mathbf{x}, \mathbf{x}_n))^T$

Some GP History

- 1940s: Wiener, Kolmogorov (time series)
- Geostatistics (Matheron, 1973), Whittle (1963)
- O'Hagan (1978); Sacks et al (Design and Analysis of Computer Experiments, 1989)
- Williams and Rasmussen (1996), inspired by Neal's (1996) construction of GPs from neural networks with an infinite number of hidden units
- Regularization framework (Tikhonov and Arsenin, 1977; Poggio and Girosi, 1990); MAP rather than fully probabilistic
- SVMs (Vapnik, 1995): non-probabilistic, use "kernel trick" and quadratic programming

Carl Edward Rasmussen and Chris Williams, MIT Press, 2006

New: available online


Gaussian Processes for Machine Learning
Carl Edward Rasmussen and Christopher K. I. Williams

Regression, classification and beyond

- Regression with Gausian noise: e.g. robot arm inverse dynamics (21-d input space)
- Classification: binary, multiclass, e.g. handwritten digit classification
- ML community tends to use approximations to deal with non-Gaussian likelihoods, cf MCMC in statistics?
- MAP solution, Laplace approximation
- Expectation Propagation (Minka, 2001; see also Opper and Winther, 2000)
- Other likelihoods (e.g. Poisson), observations of derivatives, uncertain inputs, mixtures of GPs

Covariance functions

- Covariance function is key entity, determining notion of similarity
- Squared exponential ("Gaussian") covariance function is widely applied in ML; Matern kernel not very widely used
- Polynomial kernel $k(\mathbf{x}, \mathbf{x}') = (1 + \mathbf{x} \cdot \mathbf{x}')^p$ is popular in kernel machines literature
- Neural network covariance function (Williams, 1998)

$$k_{\mathrm{NN}}(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \sin^{-1}\left(\frac{2\tilde{\mathbf{x}}^\top M \tilde{\mathbf{x}}'}{\sqrt{(1 + 2\tilde{\mathbf{x}}^\top M \tilde{\mathbf{x}})(1 + 2\tilde{\mathbf{x}}'^\top M \tilde{\mathbf{x}}')}}\right)$$

where $\tilde{\mathbf{x}} = (1, x_1, \ldots, x_D)^\top$

- String kernels: let $\phi_s(x)$ denote the number of times a substring s appears in string x

$$k(x, x') = \sum_s w_s \phi_s(x) \phi_s(x')$$

  (Watkins, 1999; Haussler, 1999).
- Efficient methods using suffix trees to compute certain string kernels in time $|x| + |x'|$ (Leslie et al, 2003; Vishwanathan and Smola, 2003)
- Extended to tree kernels (Collins and Duffy, 2002)
- Fisher kernel

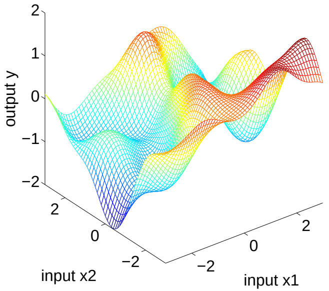
$$\phi_{\boldsymbol{\theta}}(x) = \nabla_{\boldsymbol{\theta}} \log p(x|\boldsymbol{\theta})$$
$$k(x, x') = \phi_{\boldsymbol{\theta}}(x) F^{-1} \phi_{\boldsymbol{\theta}}(x')$$

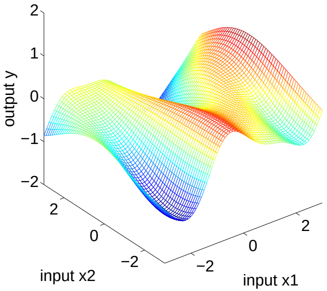
  where F is the Fisher information matrix (Jaakkola et al, 2000)

Automatic Relevance Determination

$$k_{SE}(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{x}')^\top M(\mathbf{x}_p - \mathbf{x}_q)\right)$$

- Isotropic $M = \ell^{-2} I$
- ARD: $M = \mathrm{diag}(\ell_1^{-2}, \ell_2^{-2}, \ldots, \ell_D^{-2})$





