A Framework for Evaluating Approximation Methods for Gaussian Process Regression

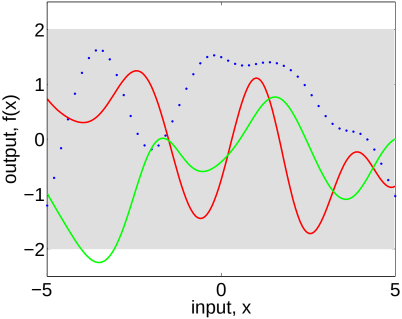
Krzysztof Chalupka, Chris Williams, Iain Murray

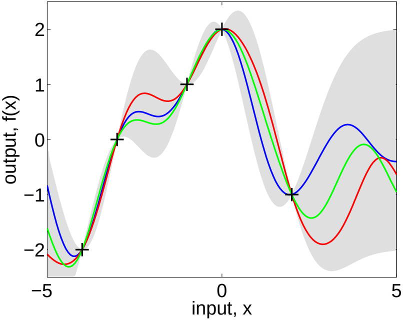
Institute for Adaptive and Neural Computation
School of Informatics, University of Edinburgh, UK

May 2012

Gaussian Processes: from Prior to Posterior

Training set $\{\mathbf{x}_i, y_i\}_{i=1}^{n}$





Predictive distribution

$$p(y_*|\mathbf{x}_*, X, y, M) = \mathcal{N}(\mathbf{k}^T(\mathbf{x}_*, X)[K + \sigma_n^2 I]^{-1}\mathbf{y},\\ k(\mathbf{x}_*, \mathbf{x}_*) + \sigma_n^2 - \mathbf{k}^T(\mathbf{x}_*, X)[K + \sigma_n^2 I]^{-1}\mathbf{k}(\mathbf{x}_*, X))$$

Marginal Likelihood

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

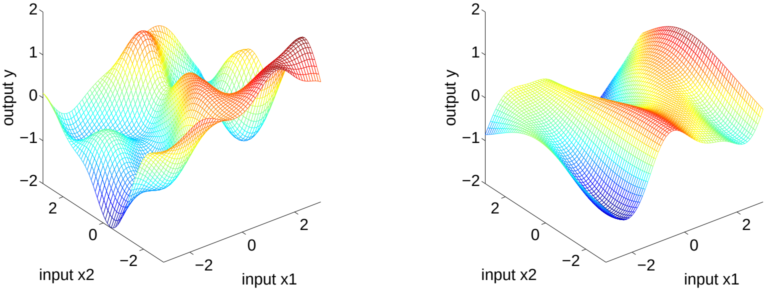
where $K_y = K + \sigma_n^2 I$.

- This can be used to adjust the free parameters (hyperparameters) of a kernel
- Prediction and evaluation of marginal likelihood are all $O(n^3)$

Automatic Relevance Determination

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

- Isotropic $M = \ell^{-2} I$
- ARD: $M = \mathrm{diag}(\ell_1^{-2}, \ell_2^{-2}, \ldots, \ell_D^{-2})$ (cf Neal, 1996)



The Nature of the Underlying Problem

- Complexity of target function (e.g. Fourier spectrum)
- Noise level
- Dimensionality of $\mathbf{x}$ space (intrinsic or apparent)

Approximate GPR methods

- Subset of Data (SoD)
  - keep m data points, simply throw away the rest
  - select points randomly, or furthest point clustering (Gonzales, 1985)
- Fully Independent Training Conditional (FITC)
  - "absorb" all datapoints onto an m-dimensional predictor
  - We choose inducing points $\mathbf{U}$ from the training set

$$k_{SoR}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{k}(\mathbf{x}_i, \mathbf{U}) K_{\mathbf{UU}}^{-1} \mathbf{k}(\mathbf{U}, \mathbf{x}_j)$$

$$k_{FITC}(\mathbf{x}_i, \mathbf{x}_j) = k_{SoR}(\mathbf{x}_i, \mathbf{x}_j) + \delta_{ij}[k(\mathbf{x}_i, \mathbf{x}_j) - k_{SoR}(\mathbf{x}_i, \mathbf{x}_j)]$$

- Local
  - Create k data clusters: run GPR in each
  - We devised Recursive Projection Clustering (RPC) to obtain clusters of equal size
  - Hyperparameters: joint across all clusters, or separate
- Each method has its associated marginal likelihood approximation

- Iterative methods and IFGT
  - Use iterative solution of linear system (e.g. conjugate gradients).
  - Approximate each matrix-vector multiply (MVM) using IFGT.
  - Slow for predictive variances
- Lots of other methods proposed, including:
  - Exploit structure, e.g. Fourier methods for stationary covariance functions and grid designs
  - GPs $\rightarrow$ GRMFs (Lindgren, Rue, Lindström, 2011)

Comparison of space and time complexity

Method	Storage	Training	Mean	Variance
Full	$O(n^2)$	$O(n^3)$	$O(n)$	$O(n^2)$
SoD	$O(m^2)$	$O(m^3)$	$O(m)$	$O(m^2)$
FITC	$O(mn)$	$O(m^2 n)$	$O(m)$	$O(m^2)$
Local	$O(mn)$	$O(m^2 n)$	$O(m)$	$O(m^2)$

Computational phases

hyperparameter learning: The hyperparameters are learned, by for example maximizing the log marginal likelihood. This is often the most computationally expensive phase.

training: Given the hyperparameters, all computations that don't involve test inputs are performed, such as computing $(K + \sigma^2 I)^{-1}\mathbf{y}$, and/or computing the Cholesky decomposition of $K + \sigma_n^2 I$.

testing: Only the computations involving the test inputs are carried out, those which could not have been done previously. This phase may be significant if there is a very large test set.

