2019-06-06

1 The basic tasks

In the past two lectures, we have discussed various interpretations of kernels and how they are used for interpolation and regression. In today's lecture, we focus on the numerical methods used to fit kernel-based models and to use them for prediction and uncertainty estimation. More specifically, we consider four problems:

- For a fixed kernel, how do we fit the *parameters* for a kernel-based model? That is, how do we compute c such that

  $$\bar{f}(x) = \sum_{j=1}^{n} c_j k(x, x_j)$$

  approximately satisfies $\hat{f}(x_i) \approx y_i$, where (x_i, y_i) are our observed data points?

- Many kernels involve *hyper-parameters*, such as length scale and smoothness parameters. How should we determine these hyper-parameters?

- How do we quickly evaluate the predicted values $\hat{f}(x)$ at new points x?

- How do we quickly evaluate measures that quantify uncertainty, such as the predictive variance?

Because fitting model parameters and hyper-parameters is more costly than evaluating the predicted mean field or predictive variance, we will focus most of our attention on the fitting tasks. We will also focus primarily on the probabilistic interpretation of kernels in terms of Gaussian processes. Because we have in mind downstream tasks in which a user can adaptively obtain new data (as opposed to fitting to a fixed data set), we will also discuss incremental fitting.

2 Learning at small n

We initially consider the case where there are few enough data points that it makes sense to use standard direct factorization methods to solve the

fitting problem. These standard factorizations require $O(n^3)$ time, but if we use well-tuned libraries, the constants need not be large. Using a standard laptop, we can work with data sets of size greater than $n = 1000$ as a matter of routine. The direct factorization approach is also useful as a starting point for scalable algorithms more appropriate to large n.

2.1 Cholesky and kernel-only fitting

The Cholesky factorization of a positive definite kernel matrix K is

$$K = R^T R$$

where R is upper triangular. The fitting system in the case of a positive definite kernel with no tail is

$$Kc = y,$$

and the standard solution algorithm is a Cholesky factorization of K followed by two triangular solves

$$c = R^{-1}(R^{-T}y).$$

The Cholesky factorization costs $O(n^3)$; the triangular solves each cost $O(n^2)$. Hence, the cost is dominated by the cost of the Cholesky factorization.

For computation, it is useful to think of the decomposition in block terms

$$\begin{bmatrix} K_{11} & K_{12} \\ K_{12}^T & K_{22} \end{bmatrix} = \begin{bmatrix} R_{11}^T & 0 \\ R_{12}^T & R_{22}^T \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{bmatrix}.$$

Rearranging block-by-block, we have

$$\begin{aligned} K_{11} &= R_{11}^T R_{11} & R_{11}^T R_{11} &= K_{11} \\ K_{12} &= R_{11}^T R_{12} & R_{12} &= R_{11}^{-T} K_{12} \\ K_{22} &= R_{12}^T R_{12} + R_{22}^T R_{22} & R_{22}^T R_{22} &= K_{22} - R_{12}^T R_{12} \end{aligned}$$

That is, we can think of the decomposition in terms of a decomposition of the leading submatrix of K, a triangular solve to get an off-diagonal block, and then decomposition of the trailing *Schur complement*

$$S_{22} = R_{22}^T R_{22} = K_{22} - R_{12}^T R_{12} = K_{22} - K_{21} K_{11}^{-1} K_{12}.$$

The Schur decomposition has a meaning independent of the Cholesky factorization, as we can see by solving the system

$$\begin{bmatrix} R_{11}^T & 0 \\ R_{12}^T & R_{22} \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 0 \\ I \end{bmatrix}.$$

Block forward substitution gives

$$\begin{bmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 0 \\ R_{22}^{-T} \end{bmatrix}$$

and block back substitution then yields

$$\begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} -R_{11}^{-1} R_{12} R_{22}^{-1} R_{22}^{-T} \\ R_{22}^{-1} R_{22}^{-T} \end{bmatrix} = \begin{bmatrix} -S_{11}^{-1} K_{12} S_{22}^{-1} \\ S_{22}^{-1} \end{bmatrix}.$$

That is, the inverse of the Schur complement is a submatrix of the inverse of the original matrix K: $S_{22}^{-1} = \left[K^{-1}\right]_{22}$. In the Gaussian process setting, if K is the covariance matrix, then K^{-1} is the *precision matrix*, and S_{22}^{-1} is the precision matrix of the posterior for the second block of variables conditioned on observations of data points from the first block.