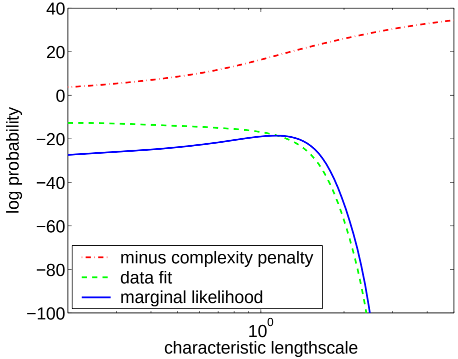
Dealing with hyperparameters

Criteria for model selection

- Marginal likelihood $p(\mathbf{y}|X, \boldsymbol{\theta})$
- Estimate the generalization error: LOO-CV $\sum_{i=1}^{n} \log p(y_i|\mathbf{y}_{-i}, X, \boldsymbol{\theta})$
- Bound the generalization error (e.g. PAC-Bayes)

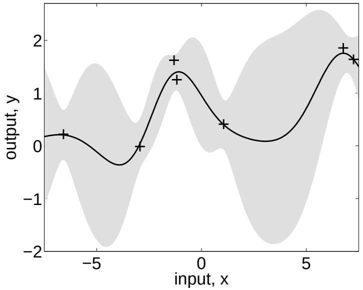
- Typically do ML-II rather than sampling of $p(\boldsymbol{\theta}|X, \mathbf{y})$
- Optimize by gradient descent (etc) on objective function
- SVMs do not generally have good methods for kernel selection

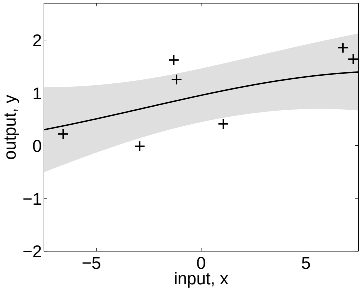
How the marginal likelihood works



$$\log p(\mathbf{y}|X,\boldsymbol{\theta}) = -\frac{1}{2}\mathbf{y}^T K_y^{-1}\mathbf{y} - \log|K_y| - \frac{n}{2}\log 2\pi$$

Marginal Likelihood and Local Optima





- There can be multiple optima of the marginal likelihood
- These correspond to different interpretations of the data

The Baby and the Bathwater

- MacKay (2003 ch 45): In moving from neural networks to kernel machines did we throw out the baby with the bathwater? i.e. the ability to learn hidden features/representations
- But consider $M = \Lambda\Lambda^\top$ for Λ being $D \times k$, for $k < D$
- The k columns of Λ can identify directions in the input space with specially high relevance (Vivarelli and Williams, 1999)

Theory

- Equivalent kernel (Silverman, 1984)
- Consistency (Diaconis and Freedman, 1986; Choudhuri, Ghoshal and Roy 2005; Choi and Schervish, 2004)
- Average case learning curves
- PAC-Bayesian analysis for GPs (Seeger, 2003)

$$p_{\mathcal{D}}\{R_{\mathcal{L}}(f_{\mathcal{D}}) \leq \hat{R}_{\mathcal{L}}(f_{\mathcal{D}}) + \mathrm{gap}(f_{\mathcal{D}}, \mathcal{D}, \delta)\} \ \geq \ 1 - \delta$$

where $R_{\mathcal{L}}(f_{\mathcal{D}})$ is the expected risk, and $\hat{R}_{\mathcal{L}}(f_{\mathcal{D}})$ is the empirical (training) risk

Approximation Methods for Large Datasets

- Fast approximate solution of the linear system
- Subset of Data
- Subset of Regressors
- Inducing Variables
- Projected Process Approximation
- FITC, PITC, BCM
- SPGP
- Empirical Comparison

Some interesting recent uses for Gaussian Processes

- Modelling transcriptional regulation using Gaussian Processes. Neil D. Lawrence, Guido Sanguinetti, Magnus Rattray (NIPS 2006)
- A Switched Gaussian Process for Estimating Disparity and Segmentation in Binocular Stereo. Oliver Williams (NIPS 2006)
- Learning to Control an Octopus Arm with Gaussian Process Temporal Difference Methods. Yaakov Engel, Peter Szabo, Dmitry Volkinshtein (NIPS 2005)
- Worst-Case Bounds for Gaussian Process Models. Sham Kakade, Matthias Seeger, Dean Foster (NIPS 2005)
- Infinite Mixtures of Gaussian Process Experts. Carl Rasmussen, Zoubin Ghahramani (NIPS 2002)