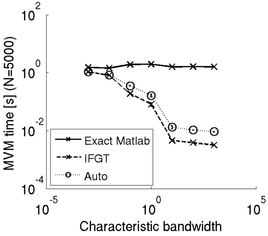
Comparing Approximations

- **Consider SMSE, MSLL as a function of training or testing compute time**
- Q: How to handle the hyperparameters?
- A: Let each method choose its own
- This is sensible for real-world data, as opposed e.g. to synthetic data

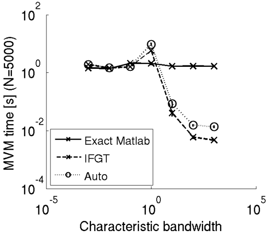
Experiments

- Datasets
  - SYNTH2 2-d GP; $n = 30,543$ for training, same for test
  - SYNTH8 8-d GP; $n = 30,543$ for training, same for test
  - SARCOS $D = 21$, $n = 44,484$, plus 4,449 for testing
  - CHEM $D = 15$, $n = 31,536$ for training, same for test
- Error measures
  - standardized mean squared error (SMSE) on test set
  - mean standardized log loss (MSLL) on test set
    average $p(y_*|\mathcal{D}, \mathbf{x}_*)$ over test set, subtract same score for trivial model which predicts mean and variance of training set
- Which method do you think will do best?

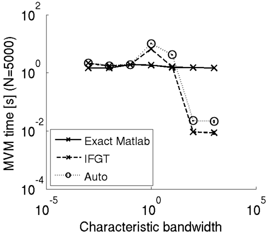
IFGT Results



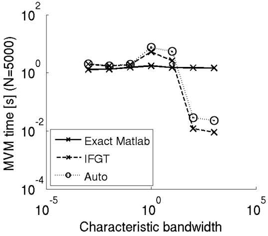
SYNTH2



SYNTH8



CHEM

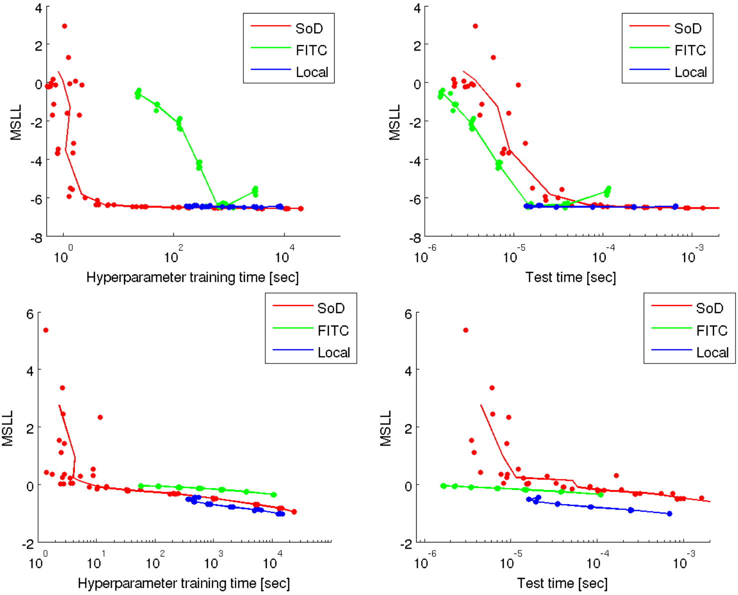


SARCOS

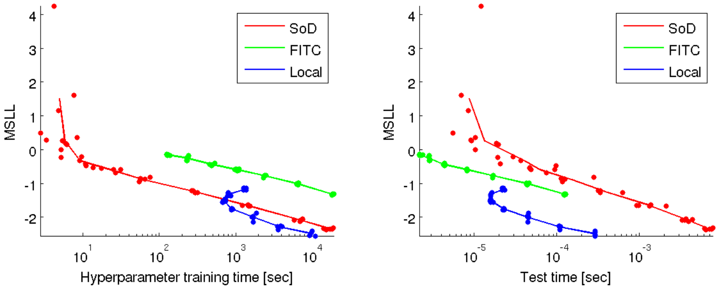
IFGT only provides useful speedups for SYNTH2