In introductory numerical methods courses, one typically first encounters the (scalar[1]) *right-looking* version of the Cholesky algorithm:

```
% Overwrite K with the Cholesky factor R
for j = 1:n
  K(j,j) = sqrt(K(j,j));
  K(j,j+1:end) = K(j,j+1:end) / K(j,j);
  K(j+1:end,j) = 0;
  K(j+1:end,j+1:end) = K(j+1:end,j+1:end) - K(j,j+1:end)'*K(j,j+1:end);
end
```

At step j of the loop, we have computed the first $j-1$ rows of R, and the trailing submatrix contains the Schur complement. At the end of the step, we perform a rank 1 update to get a new (smaller) Schur complement matrix.

An alternate organization, the *left-looking* version, defers the Schur complement update until it is needed:

[1]The *scalar* version of the algorithm works one column at a time. Various *blocked* versions of the factorization algorithm update in blocks of several columns at a time, with a small (scalar) Cholesky factorization for the diagonal blocks. Block algorithms have the same complexity as the corresponding scalar algorithms, but achieve better performance on modern hardware because they make better use of the cache.

```
% Overwrite K with the Cholesky factor R
K(1,1) = sqrt(K(1,1));
K(2:end,1) = 0;
for j = 2:n
  K(1:j-1,j) = K(1:j-1,1:j-1) \ K(1:j-1,j)
  K(j,1:j-1) = 0;
  K(j,j) = sqrt(K(j,j) - K(1:j-1,j)'*K(1:j-1,j));
end
```

Both organizations of Cholesky do the same operations, but in a different order. Both require $O(n^3)$ time; the dominant cost per step for the right-looking variant is the rank-1 update of the Schur complement, where the dominant cost per step for the left-looking variant is the triangular solve.

The advantage of the left-looking factorization is that it can be applied *incrementally*. That is, suppose that K_{11} corresponds to the kernel matrix associated with an initial sample of data points, and we have computed $K_{11} = R_{11}^T R_{11}$. Then to add a second set of data points, we can do a left-looking block update

$$R_{12} = R_{11}^{-T} K_{12}$$
$$R_{22}^T R_{22} = K_{22} - R_{12}^T R_{12}$$

The cost of this update is $O((n_1^2 + n_2^2)n_2)$, which is significantly less than the $O((n_1 + n_2)^3)$ cost of recomputing the factorization from scratch in the case that $n_2 \ll n_1$.

2.2 Fitting with a tail

We now consider fitting in the presence of a tail term. Polynomial tails are typically used with conditionally positive definite kernel functions, but they can also be used with positive definite kernels — and indeed they usually are used in some geostatistical applications. One can also incorporate non-polynomial terms into the tail if they are useful to the model.

The fitting problem with a tail looks like

$$\hat{f}(x) = \sum_i c_i k(x, x_i) + \sum_j d_j p_j(x)$$

where the coefficients d satisfy the discrete orthogonality condition

$$\sum_i p_j(x_i) d_i = 0$$

for each basis function $p_j(x)$. This gives us the linear system

$$\begin{bmatrix} K & P \\ P^T & 0 \end{bmatrix} \begin{bmatrix} c \\ d \end{bmatrix} = \begin{bmatrix} y \\ 0 \end{bmatrix}.$$

We can also see this linear system from the perspective of constrained optimization: we are minimizing a quadratic objective (associated with K) subject to linear constraints (associated with P). Note that this formulation is only well posed if P is full rank, a condition sometimes known as unisolvency of the interpolation points.