3. Multi-task Learning

- There are multiple (possibly) related tasks, and we wish to avoid *tabula rasa* learning by sharing information across tasks
- E.g. Task clustering, inter-task correlations
- Two cases:
  - With task-descriptor features $\mathbf{t}$
  - Without task-descriptor features, based solely on task identities
- Joint work with Edwin Bonilla & Felix Agakov (AISTATS 2007) and Kian Ming Chai

Multi-task Learning using Task-specific Features

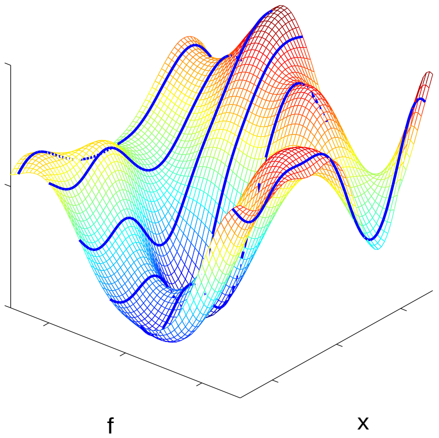
- M tasks, learn mapping $g_i(\mathbf{x})$, $i = 1, \ldots, M$
- $\mathbf{t}_i$ is task descriptor (task-specific feature vector) for task i
- $g_i(\mathbf{x}) = g(\mathbf{t}_i, \mathbf{x})$: potential for *transfer* across tasks
- Out motivation is for compiler performance prediction, where there are multiple benchmark programs (=tasks), and $\mathbf{x}$ describes sequences of code transformations
- Another example: predicting school pupil performance based on pupil and school features
- We particularly care about the case when we have very little data from the test task; here inter-task transfer will be most important

Overview

- Model setup
- Related work
- Experimental setup, feature representation
- Results
- Discussion

Task-descriptor Model

- $\mathbf{z} = \begin{pmatrix} \mathbf{x} \\ \mathbf{t} \end{pmatrix}$
- $k(\mathbf{z}, \mathbf{z}') = k_x(\mathbf{x}, \mathbf{x}') k_t(\mathbf{t}, \mathbf{t}')$
- Decomposition into task similarity (k_t) and input similarity (k_x)
- For the widely-used “Gaussian” kernel, this occurs naturally
- Independent tasks if $k_t(\mathbf{t}_i, \mathbf{t}_j) = \delta_{ij}$
- C.f. co-kriging in geostatistics (e.g. Wackernagel, 1998)
- Without task-descriptors, simply parameterize K_t


f
x

Related Work

Work using task-specific features

- Bakker and Heskes (2003) use neural networks. These can be tricky to train (local optima, number of hidden units etc)
- Yu et al (NIPS 2006, Stochastic Relational Models for Discriminative Link Prediction)

General work on Multi-task Learning

What should be transferred?

- Early work: Thrun (1996), Caruana (1997)
- Minka and Picard (1999); multiple tasks share same GP hyperparameters (but are uncorrelated)
- Evgeniou et al (2005): induce correlations between tasks based on a correlated prior over linear regression parameters (special case of co-kriging)
- Multilevel (or hierarchical) modelling in statistics (e.g. Goldstein, 2003)

Compiler Performance Prediction

- Goal: Predict speedup of a new program under a given sequence of compiler transformations
- Only have a limited number of runs of the new program, but also have data from other (related?) tasks
- Speedup s measured as

$$s(\mathbf{x}) = \frac{\mathrm{time(baseline)}}{\mathrm{time}(\mathbf{x})}$$

Example Transformation

Loop unrolling

```
 // original loop
for(i=0; i<100; i++)
  a[i] = b[i] + c[i];
```

```
 // loop unrolled twice
for(i=0; i<100; i+=2){
  a[i]   = b[i] + c[i];
  a[i+1] = b[i+1] +
c[i+1];
}
```

Experimental Setup

- Benchmarks: 11 C programs from UTDSP
- Transformations: Source-to-source using SUIF
- Platform: TI C6713 board
- 13 transformations in sequences up to length 5, using each transformation at most once $\Rightarrow$ 88214 sequences per benchmark (exhaustively enumerated)
- Significant speedups can be obtained (max is 1.84)

