Journal of Machine Learning Research 13 (2012) 775-779 Submitted 9/11; Published 1/12

GPLP: A Local and Parallel Computation Toolbox for Gaussian Process Regression

Chiwoo Park CHIWOO.PARK@ENG.FSU.EDU
Department of Industrial and Manufacturing Engineering
Florida A&M - Florida State University College of Engineering
2525 Pottsdamer St
Tallahassee, FL 32310-6046, USA

Jianhua Z. Huang JIANHUA@STAT.TAMU.EDU
Department of Statistics
Texas A&M University
3143 TAMU
College Station, TX 77843-3143, USA

Yu Ding YUDING@IEMAIL.TAMU.EDU
Department of Industrial and Systems Engineering
Texas A&M University
3131 TAMU
College Station, TX 77843-3131, USA

Editor: Mikio Braun

Abstract

This paper presents the *Getting-started* style documentation for the local and parallel computation toolbox for Gaussian process regression (`GPLP`), an open source software package written in `Matlab` (but also compatible with `Octave`). The working environment and the usage of the software package will be presented in this paper.

Keywords: Gaussian process regression, domain decomposition method, partial independent conditional, bagging for Gaussian process, local probabilistic regression

1. Introduction

The Gaussian process regression (GP regression) has recently developed to be a useful tool in machine learning (Rasmussen and Williams, 2006). A GP regression provides the best unbiased linear estimator computable by a simple closed form expression and is a popular method for interpolation or extrapolation. A major limitation of GP regression is its computational complexity, scaled by $O(N^3)$, where N is the number of training observations.

Many fast computation methods have been introduced in the literature to relieve the computation burden: *matrix approximation* (Williams and Seeger, 2000; Smola and Bartlett, 2001), *likelihood approximation* (Seeger et al., 2003; Snelson and Ghahramani, 2006, 2007) and *localized regression* (Tresp, 2000; Schwaighofer et al., 2003; Urtasun and Darrell, 2008; Rasmussen and Ghahramani, 2002; Gramacy and Lee, 2008; Chen and Ren, 2009; Park et al., 2011).

Many of the computation methods have been implemented as software, which includes SOGP[1], GPML[2], SPGP[3], TGP[4] and GPStuff[5]. However, many of the methods are still not implemented, because of various complexities involved in the methods as well as in their implementation. In particular, most of the localized regression methods are not implemented in spite of their unique advantages such as adaptivity to non-stationary changes and easiness of being parallelized for faster computation.

The GPLP is the Octave and Matlab implementation of several localized regression methods: the domain decomposition method (Park et al., 2011, DDM), partial independent conditional (Snelson and Ghahramani, 2007, PIC), localized probabilistic regression (Urtasun and Darrell, 2008, LPR), and bagging for Gaussian process regression (Chen and Ren, 2009, BGP). Most of the localized regression methods can be applied for general machine learning problems although DDM is only applicable for spatial data sets. In addition, the GPLP provides two parallel computation versions of the domain decomposition method. The easiness of being parallelized is one of the advantages of the localized regression, and the two parallel implementations will provide a good guidance about how to materialize this advantage as software.

This manual is written in *Getting-started* style; it introduces the working environment of GPLP (in Section 2) and illustrates the usage with an simple example (in Section 3). If you need more detailed documentation, please refer to *User Manual* at ./doc directory.

2. Implementation

The GPLP is implemented in Matlab code such that it is executable and has been tested in Matlab Version 7.7 or later versions, and Octave Version 3.2.4 or later versions. It might be executable in any of Matlab Version 7.x and any of Octave Version 3.2.x, but it has not been tested on those versions. One exception is the implementation of LPR that only works in Matlab 7.12.0, in Matlab 7.7.0 or later versions with a compiler supporting mex-compile, or in Octave 3.2.4 or later versions. For information on the list of compilers to support the mex-compile in Matlab, please refer to the technical support webpage at http://www.mathworks.com/support/compilers/previous_releases.html.

The GPLP also includes the parallel computation version of DDM, which requires the open source message passing interface, MatMPI Version 1.2, to be pre-installed before executing the parallel version. All of the Matlab, Octave and MatMPI are working in many versions of Windows and Unix, so GPLP is virtually OS-independent.

