Sparse Gaussian Processes: sgp

Carl Edward Rasmussen

April 1st, 2016

This document describes an implementation of two methods for sparse approximate inference in Gaussian Processes (GPs), the Fully Independent Training Conditionals (FITC) [1] and a Variational Free Energy approximation (VFE) [2].

The implementation assumes that the covariance function is squared exponential, and that the mean function is linear (for brevity, the mean function is not included in the documentation).

1 Log Marginal Likelihood

In the sparse approximation, the approximate log marginal likelihood is given by

$$\log(q(\mathbf{y}|u)) = -\tfrac{1}{2}\mathbf{y}^\top(Q+G)^{-1}\mathbf{y} - \tfrac{1}{2}\log|Q+G| - \tfrac{1}{2\sigma_n^2}\operatorname{tr}(T) - \tfrac{n}{2}\log(2\pi), \tag{1}$$

where

$$Q_{f,f} = K_{f,u}K_{u,u}^{-1}K_{u,f}, \tag{2}$$

$$G_{\mathrm{FITC}} = \operatorname{diag}[K_{f,f} - Q_{f,f}] + \sigma_n^2 I, \quad \text{and} \quad G_{\mathrm{VFE}} = \sigma_n^2 I, \tag{3}$$

$$T_{\mathrm{FITC}} = 0, \quad \text{and} \quad T_{\mathrm{VFE}} = K_{f,f} - Q_{f,f}. \tag{4}$$

Note that all elements of G are bounded below by σ_n^2, because K is positive definite. Rewriting $Q_{f,f} = V^\top V$, where $V = L^{-1}K_{u,f}$ and $K_{u,u} = LL^\top$, and using the matrix inversion lemma, the approximate log marginal likelihood can be written as

$$\begin{aligned}\log(q(\mathbf{y}|u)) &= -\tfrac{1}{2}\mathbf{y}^\top(G^{-1} - G^{-1}V^\top A^{-1}VG^{-1})\mathbf{y} - \tfrac{1}{2}\log|A| - \tfrac{1}{2}\log|G| - \tfrac{1}{2\sigma_n^2}\operatorname{tr}(T) - \tfrac{n}{2}\log(2\pi)\\ &= -\tfrac{1}{2}\mathbf{y}^\top\mathbf{z} - \tfrac{1}{2}\log|A| - \tfrac{1}{2}\log|G| - \tfrac{1}{2\sigma_n^2}\operatorname{tr}(T) - \tfrac{n}{2}\log(2\pi)\end{aligned} \tag{5}$$

where $A = I + VG^{-1}V^\top$ and $\mathbf{z} = (Q+G)^{-1}\mathbf{y} = (G^{-1} - G^{-1}V^\top A^{-1}VG^{-1})\mathbf{y}$.

1 ⟨*nlml* 1⟩≡ (3a)

```
l = exp(hyp(e).l); s2 = exp(2*hyp(e).s); n2 = exp(2*hyp(e).n);
u = bsxfun(@rdivide, induce, l');                    % scaled inducing inputs
x = bsxfun(@rdivide, inputs, l');                    % scaled training inputs
Kuu = s2*(exp(-maha(u,u)/2) + ridge*eye(M));
Kuf = s2*exp(-maha(u,x)/2);
L = chol(Kuu)';
V = L\Kuf;
r = s2 - sum(V.*V,1)';              % diagonal residual Kff - Kfu Kuu^-1 Kuf
G = fitc*r + n2; iG = 1./G;
A = eye(M) + V*bsxfun(@times,iG,V');
J = chol(A)';
B = J\V;
z = iG.*y - (y'.*iG'*B'*B.*iG')';
nlml = nlml + y'*z/2 + sum(log(diag(J))) + sum(log(G))/2 ...
                                  + vfe*sum(r)/n2/2 + N*log(2*pi)/2;
```

2 Derivatives

The derivative of the approximate log marginal likelihood wrt parameters θ

$$\frac{\partial \log(q(y|u))}{\partial\theta} = \frac{\partial \log q}{\partial Q}\frac{\partial Q}{\partial\theta} + \frac{\partial \log q}{\partial G}\frac{\partial G}{\partial\theta} + \frac{\partial \log q}{\partial T}\frac{\partial T}{\partial\theta}, \quad \text{where} \quad q \triangleq q(y|u), \tag{6}$$

where θ could be either the inducing inputs u or parameters of the covariance function (hyperparameters). We have

$$2\frac{\partial \log q}{\partial Q} = \mathbf{z}\mathbf{z}^\top - G^{-1} + G^{-1}V^\top A^{-1}VG^{-1}, \quad \text{and} \quad \frac{\partial \log q}{\partial G} = \operatorname{diag}\Big[\frac{\partial \log q}{\partial Q}\Big]. \tag{7}$$

Note that we cannot compute $\frac{\partial \log q}{\partial Q}$ itself, as the cost would be prohibitive (N^2M).