Input Features $\mathbf{x}$

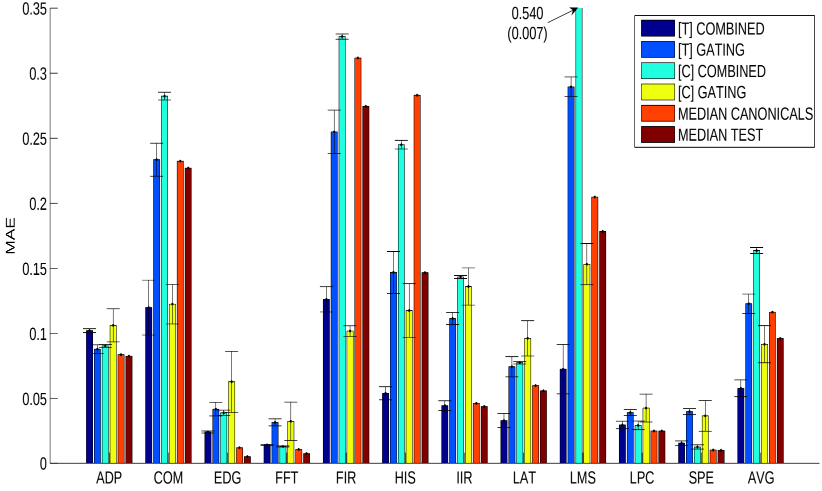
- Code features (C), or transformation-based representation (T)
- **Code features**: extract features from transformed program based on knowledge of compiler experts (code size, instructions executed, parallelism)
- 83 features reduced to 15-d with PCA
- **Transformation-based representation**: length-13 bit vector stating what transformations were used ("bag of characters")

Task-specific features $\mathbf{t}$

- Record the speedup on a small number of *canonical* sequences: response-based approach
- Canonical sequences selected by principal variables method (McCabe, 1984)
- A variety of possible criteria can be used, e.g. maximize $|\Sigma_{S_{(1)}}|$, minimize $\mathrm{tr}(\Sigma_{S_{(2)}|S_{(1)}})$. Use greedy selection
- We don't use all 88214 sequences to define the canonical sequences, only only 2048. In our experiments we use 8 canonical variables
- Could consider e.g. code features from untransformed programs, but experimentally response-based method is superior

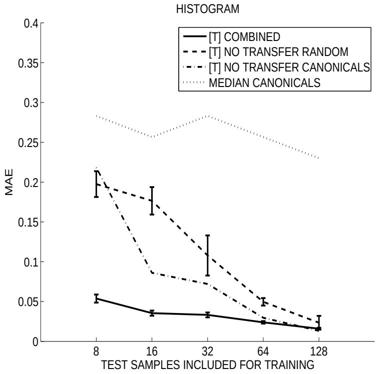
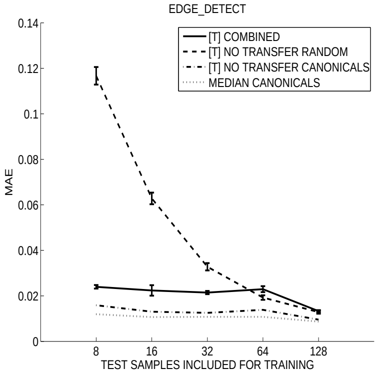
Experiments

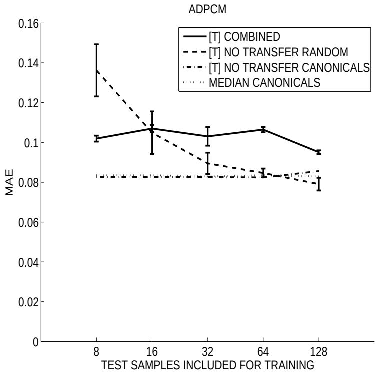
- LOO-CV setup (leave out one task at a time)
- Therefore 10 *reference* tasks for each prediction task; we used $n_r = 256$ examples per benchmark
- Use n_{te} examples from the test task ($n_{te} \geq 8$)
- Assess performance using mean absolute error (MAE) on all remaining test sequences
- Comparison to baseline "no transfer" method using just data from test task
- Used GP regression prediction with squared exponential kernel
- ARD was used, except for "no transfer" case when $n_{te} \leq 64$

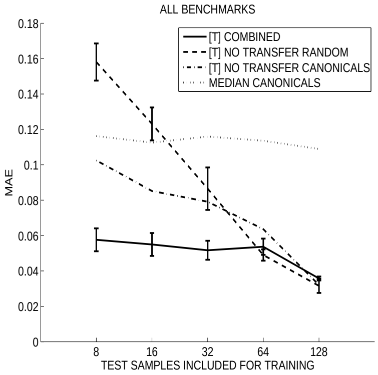

0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
MAE
0.540
(0.007)
[T] COMBINED
[T] GATING
[C] COMBINED
[C] GATING
MEDIAN CANONICALS
MEDIAN TEST
ADP
COM
EDG
FFT
FIR
HIS
IIR
LAT
LMS
LPC
SPE
AVG