The implementation consists of six different main modules for the six different methods implemented, but all of the main modules are structured in the common form having the similar input and output arguments. In addition, the implementation partially supports the separation of the main

1. Implementation of SOGP (Smola and Bartlett, 2001) is available at http://cs.brown.edu/people/dang/code.shtml.
2. Implementation of GPML (Williams and Seeger, 2000) is available at http://gaussianprocess.org/gpml/code/matlab/doc/index.html.
3. Implementation of SPGP (Snelson and Ghahramani, 2006) is available at http://www.gatsby.ucl.ac.uk/~snelson.
4. Implementation of TGP (Gramacy and Lee, 2008) is available at http://users.soe.ucsc.edu/~rbgramacy/tgp.html.
5. Implementation of GPStuff (Snelson and Ghahramani, 2006, 2007; Schwaighofer et al., 2003) is available at http://www.lce.hut.fi/research/mm/gpstuff/.

logic from the specification of the covariance function and the mesh generation function (the specification of mesh generation function is only applicable for DDM and its two parallel computation versions; the explanation of the mesh generation function will be in the next section). With such separation, users can easily extend the function of `GPLP` by adding a new covariance function and adding a new mesh generation function without major modification of the main logic.

The code and documentation of `GPLP` are publicly available on the JMLR MOSS website at `http://www.jmlr.org/mloss` under GNU General Public License version 3.0 (GPL-3.0).

3. GPLP: A software Package for Localized and Parallel Computation of GP Regression

The `GPLP` provides an individual function for calling each one of the six localized regression methods (including two parallel implementations. The individual functions have a common structure of input and output arguments so that users can easily use all functions once they learn the common structure. In this section, we will explain the common structure by means of a simple example.

Consider a unknown random function $f : \mathcal{X} \to \mathbb{R}$. The GP regression predicts the realization of the random function at test locations `xs`, given a set of observations `x` from the realization. The localized GP regression partitions `x` into many smaller chunks, `x_j`'s, and it does localized predictions at `xs` with each one of `x_j`'s as the training data for every j. Finally, the localized GP regression combines the localized predictions to make a global prediction in many different ways. The key design parameters for the localized GP regression are (1) mean function and covariance function defining the GP, and (2) mesh generation function for partitioning `x` into `x_j`'s.

```
1  % define the structure of local regions
2  param1.meshfunc = 'rectMesh';   % mesh generation function
3  param1.mparam   = [14 21];      % mesh generation function parameters
4  param1.p = 3; param1.q = 3;     % parameters defining the interaction
5                                  %   between local regions for improving
6                                  %   prediction accuracy
7
8  % set the prior GP by specifying a covariance function
9  param2.covfunc  = {'covSum', {'covSEard','covNoise'}}; %covariance function
10 D               = size(x, 2);
11 logtheta0       = log(ones(D+2,1));
12 logtheta0(D+2)  = log(0.3);
13 param2.logtheta0 = logtheta0; % initial value of log hyperparameters
14 param2.frachyper = 0.5; % fraction of training data used for learning
15                         %                                   hyperparameters
16 param2.nIter     = 100; % maximum number of iterations in optimizing the
17                         %    log hyperparameters
18
19 % train the localized regression model for Gaussian process regression
20 [model, elpasedTrain] = ddmGP(x, y, param1, param2);
21
22 % predict at test inputs
23 [meanPred, varPred, elapsedPred] = ddm_pred(model, xs);
```

In line 2 and 3, we specify the mesh generation function as `rectMesh` with its input parameter `(14, 21)`. The mesh generation function decomposes $\mathcal{X}$ (domain of f) into 14-by-21 rectangular meshes, $\{\mathcal{X}_j\}$, and it partitions `x` into `x_j`'s such that `x_j` belongs to $\mathcal{X}_j$.

In line 4, there are two parameters that defines how many localized predictions are combined to produce a global prediction. In the domain decomposition method (DDM), a localized prediction is available for each mesh $\mathcal{X}_j$, which becomes the global prediction if the test input is in the interior of local domain $\mathcal{X}_j$. If the test input is over the common boundary of $\mathcal{X}_j$ and $\mathcal{X}_k$, the localized predictions are constrained by two factors: (1) the two local predictions for both of $\mathcal{X}_j$ and $\mathcal{X}_k$ should have limited degrees of freedom on the common boundary (called *flexibility of boundary prediction*); and (2) the two localized predictions should produce the same values on the boundary (called *consistency of boundary prediction*). The `param1.q` is the the number of control points on the boundary where the DDM checks the *consistency of boundary prediction*, and the `param1.p` is the number of degrees of freedom to constrain the *flexibility of boundary prediction*.