One way to deal with the tail term is the "null space" approach; that is, rather than adding equations that enforce the discrete orthogonality constraint, we find a coordinate system in which the discrete orthogonality constraint is automatic. Specifically, suppose we write the full QR decomposition of P as

$$P = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R_1 \\ 0 \end{bmatrix}.$$

Then the constraint $P^T c = 0$ can be rewritten as $c = Q_2 w$, giving us

$$Q_2^T K Q_2 w = Q_2^T y$$

where $Q_2^T K Q_2$ is generally symmetric and positive definite even for a conditionally positive definite kernel. Once we have computed w (and from there c), we can compute d by the relation

$$R_1 d = Q_1^T (y - Kc).$$

An alternate approach is to partition the data points into two groups, the first of which is unisolvent and has the same size as the dimension of the tail. Then we can write any c satisfying $P^T c = 0$ in the form

$$\begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} -W^T \\ I \end{bmatrix} c_2$$

where $W = P_2 P_1^{-1}$. Substituting this into the constrained optimization of $c^T K c$ gives the reduced problem

$$\tilde{K}_{22} c_2 = (K_{22} - W K_{12} - K_{21} W^T + W K_{11} W) c_2 = y_2,$$

from which we can recover the remaining coefficients by solving

$$\begin{aligned} c_1 &= -W^T c_2 \\ P_1 d &= y_1 - K_{11} c_1 - K_{12} c_2. \end{aligned}$$

This reduction is formally equivalent to Gaussian elimination and substitution on the system

$$\begin{bmatrix} 0 & P_1^T & P_2^T \\ P_1 & K_{11} & K_{12} \\ P_2 & K_{21} & K_{22} \end{bmatrix} \begin{bmatrix} d \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 0 \\ y_1 \\ y_2 \end{bmatrix}.$$

As with the left-looking factorization described in the previous section, we can form W, K, $\tilde{K}_{22}$, and the Cholesky factorization of $\tilde{K}_{22}$ incrementally as new data points are added.

2.3 Likelihoods and gradients

Recall from the last lecture that the log likelihood function for a (mean zero) Gaussian process is

$$\mathcal{L} = -\frac{1}{2} y^T K^{-1} y - \frac{1}{2} \log\det K - \frac{n}{2} \log(2\pi).$$

Given a Cholesky factorization $K = R^T R$, we can rewrite this as

$$\mathcal{L} = -\frac{1}{2} \|R^{-T} y\|^2 - \log\det R - \frac{n}{2} \log(2\pi),$$

and note that

$$\log\det R = \sum_i \log r_{ii}.$$

The cost of evaluating the log likelihood is dominated by the cost of the initial Cholesky factorization.

In order to optimize kernel hyper-parameters via maximum likelihood, we would also like to compute the gradient (and maybe the Hessian) with respect to the hyper-parameters. Recall from last time that we computed the derivative

$$\delta\mathcal{L} = \frac{1}{2} c^T [\delta K] c - \frac{1}{2} \operatorname{tr}(K^{-1} \delta K)$$

where $Kc = y$. Unfortunately, I know of no tricks to exactly compute $\mathrm{tr}(K^{-1}\delta K)$ for arbitrary δK without in time less than $O(n^3)$ without exploiting additional structure beyond what we have discussed so far.

Simply computing gradients of the log likelihood is sufficient for gradient descent or quasi-Newton methods such as BFGS. However, if the number of hyper-parameters is not too great, we may also decide to compute second derivatives and compute a true Newton iteration. Let θ be the vector of hyper-parameters and use $[f]_{,j}$ to denote the partial derivative of an expression f with respect to θ_j; then

$$\begin{aligned}
\left[y^T K^{-1} y\right]_{,ij} &= \left[-y^T K^{-1} K_{,i} K^{-1} y\right]_{,j} \\
&= 2y^T K^{-1} K_{,i} K^{-1} K_{,j} K^{-1} y - y^T K^{-1} K_{,ij} K^{-1} y \\
&= 2c^T K_{,i} K^{-1} K_{,j} c - c^T K_{,ij} c \\
[\log\det K]_{,ij} &= \left[\mathrm{tr}\left(K^{-1} K_{,i}\right)\right]_{,j} \\
&= \mathrm{tr}\left(K^{-1} K_{,ij} - K^{-1} K_{,i} K^{-1} K_{,j}\right) \\
\mathcal{L}_{,ij} &= \frac{1}{2} c^T K_{,ij} c - c^T K_{,i} K^{-1} K_{,j} c - \frac{1}{2}\mathrm{tr}\left(K^{-1} K_{,ij} - K^{-1} K_{,i} K^{-1} K_{,j}\right).
\end{aligned}$$

Barring tricks that take advantage of further structure in the kernel matrix K or its derivatives, computing these second partials has the same $O(n^3)$ complexity as computing the first derivatives.