2.1 Inducing inputs

We need the derivatives of Q, G_{FITC} and T_{VFE} (ignoring the trivial G_{VFE} and T_{FITC})

$$\begin{aligned}
\frac{\partial Q}{\partial u} &= 2K_{f,u}K_{u,u}^{-1}\frac{\partial K_{u,f}}{\partial u} - K_{f,u}K^{-1}\frac{\partial K_{u,u}}{\partial u}K^{-1}K_{u,f} = 2\operatorname{sym}(R^\top P), && (8)\\
\frac{\partial G_{\text{FITC}}}{\partial u} &= -2\operatorname{diag}(R^\top P), && (9)\\
\frac{\partial T_{\text{VFE}}}{\partial u} &= -2\operatorname{tr}(R^\top P), && (10)
\end{aligned}$$

where $\operatorname{sym}(A) = \frac{1}{2}(A + A^\top)$ and we have defined

$$R = K_{u,u}^{-1}K_{u,f} \quad \text{and} \quad P = \frac{\partial K_{u,f}}{\partial u} - \frac{\partial \tilde{K}_{u,u}}{\partial u}R, \tag{11}$$

and the derivative of $\tilde{K}_{u,u}$ denotes the derivative taken with respect to only the first argument of K (as the derivative wrt the second argument is just the transpose of the derivative wrt the first argument). Of course, in an actual implementation, the product $R^\top P$ in eq. (8) should never explicitly be computed (as this would cost N^2M operations).

2 ⟨*deriv 2*⟩≡ (3a)

```
R = L'\V;
RiG = bsxfun(@times,R,iG');
RdQ = -R*z*z' + RiG - bsxfun(@times,RiG*B'*B,iG');
dG = z.^2 - iG + iG.^2.*sum(B.*B,1)';
RdQ2 = RdQ + bsxfun(@times, R, fitc*dG' - vfe/n2);
KW = Kuf.*RdQ2;
KWR = Kuu.*(RdQ2*R');
P = KW*x + bsxfun(@times, sum(KWR, 2) - sum(KW, 2), u) - KWR*u;
dnlml.induce = dnlml.induce + bsxfun(@rdivide, P, l');
dnlml.hyp(e).l = -sum(P.*u,1) ...
                        - sum((KW'*u - bsxfun(@times, sum(KW',2), x)).*x,1);
dnlml.hyp(e).n = -sum(dG)*n2 - vfe*sum(r)/n2;
dnlml.hyp(e).s = sum(sum(Kuf.*RdQ)) - fitc*r'*dG + vfe*sum(r)/n2;
dnlml.hyp(e).b = -sum(z);
dnlml.hyp(e).m = -inputs'*z;
```

2.2 Hyperparameters

For the log lengthscale hyperparameter

$$\frac{\partial Q}{\partial \log \ell} = \tag{12}$$

3a ⟨*sgp.m* 3a⟩≡

```
function [nlml, dnlml] = sgp(p, inputs, target, style, test)

ridge = 1e-06;                  % relative jitter to make matrix better conditioned
switch style, case 'fitc', fitc = 1; vfe = 0; case 'vfe', vfe = 1; fitc = 0; end
induce = p.induce; hyp = p.hyp;                                % shorthand notation
[N, D] = size(inputs); M = size(induce,1); nlml = 0; dnlml.induce = zeros(M, D);

for e = 1:length(hyp)
  y = target(:,e) - inputs*hyp(e).m - hyp(e).b;
  ⟨nlml 1⟩
  if nargin == 5                                                 % make predictions
    ⟨predict 3b⟩
  elseif nargout == 2                                         % compute derivatives
    ⟨deriv 2⟩
  end
end
```

3 Predictions for deterministic test inputs

For efficiency, predictions are made based on pre-computation of the quantities which don't depend on the test cases. These quanteties are β and W. The Gaussian predictions

$$\mu = k(x^*, x)\beta, \quad \text{and} \quad \sigma^2 = k(x^*, x^*) - k(x^*, x)Wk(x, x^*). \tag{13}$$

The definition of β and W depend on the inference method being used. For the full GP we have

$$\beta = (K + \sigma_n^2 I)^{-1}\mathbf{y}, \quad \text{and} \quad W = (K + \sigma_n^2 I)^{-1}. \tag{14}$$

For the sparse FITC and VFE method we have

$$\beta = (K_{u,u} + K_{u,f}G^{-1}K_{f,u})^{-1}K_{u,f}G^{-1}\mathbf{y}, \quad \text{and} \quad W = K_{u,u}^{-1} - (K_{u,u} + K_{u,f}G^{-1}K_{f,u})^{-1}. \tag{15}$$

3b ⟨*predict* 3b⟩≡ (3a)

```
beta = (y'.*iG'*B'/J/L)';
W = L'\(eye(M)-eye(M)/J'/J)/L;
Ktu = s2*exp(-maha(bsxfun(@rdivide,test,l'), u)/2);
nlml = Ktu*beta + test*hyp.m + hyp.b;
dnlml = s2+n2-sum(Ktu*W.*Ktu,2);
```

References

[1] Edward Snelson and Zoubin Ghahramani. Sparse Gaussian processes using pseudo-inputs. In Y. Weiss, B. Schölkopf, and J. Platt, editors, Advances in Neural Information Processing Systems 18, pages 1257-1264. The MIT Press, Cambridge, MA, 2006.

[2] Michalis K. Titsias. Variational Learning of Inducing Variables in Sparse Gaussian Processes. Twelfth International Conference on Artificial Intelligence and Statistics, (AISTATS), JMLR: W&CP 5, pp. 567-574, 2009.