Results

- T-combined is best overall (av MAE is 0.0576, compared to 0.1162 for median canonicals)
- T-combined generally either improves performance or leaves it about the same compared to T-no-transfer-canonicals


HISTOGRAM
[T] COMBINED
[T] NO TRANSFER RANDOM
[T] NO TRANSFER CANONICALS
MEDIAN CANONICALS
MAE
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
8
16
32
64
128
TEST SAMPLES INCLUDED FOR TRAINING

EDGE_DETECT
[T] COMBINED
[T] NO TRANSFER RANDOM
[T] NO TRANSFER CANONICALS
MEDIAN CANONICALS
MAE
0
0.02
0.04
0.06
0.08
0.1
0.12
0.14
8
16
32
64
128
TEST SAMPLES INCLUDED FOR TRAINING





- T-combined generally improves performance or leaves it about the same compared to the best “no transfer” scenario

Understanding Task Relatedness

- GP predictive mean is

$$\bar{s}(\mathbf{z}_*) = \mathbf{k}^T(\mathbf{z}_*)(K_f \otimes K_x + \sigma^2 I)^{-1}\mathbf{s}$$

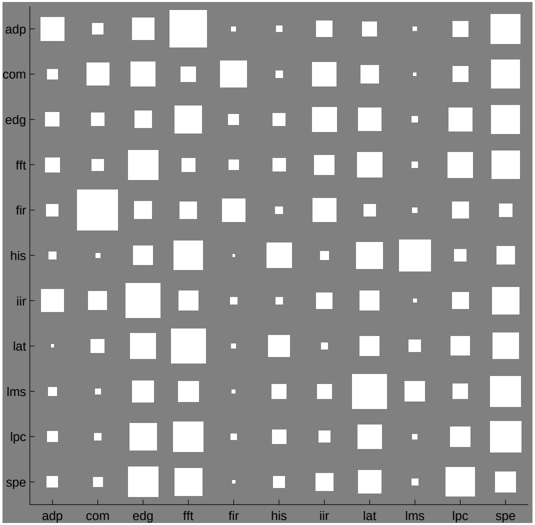
- Can look at K_f, but difficult to interpret?
- Predictive mean $s(\mathbf{z}_*) = \mathbf{h}^T(\mathbf{z}_*)\mathbf{s}$, where

$$\mathbf{h}^T(\mathbf{z}) = (h_1^1, \ldots, h_{n_r}^1, \ldots, h_1^M, \ldots, h_{n_r}^M, h_1^{M+1}, \ldots, h_{n_{te}}^{M+1},)$$

- Measure contribution of task i on test point $\mathbf{z}_*$ by computing

$$r^i(\mathbf{z}_*) = \frac{|\mathbf{h}^i(\mathbf{z}_*)|}{|\mathbf{h}(\mathbf{z}_*)|}$$

Average r's over test examples



Discussion

- Our focus is on the hard problem of prediction on a new task given very little data for that task
- The presented method allows sharing over tasks. This should be beneficial, but note that "no transfer" method has the freedom to use different hyperparams on each task
- Can learn similarity between tasks directly (unparameterized K_t), but this is not so easy if n_{te} is very small
  - Note that there is no inter-task transfer in noiseless case! (autokrigeability)

General Conclusions

Key issues:

- Designing/discovering covariance functions suitable for various types of data
- Methods for setting/inference of hyperparameters
- Dealing with large datasets

Gaussian Process Regression

Dataset $\mathcal{D} = (\mathbf{x}_i, y_i)_{i=1}^n$, Gaussian likelihood $p(y_i|f_i) \sim N(0, \sigma^2)$

$$\bar{f}(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}, \mathbf{x}_i)$$

where

$$\boldsymbol{\alpha} = (K + \sigma^2 I)^{-1}\mathbf{y}$$

$$\mathrm{var}(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}) - \mathbf{k}^T(\mathbf{x})(K + \sigma^2 I)^{-1}\mathbf{k}(\mathbf{x})$$

in time $O(n^3)$, with $\mathbf{k}(\mathbf{x}) = (k(\mathbf{x}, \mathbf{x}_1), \ldots, \mathbf{k}(\mathbf{x}, \mathbf{x}_n))^T$