2.4 Optimizing the nugget

An important special case is optimization with only a single hyper-parameter, the noise variance term or "nugget" term[2] That is, we seek to maximize

$$\mathcal{L}(\eta) = -\frac{1}{2} y^T (K + \eta I)^{-1} y - \frac{1}{2}\log\det(K + \eta I) - \frac{n}{2}\log(2\pi).$$

In this case, rather than doing a Cholesky factorization, we may choose to compute an eigenvalue decomposition $K = Q\Lambda Q^T$ in order to obtain

$$\mathcal{L}(\eta) = -\frac{1}{2}\hat{y}^T (\Lambda + \eta I)^{-1} \hat{y} - \frac{1}{2}\log\det(\Lambda + \eta I) - \frac{n}{2}\log(2\pi)$$

[2]The term "nugget" comes from the use of Gaussian process regression in geostatistical applications. When trying to use Gaussian processes to predict the location of precious mineral deposits based on the mineral content in bore measurements, it is necessary to account for the possibility that a measurement may accidentally happen on a mineral nugget.

where $\hat{y} = Q^T y$. In this case, the stationary points satisfy a rational equation in η:

$$\sum_{j=1}^{n} \left(\frac{\hat{y}_j^2}{(\lambda_j + \eta)^2} - \frac{1}{2(\lambda_j + \eta)} \right) = 0.$$

We can run Newton on this equation in $O(n)$ time per step.

3 Smooth kernels and low-rank structure

The cost of parameter fitting for a general kernel matrix is $O(n^3)$, and the cost of hyper-parameter fitting by a gradient-based method applied to maximization of the log-likelihood involves an additional $O(n^3)$ cost per hyper-parameter per step. However, for *smooth* kernels, the kernel matrix (without a nugget term) may be effectively very low rank. This case is of enough interest that we give it special treatment here. More specifically, we assume a kernel matrix of the form

$$\tilde{K} = K + \eta I$$

where most of the eigenvalues of K are much less than η. In this case, we can solve the fitting problem and the computation of the log-likelihood and gradients in much less than $O(n^3)$ time.

3.1 Pivoted Cholesky and kernel approximation

We begin with an outer product factorization of the kernel matrix K without a regularizing nugget. Specifically, we consider the pivoted Cholesky factorization

$$\Pi^T K \Pi = R^T R$$

where the diagonal elements of R appear in descending order and the off-diagonal elements in each row of R are dominated by the diagonal element. The pivoted Cholesky factorization algorithm looks very much like the standard Cholesky algorithm, except before adding row j to the R factor, we first swap rows and columns j and $j_{\max}$ of K, where $j_{\max}$ is the index of the largest entry in the Schur complement matrix.

The approximation error associated with truncating the pivoted Cholesky factorization is just the Schur complement matrix; that is (suppressing the

permutation momentarily),

$$\begin{bmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{bmatrix} - \begin{bmatrix} R_{11}^T \\ R_{12}^T \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & K_{22} - R_{12}^T R_{12} \end{bmatrix}$$

Typically, we would stop the factorization when all the diagonal elements in the Schur complement are less than some tolerance. Because the Schur complement is positive definite, the sum of the diagonal elements in the Schur complement is the same as the nuclear norm of the Schur complement, so controlling the size of the largest diagonal element also controls the approximation error in the nuclear norm (which dominates both the Frobenius norm and the spectral norm).

Part of what makes pivoted Cholesky attractive is that we can run the algorithm *without ever forming* K — we only need the rows of K associated with the pivots selected up to the current step, plus a running computation of the diagonal of the Schur complement. If we have formed $j-1$ steps of the pivoted Cholesky factorization, we can extend the factorization by:

- Searching the Schur complement diagonal d to find the pivot row $j_{\max}$ and swapping points j and $j_{\max}$ and columns j and $j_{\max}$ of R (and d).
- Forming the top row of the current Schur complement using the formula

$$s_{jl} = k(x_j, x_l) - \sum_{i=1}^{j-1} r_{ij} r_{il}$$

- Extending the factorization with $s_{jj} = \sqrt{s_{jj}}$ and $r_{jl} = s_{jl}/r_{jj}$ for $l > j$.
- Updating the diagonal of the Schur complement via $d_{ll}^{\text{new}} = d_{ll} - r_{jl}^2$.