In line 9 through 13, we specify the `covSum` composite covariance function. The composite covariance function generates the covariance by summing two base covariance functions: the anisotropic version of squared exponential covariance function (`covSEard`) and the noise covariance function (`covNoise`). The `covSEard` is parameterized by $(D+1)$ hyperparameters as follows:

$$K(\mathbf{x},\mathbf{x}') = \theta_{D+1}^2 \exp\left\{-\frac{1}{2}\sum_{d=1}^{D}\left(\frac{x_d - x'_d}{\theta_d}\right)^2\right\},$$

where D is the dimension of $\mathcal{X}$. The `covNoise` is parameterized by noise variance parameter σ^2 as $K(\mathbf{x},\mathbf{x}') = \sigma^2\delta(\mathbf{x},\mathbf{x}')$. In total, the composite covariance function is parameterized by $(D+2)$ parameter values, so the initial guess of hyperparameter, `logtheta0`, should be $(D+2)$-dimensional. In line 11 and 12, the first $(D+1)$ elements of `logtheta0` are initialized for the hyperparameter values of `covSEard`, and the last one element of `logtheta0` is initialized for the value of σ^2.

In line 14 and 15, `param2.frachyper` and `param2.nIter` are the process parameters used in maximizing the likelihood function with respect to the hyperparameters. The maximization is an iterative process that updates the log hyperparameter values, starting with the initial guess `logtheta0`. The `param2.nIter=100` implies that the number of the iterations allowed for the iterative maximization is at most one hundred. In each iteration, the likelihood function is evaluated. Since the evaluation is computationally expensive with big size of training data, people usually uses only a subset of the training data for the evaluation. The `param2.frachyper = 0.5` implies that only half of the training data `x` will be used for the evaluation of the likelihood function.

Last, in line 20, the function `ddmGP` trains the domain decomposition method for the localized GP regression with training data set `x` and the previously specified parameters, and `ddmGP` returns the trained model (`model`) and the elapsed time (`elpasedTrain`). The number of the parameters to be specified depends on the method used for the training. For more details, please refer to *User Manual* at `./doc` directory in this package. In line 22, the function `ddm_pred` produces the mean prediction `meanPred` and the variance prediction `varPred` at test locations `xs`, and also reports the time used for prediction (`elpasedPred`).

References

Tao Chen and Jianghong Ren. Bagging for Gaussian process regression. *Neurocomputing*, 72(7-9): 1605–1610, 2009.

Robert B. Gramacy and Herbert K. H. Lee. Bayesian treed Gaussian process models with an application to computer modeling. *Journal of the American Statistical Association*, 103(483):1119–

1130, 2008.

Chiwoo Park, Jianhua Z. Huang, and Yu Ding. Domain decomposition approach for fast gaussian process regression of large spatial data sets. *Journal of Machine Learning Research*, 12:1697–1728, 2011.

Carl E. Rasmussen and Zoubin Ghahramani. Infinite mixtures of Gaussian process experts. In *Advances in Neural Information Processing Systems 14*, pages 881–888. MIT Press, 2002.

Carl E. Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

Anton Schwaighofer, Marian Grigoras, Volker Tresp, and Clemens Hoffmann. Transductive and inductive methods for approximate Gaussian process regression. In *Advances in Neural Information Processing Systems 16*, pages 977–984. MIT Press, 2003.

Matthias Seeger, Christopher K. I. Williams, and Neil D. Lawrence. Fast forward selection to speed up sparse Gaussian process regression. In *International Workshop on Artificial Intelligence and Statistics 9*. Society for Artificial Intelligence and Statistics, 2003.

Alexander J. Smola and Peter L. Bartlett. Sparse greedy Gaussian process regression. In *Advances in Neural Information Processing Systems 13*, pages 619–625. MIT Press, 2001.

Edward Snelson and Zoubin Ghahramani. Sparse Gaussian processes using pseudo-inputs. In *Advances in Neural Information Processing Systems 18*, pages 1257–1264. MIT Press, 2006.

Edward Snelson and Zoubin Ghahramani. Local and global sparse Gaussian process approximations. In *International Conference on Artifical Intelligence and Statistics 11*, pages 524–531. Society for Artificial Intelligence and Statistics, 2007.

Volker Tresp. A Bayesian committee machine. *Neural Computation*, 12(11):2719–2741, 2000.

Raquel Urtasun and Trevor Darrell. Sparse probabilistic regression for activity-independent human pose inference. In *IEEE Conference on Computer Vision and Pattern Recognition 2008*, pages 1–8, 2008.

Christopher K. I. Williams and Matthias Seeger. Using the Nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems 12*, pages 682–688. MIT Press, 2000.