Fast approximate solution of linear systems

- Iterative solution of $(K + \sigma_n^2 I)\mathbf{v} = \mathbf{y}$, e.g. using Conjugate Gradients. Minimizing

$$\frac{1}{2}\mathbf{v}^T(K + \sigma_n^2 I)\mathbf{v} - \mathbf{y}^T\mathbf{v}.$$

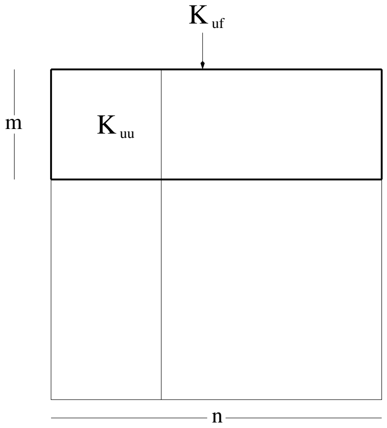
  This takes $O(kn^2)$ for k iterations.
- Fast approximate matrix-vector multiplication

$$\sum_{i=1}^{n} k(\mathbf{x}_j, \mathbf{x}_i) v_i$$

- k-d tree/ dual tree methods (Gray, 2004; Shen, Ng and Seeger, 2006; De Freitas et al 2006)
- Improved Fast Gauss transform (Yang et al, 2005)

Subset of Data

- Simply keep m datapoints, discard the rest: $O(m^3)$
- Can choose the subset randomly, or by a greedy selection criterion
- If we are prepared to do work for each test point, can select training inputs nearby to the test point. Stein (*Ann. Stat.*, 2002) shows that a screening effect operates for some covariance functions



$$\tilde{K} = K_{fu} K_{uu}^{-1} K_{uf}$$

Nyström approximation to K

Subset of Regressors

- Silverman (1985) showed that the *mean* GP predictor can be obtained from the finite-dimensional model

$$f(\mathbf{x}_*) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_*, \mathbf{x}_i)$$

with a prior $\boldsymbol{\alpha} \sim \mathcal{N}(\mathbf{0}, K^{-1})$

- A simple approximation to this model is to consider only a subset of regressors

$$f_{\mathrm{SR}}(\mathbf{x}_*) = \sum_{i=1}^{m} \alpha_i k(\mathbf{x}_*, \mathbf{x}_i), \qquad \text{with} \qquad \boldsymbol{\alpha}_u \sim \mathcal{N}(\mathbf{0}, K_{uu}^{-1})$$

$$
\begin{aligned}
\bar{f}_{\mathrm{SR}}(\mathbf{x}_*) &= \mathbf{k}_u(\mathbf{x}_*)^\top (K_{uf}K_{fu} + \sigma_n^2 K_{uu})^{-1} K_{uf}\mathbf{y}, \\
\mathbb{V}[f_{\mathrm{SR}}(\mathbf{x}_*)] &= \sigma_n^2 \mathbf{k}_u(\mathbf{x}_*)^\top (K_{uf}K_{fu} + \sigma_n^2 K_{uu})^{-1} \mathbf{k}_u(\mathbf{x}_*)
\end{aligned}
$$

- SoR corresponds to using a *degenerate* GP prior (finite rank)

Inducing Variables

Quiñonero-Candela and Rasmussen (JMLR, 2005)

$$p(\mathbf{f}_*|\mathbf{y}) = \frac{1}{p(\mathbf{y})} \int p(\mathbf{y}|\mathbf{f}) p(\mathbf{f}, \mathbf{f}_*) d\mathbf{f}$$

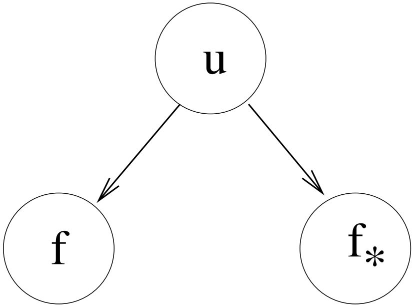
Now introduce inducing variables $\mathbf{u}$

$$p(\mathbf{f}, \mathbf{f}_*) = \int p(\mathbf{f}, \mathbf{f}_*, \mathbf{u}) d\mathbf{u} = \int p(\mathbf{f}, \mathbf{f}_*|\mathbf{u}) p(\mathbf{u}) d\mathbf{u}$$