Running pivoted Cholesky in this way for r steps only requires $O(nr)$ kernel evaluations, $O(nr)$ storage, and $O(nr^2)$ time.

3.2 The effects of truncation error

We now consider the solution of the linear system

$$(K + \eta I)c = y$$

where ηI is the nugget term. Suppose we have a low-rank factorization (computed by truncated pivoted Cholesky, for example) of the form $K \approx WW^T$. Substituting gives us the approximate system

$$(WW^T + \eta I)\hat{c} = y.$$

How close are $\hat{c}$ and c? Let $K = WW^T + E$, i.e. E is the part of K discarded by truncating a decomposition of K. Then subtracting the approximate linear systems gives

$$(K + \eta I)(c - \hat{c}) = E\hat{c},$$

and norm bounds yield

$$\|c - \hat{c}\| \leq \|(K + \eta I)^{-1}E\|\|\hat{c}\| \leq \frac{\|E\|}{\eta}\|\hat{c}\|.$$

where we have throughout used the matrix two-norm. The matrix two-norm is bounded by the nuclear norm; and if W comes from a pivoted Cholesky factorization, we can compute the nuclear norm of E as the sum of the diagonal elements in the truncated Schur complement term. Hence, if we use pivoted Cholesky with the termination criterion

$$\|S\|_* \leq \delta\eta,$$

then we obtain the relative error bound

$$\frac{\|c - \hat{c}\|}{\|\hat{c}\|} \leq \delta$$

if all computations are done in exact arithmetic.

3.3 Solving with the kernel

To solve a system $(WW^T+\eta I)c = y$ efficiently, let us introduce a new variable $z = W^T c$; putting together the definition of z with the equation for c gives us

$$\begin{bmatrix} \eta I & W \\ W^T & -I \end{bmatrix} \begin{bmatrix} c \\ z \end{bmatrix} = \begin{bmatrix} y \\ 0 \end{bmatrix}.$$

Of course, we can eliminate z to get back to the original equation; but what happens if we instead eliminate c? Then we get the reduced system

$$(-I - \eta^{-1}W^T W)z = -\eta^{-1}W^T y.$$

Multiplying through by $-\eta$ gives

$$(W^T W + \eta I)z = W^T y,$$

Back substitution gives the formula

$$c = \frac{1}{\eta}\left(y - W(W^T W + \eta I)^{-1} W^T y\right).$$

This is a special case of the famous Sherman-Morrison-Woodbury formula for the inverse of a matrix (in this case $\sigma^2 I$) plus a low-rank modification.

While we can certainly work with the Sherman-Morrison-Woodbury formula directly, we can do a little better if we recognize that the equations in z alone are the regularized normal equations for the optimization

$$\text{minimize } \frac{1}{2}\|Wz - y\|^2 + \frac{\eta}{2}\|z\|^2.$$

Furthermore, we have that

$$c = \eta^{-1}(y - Wz);$$

that is, c is just the scaled residual of the least squares problem. In terms of an economy QR decomposition

$$\begin{bmatrix} W \\ \sqrt{\eta} I \end{bmatrix} = \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} R$$

we have

$$c = \eta^{-1}\left(y - Q_1(Q_1^T y)\right).$$

This computational approach (using an economy QR decomposition) enjoys a modest stability advantage compared to working directly with the Sherman-Morrison-Woodbury formula.

3.4 Evaluating the kernel approximant

We have so far treated the pivoted Cholesky approach as a way to approximate the kernel *matrix*. Another interpretation is that we have computed a semidefinite kernel function

$$\hat{k}(x, y) = k_{x\mathcal{I}} K_{\mathcal{I}\mathcal{I}}^{-1} k_{\mathcal{I}x}$$

where $\mathcal{I}$ refers to the r indices selected in the Cholesky factorization. These two interpretations lead to the same linear system to approximate the coefficient vector c, but they suggest two different approximations to the function:

$$s(x) = \sum_{i=1}^{n} k(x, x_i)c_i \quad \text{or} \quad \hat{s}(x) = \sum_{i=1}^{n} \hat{k}(x, x_i)c_i.$$

The two approximations agree for points x that are sufficiently close to the training data, but they may differ when x is farther away. The former approximation is the one that may have motivated us initially, but it is worth spending a moment on the second approximation.