SYNTH2

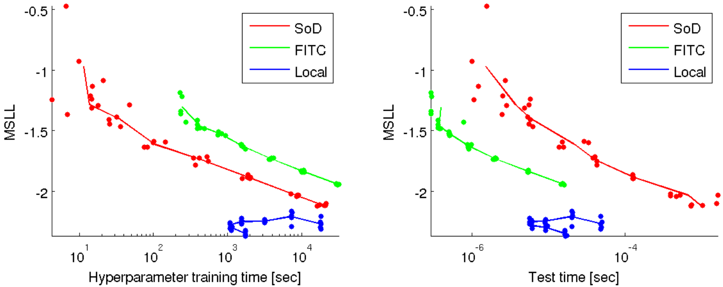
SYNTH8



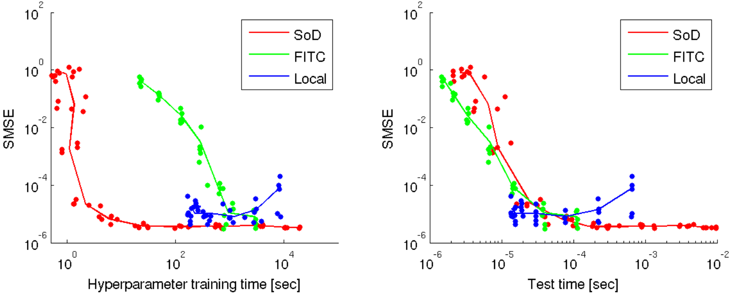
CHEM


SoD
FITC
Local
MSLL
Hyperparameter training time [sec]
Test time [sec]

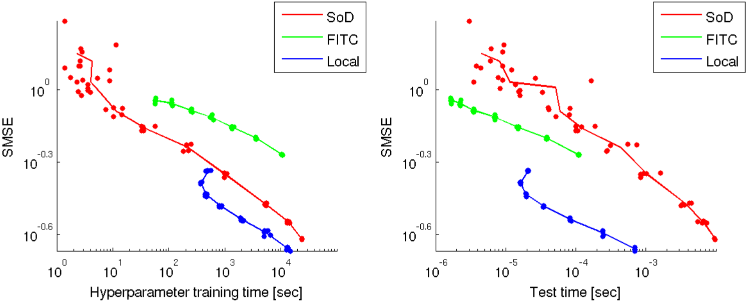
SARCOS


SoD
FITC
Local
MSLL
Hyperparameter training time [sec]
Test time [sec]

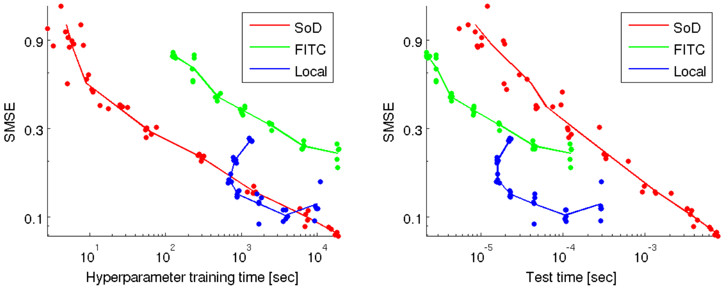
SYNTH2



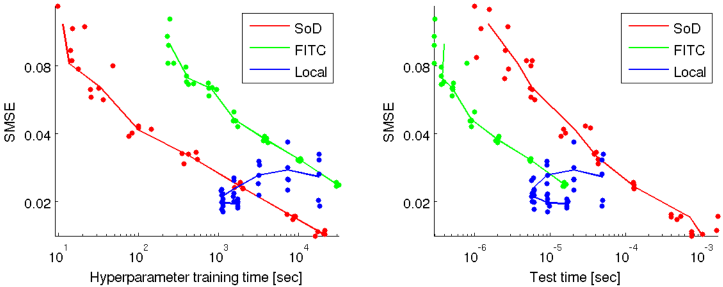
SYNTH8



CHEM


SoD
FITC
Local
SMSE
0.9
0.3
0.1
10^1
10^2
10^3
10^4
Hyperparameter training time [sec]
10^{-5}
10^{-4}
10^{-3}
Test time [sec]

SARCOS


SoD
FITC
Local
SMSE
0.08
0.04
0.02
10^1
10^2
10^3
10^4
Hyperparameter training time [sec]
10^{-6}
10^{-5}
10^{-4}
10^{-3}
Test time [sec]

Results: Conclusions

- SoD dominates FITC wrt hyperparameter learning
- FITC dominates SoD wrt test time
- Both SoD and FITC behaved monotonically with m
- The Local method is more variable, but can win for some problems and cluster sizes. Non-monotonic time wrt m.
- IFGT only provided a speedup for SYNTH2

Futher issues

- Subset selection methods (e.g. IVM)
- Mix-and-match, e.g. train hyperparameters with SoD, then use FITC at test time?
- Lower-level programming to improve Local for small *m*

Conclusions

- Assess approximate methods by quality obtained vs compute time
- New methods should compare to standard baselines (e.g. SoD, FITC)
- Paper available at `http://homepages.inf.ed.ac.uk/ckiw/online_pubs.html`
- Code and data at `http://homepages.inf.ed.ac.uk/ckiw/code/gpr_approx.html`

Carl Edward Rasmussen and
Chris Williams
MIT Press, 2006

`www.GaussianProcess.org/gpml`

Available free on the internet