Approximation

$$p(\mathbf{f}, \mathbf{f}_*) \simeq q(\mathbf{f}, \mathbf{f}_*) \overset{def}{=} \int q(\mathbf{f}|\mathbf{u}) q(\mathbf{f}_*|\mathbf{u}) p(\mathbf{u}) d\mathbf{u}$$

$q(\mathbf{f}|\mathbf{u})$ – training conditional
$q(\mathbf{f}_*|\mathbf{u})$ – test conditional


$$\begin{tikzcd} & \mathbf{u} \arrow[dl] \arrow[dr] & \\ \mathbf{f} & & \mathbf{f}_* \end{tikzcd}$$

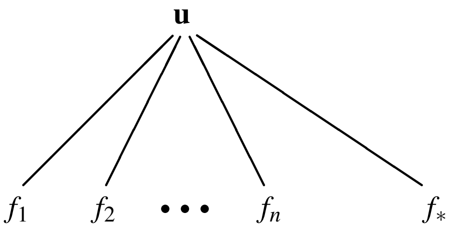
Inducing variables can be:

- (sub)set of training points
- (sub)set of test points
- new $\mathbf{x}$ points

Projected Process Approximation—PP

(Csato & Opper, 2002; Seeger, et al 2003; aka PLV, DTC)

- Inducing variables are subset of training points
- $q(\mathbf{y}|\mathbf{u}) = \mathcal{N}(\mathbf{y}|K_{fu}K_{uu}^{-1}\mathbf{u}, \sigma_n^2 I)$
- $K_{fu}K_{uu}^{-1}\mathbf{u}$ is mean prediction for $\mathbf{f}$ given $\mathbf{u}$
- Predictive mean for PP is the same as SR, but variance is never smaller. SR is like PP but with deterministic $q(f_*|\mathbf{u})$



FITC, PITC and BCM

See Quiñonero-Candela and Rasmussen (2005) for overview

- Under PP, $q(\mathbf{f}|\mathbf{u}) = \mathcal{N}(\mathbf{y}|K_{fu}K_{uu}^{-1}\mathbf{u}, 0)$
- Instead FITC (Snelson and Ghahramani, 2005) uses individual predictive variances $\mathrm{diag}[K_{ff} - K_{fu}K_{uu}^{-1}K_{uf}]$, i.e. fully independent training conditionals
- PP can make poor predictions in low noise [S Q-C M R W]
- PITC uses blocks of training points to improve the approximation
- BCM (Tresp, 2000) is the same approximation as PITC, except that the *test* points are the inducing set

Sparse GPs using Pseudo-inputs

(Snelson and Ghahramani, 2006)

- FITC approximation, but inducing inputs are new points, in neither the training or test sets
- Locations of the inducing inputs are changed along with hyperparameters so as to maximize the approximate marginal likelihood