Writing $\hat{s}$ in matrix terms, we have

$$\begin{aligned}\hat{s}(x) &= \hat{k}_{xX}(\hat{K}_{XX} + \eta I)^{-1}y \\ &= k_{xI}K_{\mathcal{I}\mathcal{I}}^{-1}K_{\mathcal{I}X}(K_{X\mathcal{I}}K_{\mathcal{I}\mathcal{I}}^{-1}K_{\mathcal{I}X} + \eta I)^{-1}y.\end{aligned}$$

Equivalently, we can write

$$\hat{s}(x) = k_{xI}d$$

where (omitting some algebra) d is the solution to the regularized least squares problem

$$\text{minimize } \frac{1}{2}\|K_{XI}\hat{d} - y\|^2 + \frac{\eta}{2}d^T K_{II} d.$$

If $K_{II} = R_{II}^T R_{II}$ is the leading part of the pivoted Cholesky factorization, we can rewrite this minimization as

$$\text{minimize } \frac{1}{2}\left\| \begin{bmatrix} K_{XI} \\ \sqrt{\eta}R_{II} \end{bmatrix} d - \begin{bmatrix} y \\ 0 \end{bmatrix} \right\|^2.$$

Alternately, we can think of the pivoted Cholesky factorization as inducing a feature map associated with the approximate kernel $\hat{k}$:

$$\hat{k}(x, y) = \psi(x)^T\psi(y), \qquad \psi(x) = R_{II}^{-T}k_{Ix}.$$

Let W denote the matrix of feature vectors (i.e. row i is $\psi(x_i)^T$); in the pivoted Cholesky factor, this is just $R_{I,:}^T$. Then the approximation system involves solving

$$\text{minimize } \frac{1}{2}\|Wz - y\| + \frac{\eta}{2}\|z\|^2,$$

which is the same minimization we saw in the previous subsection; and we can write the $\hat{s}(x)$ function as

$$\hat{s}(x) = \psi(x)^T z.$$

3.5 Predictive variance

Treating k as the covariance kernel of a Gaussian process, the predictive variance at a test point x (conditioned on noisy data at points X) is

$$v(x) = k(x,x) - k_{xX}(K_{XX} + \eta I)^{-1} k_{Xx}.$$

As in the previous section, we can again think about using low rank approximation in one of two ways: either we can use the low rank structure with the original kernel for the linear solve $(WW^T + \eta I)^{-1} k_{Xx}$; or we can think of the low rank factorization (via pivoted Cholesky) as defining the new kernel $\hat{k}$, for which the predictive variance is

$$\hat{v}(x) = k_{x\mathcal{I}}(K_{\mathcal{I}\mathcal{I}} + \eta^{-1} K_{\mathcal{I}X} K_{X\mathcal{I}})^{-1} k_{\mathcal{I}x}.$$

Using the Cholesky factorization $R_{\mathcal{I}\mathcal{I}}^T R_{\mathcal{I}\mathcal{I}} = K_{\mathcal{I}\mathcal{I}}$, we can rewrite

$$(K_{\mathcal{I}\mathcal{I}} + \eta^{-1} K_{\mathcal{I}X} K_{X\mathcal{I}})^{-1} = \eta \tilde{Q}_1^T \tilde{Q}_1$$

where $\tilde{Q}_1$ comes from the economy QR decomposition

$$\begin{bmatrix} K_{X\mathcal{I}} \\ \sqrt{\eta} R_{\mathcal{I}\mathcal{I}} \end{bmatrix} = \begin{bmatrix} \tilde{Q}_1 \\ \tilde{Q}_2 \end{bmatrix} \tilde{R},$$

which we may have already formed in the process of computing the coefficients in $\hat{s}(x) = k_{x\mathcal{I}} d$.

3.6 Likelihoods and gradients

Recall the log likelihood function is

$$\mathcal{L} = \frac{1}{2} y^T c - \frac{1}{2} \log\det \tilde{K} - \frac{n}{2} \log(2\pi)$$

where $\tilde{K} = K + \eta I$ is the regularized kernel matrix. The first and last terms need no special treatment in the case when the kernel matrix has special structure, so long as we are able to rapidly compute the coefficient vector c. The only non-obvious computation is the log determinant.