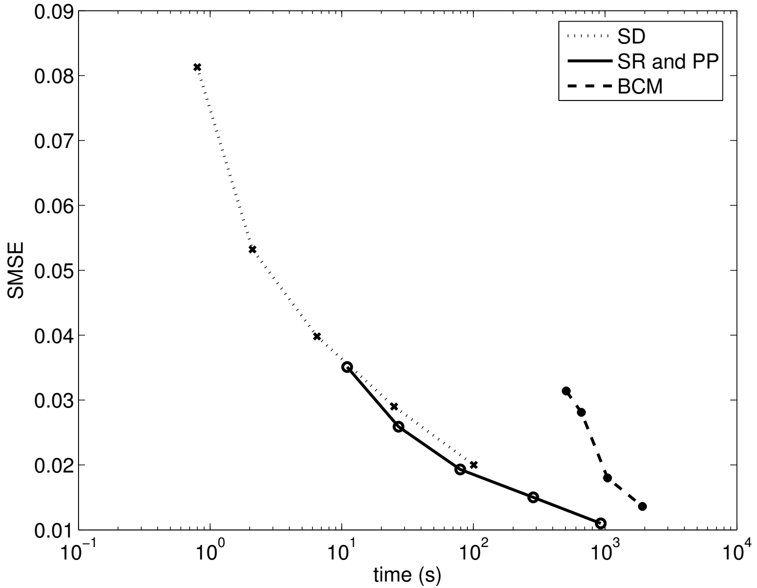
Complexity

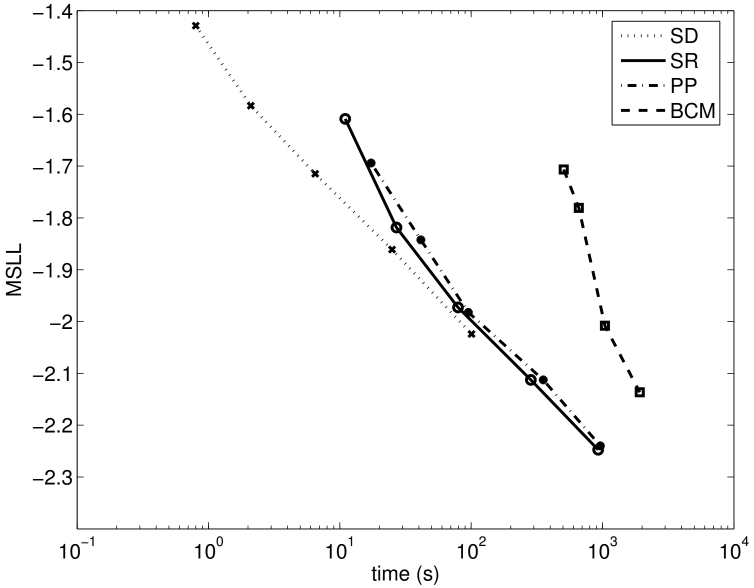
Method	Storage	Initialization	Mean	Variance
SD	$O(m^2)$	$O(m^3)$	$O(m)$	$O(m^2)$
SR	$O(mn)$	$O(m^2n)$	$O(m)$	$O(m^2)$
PP, FITC	$O(mn)$	$O(m^2n)$	$O(m)$	$O(m^2)$
BCM	$O(mn)$		$O(mn)$	$O(mn)$

Empirical Comparison

- Robot arm problem, 44,484 training cases in 21-d, 4,449 test cases
- For SD method subset of size m was chosen at random, hyperparameters set by optimizing marginal likelihood (ARD). Repeated 10 times
- For SR, PP and BCM methods same subsets/hyperparameters were used (BCM: hyperparameters only)

Method	m	SMSE	MSLL	mean runtime (s)
SD	256	0.0813 ± 0.0198	-1.4291 ± 0.0558	0.8
	512	0.0532 ± 0.0046	-1.5834 ± 0.0319	2.1
	1024	0.0398 ± 0.0036	-1.7149 ± 0.0293	6.5
	2048	0.0290 ± 0.0013	-1.8611 ± 0.0204	25.0
	4096	0.0200 ± 0.0008	-2.0241 ± 0.0151	100.7
SR	256	0.0351 ± 0.0036	-1.6088 ± 0.0984	11.0
	512	0.0259 ± 0.0014	-1.8185 ± 0.0357	27.0
	1024	0.0193 ± 0.0008	-1.9728 ± 0.0207	79.5
	2048	0.0150 ± 0.0005	-2.1126 ± 0.0185	284.8
	4096	0.0110 ± 0.0004	-2.2474 ± 0.0204	927.6
PP	256	0.0351 ± 0.0036	-1.6940 ± 0.0528	17.3
	512	0.0259 ± 0.0014	-1.8423 ± 0.0286	41.4
	1024	0.0193 ± 0.0008	-1.9823 ± 0.0233	95.1
	2048	0.0150 ± 0.0005	-2.1125 ± 0.0202	354.2
	4096	0.0110 ± 0.0004	-2.2399 ± 0.0160	964.5
BCM	256	0.0314 ± 0.0046	-1.7066 ± 0.0550	506.4
	512	0.0281 ± 0.0055	-1.7807 ± 0.0820	660.5
	1024	0.0180 ± 0.0010	-2.0081 ± 0.0321	1043.2


SD
SR and PP
BCM
SMSE
0.09
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}
10^{4}
time (s)



SD
SR
PP
BCM
MSLL
time (s)
−1.4
−1.5
−1.6
−1.7
−1.8
−1.9
−2
−2.1
−2.2
−2.3
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}
10^{4}

- Judged on time, for this dataset SD, SR and PP are on the same trajectory, with BCM being worse
- But what about greedy vs random subset selection, methods to set hyperparameters, different datasets?
- In general, we must take into account *training* (initialization), *testing* and *hyperparameter learning* times separately [S Q-C M R W]. Balance will depend on your situation.