Consider the regularized kernel matrix $\tilde{K} = WW^T + \eta I$ where $W \in \mathbb{R}^{n \times r}$, and suppose we write a full SVD for W:

$$W = U \Sigma V^T = \begin{bmatrix} U_1 & U_2 \end{bmatrix} \begin{bmatrix} \Sigma_1 \\ 0 \end{bmatrix} V^T$$

Then

$$U^T \tilde{K} U = \begin{bmatrix} \Sigma^2 + \eta I & 0 \\ 0 & \eta I_2 \end{bmatrix}$$

Hence,

$$\log\det(\tilde{K}) = \log\det(U^T \tilde{K} U) = \log\det(\Sigma^2 + \eta I) + (n-r)\log(\eta).$$

If we use that $\Sigma^2 + \eta I$ is the eigenvalue matrix for $W^T W + \eta I = R^T R$, with R computed by the economy QR we used for kernel solves, we have

$$\log\det(K) = 2\sum_{j=1}^{r} \log(r_{jj}) - (n-r)\log\eta.$$

What of the derivative of the log likelihood? That is, we would like a fast method to compute

$$\delta L = \frac{1}{2} c^T (\delta\tilde{K}) c - \frac{1}{2}\operatorname{tr}(\tilde{K}^{-1}\delta\tilde{K}).$$

We can simplify the latter term by observing that

$$\tilde{K}^{-1} = \frac{1}{\eta}(I - Q_1 Q_1^T)$$

where

$$\begin{bmatrix} W \\ \sqrt{\eta} I \end{bmatrix} = \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} R;$$

substitution and using properties of traces gives us

$$\operatorname{tr}(\tilde{K}^{-1}\delta\tilde{K}) = \frac{1}{\eta}\left(\operatorname{tr}(\delta\tilde{K}) - \operatorname{tr}(Q_1^T \delta\tilde{K} Q_1)\right).$$

If we work with the original kernel, it is difficult to do better than this.

Now suppose we differentiate the log-likelihood for the approximate kernel induced by the points $\mathcal{I}$, i.e.

$$\tilde{K}_{XX} = K_{X\mathcal{I}} K_{\mathcal{I}\mathcal{I}}^{-1} K_{\mathcal{I}X} + \eta I.$$

Differentiating this expression gives us

$$\delta\tilde{K} = (\delta K_{:,\mathcal{I}}) F^T + F(\delta K_{:,\mathcal{I}})^T - F(\delta K_{\mathcal{I}\mathcal{I}}) F^T + (\delta\eta) I$$

where $F = K_{:,\mathcal{I}} K_{\mathcal{I}\mathcal{I}}^{-1}$. Therefore

$$\begin{aligned}
\frac{1}{2} c^T (\delta K) c &= (\delta K_{:,\mathcal{I}} c)^T (Fc) - \frac{1}{2} (Fc)^T (\delta K_{II})(Fc) + \frac{1}{2} \|c\|^2 \delta\eta \\
\frac{1}{2} \operatorname{tr}(\tilde{K}^{-1} \delta \tilde{K}) &= \langle \tilde{F}, \delta K_{:,\mathcal{I}} \rangle_F - \frac{1}{2} \langle \tilde{F}, F \delta K_{\mathcal{I}\mathcal{I}} \rangle_F + \delta\eta \left(n - \|Q_1\|_F^2 \right) \\
\tilde{F} &\equiv \frac{1}{\eta} (I - Q_1 Q_1^T) F
\end{aligned}$$

and so we can compute derivatives of the log likelihood in $O(nr)$ space and $O(nr^2)$ time in this approximation.

3.7 Beyond low rank

For a number of kernels, the kernel matrix is not all that low rank — the singular values decay, but not so quickly that we would be happy with $O(nr^2)$ time algorithms. However, in low-dimensional spaces, one may still have low-rank *submatrices*, and there are fast factorization techniques based on these rank-structured matrices as well. There are also a variety of iterative solvers that take advantage of fast matrix-vector multiplies; for example, we can use the method of conjugate gradients (CG) to solve linear systems using only matrix-vector products with a kernel matrix. For these types of solvers, the low rank approximations using pivoted Cholesky as described above still give us a good *preconditioner* that can accelerate convergence.