Optimizing Gaussian Processes

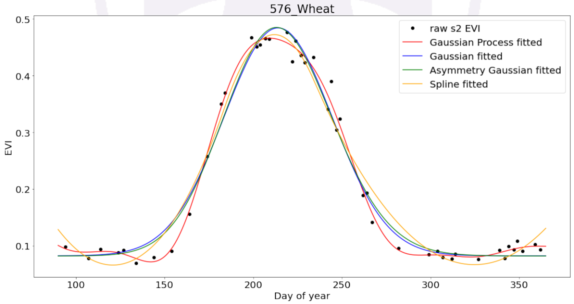
Honours Research Project

Michael Ciccotosto-Camp - 44302913

The University of Queensland
AUSTRALIA

Problem Setting and Motivation

- This project focuses on the problem of time series prediction.
- Given a data set of n observations $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$, where each input $x_i \in \mathbb{R}_{>0}$ is a time value and $y_i \in \mathbb{R}$ is a output or experimental observation that acts a function of time, the goal of time series prediction is to try and best predict a value $y_\star$ at time $x_\star$.
- The idea of studying time series prediction came from a research group from the Gatton campus, lead by Andries Potgieter, analysing crop growth from previous seasons to forecast when certain phenological stages will take place in the current harvest.



Introduction to Gaussian Processes

- Originally, Potgieter's team surveyed a number of different parameteric models to carry out forecasting. However, the parameteric models we serverely limited in their ability to inform when key phenological stages would take place. After seeing the success of applying GPs to other remote sensing tasks, the team investigated the use of GPs in their own research to find that they could produce much higher resolution predictions from which they could infer a far richer phenological timeline.
- A Gaussian Process (GP) is a collection of random variables with index set I, such that every finite subset of random variables has a joint Gaussian distribution.
- A GP is completely characterized by a mean function $m : X \to \mathbb{R}$ and a kernel $k : X \times X \to \mathbb{R}$.

$$\begin{aligned} m(\boldsymbol{x}) &= \mathbb{E}\left[f(\boldsymbol{x})\right] \\ k(\boldsymbol{x}, \boldsymbol{x}') &= \mathbb{E}\left[(f(\boldsymbol{x}) - m(\boldsymbol{x}))(f(\boldsymbol{x}') - m(\boldsymbol{x}'))\right]. \end{aligned}$$

- In this context, think of the kernel as a function that provides some notion of similarity between points.

Predictions

- The aim of GPs is to find a suitable mean function, m, for which we can then predict inputs from outside the observed values, $\boldsymbol{X}_\star$. This requires an understanding of the function f.
- Adopting the notation $(\boldsymbol{K}_{\boldsymbol{WW}'})_{i,j} \triangleq k\left(\boldsymbol{w}_i, \boldsymbol{w}_j'\right)$, when attempting to model our value function we usually do not have access to the value function itself but a noisy version thereof, $y = f(\boldsymbol{x}) + \varepsilon$ where $\varepsilon \sim \mathbb{N}(0, \sigma_n^2)$ meaning the prior on the noisy values becomes $\text{cov}(\boldsymbol{y}) = \boldsymbol{K}_{\boldsymbol{XX}} + \sigma_n^2 \boldsymbol{I}$.
- Using the assumption that our data can be modelled as a Gaussian process, we can write out the new distribution of the observed noisy values along the points at which we wish to test the underlying function as

$$\begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{f}_\star \end{bmatrix} \sim \mathbb{N}\left(\boldsymbol{0}, \quad \begin{bmatrix} \boldsymbol{K}_{\boldsymbol{XX}} + \sigma_n^2 \mathbb{I}_{n\times n} & \boldsymbol{K}_{\boldsymbol{X}_\star \boldsymbol{X}}^\intercal \\ \boldsymbol{K}_{\boldsymbol{X}_\star \boldsymbol{X}} & \boldsymbol{K}_{\boldsymbol{X}_\star \boldsymbol{X}_\star} \end{bmatrix}\right).$$

- The mean and covariance can then be computed as

$$\overline{\boldsymbol{f}_\star} \triangleq \boldsymbol{K}_{\boldsymbol{X}_\star \boldsymbol{X}} \left[\boldsymbol{K}_{\boldsymbol{XX}} + \sigma_n^2 \mathbb{I}_{n\times n}\right]^{-1} \boldsymbol{y}$$
$$\text{cov}(\boldsymbol{f}_\star) = \boldsymbol{K}_{\boldsymbol{X}_\star \boldsymbol{X}_\star} - \boldsymbol{K}_{\boldsymbol{X}_\star \boldsymbol{X}} \left[\boldsymbol{K}_{\boldsymbol{XX}} + \sigma_n^2 \mathbb{I}_{n\times n}\right]^{-1} \boldsymbol{K}_{\boldsymbol{X}_\star \boldsymbol{X}}^\intercal.$$

Unoptimized GPR

Algorithm 1: Unoptimized GPR

```
input  : Observations $\boldsymbol{X}, \boldsymbol{y}$ and a test input $\boldsymbol{x}_\star$.
output: A prediction $\overline{f_\star}$ with its corresponding variance $\mathbb{V}[f_\star]$.
1 $\boldsymbol{L} = \text{cholesky}\left(\boldsymbol{K}_{\boldsymbol{XX}} + \sigma_n^2 \mathbb{I}_{n \times n}\right)$
2 $\alpha = \text{lin-solve}\left(\boldsymbol{L}^\intercal, \text{lin-solve}\left(\boldsymbol{L}, \boldsymbol{y}\right)\right)$
3 $\overline{f_\star} = \boldsymbol{K}_{\boldsymbol{x}_\star \boldsymbol{X}} \alpha$
4 $\boldsymbol{v} = \text{lin-solve}\left(\boldsymbol{L}, \boldsymbol{K}_{\boldsymbol{x}_\star \boldsymbol{X}}\right)$
5 $\mathbb{V}[f_\star] = \boldsymbol{K}_{\boldsymbol{x}_\star \boldsymbol{x}_\star} - \boldsymbol{v}^\intercal \boldsymbol{v}$
6 return $\overline{f_\star}, \mathbb{V}[f_\star]$
```

Problems with Unoptimized GPR

Algorithm 2: Unoptimized GPR

```
input  : Observations $\boldsymbol{X}, \boldsymbol{y}$ and a test input $\boldsymbol{x}_\star$.
output: A prediction $\overline{f_\star}$ with its corresponding variance $\mathbb{V}[f_\star]$.
1 $\boldsymbol{L} = \text{cholesky}\left(\boldsymbol{K}_{\boldsymbol{XX}} + \sigma_n^2 \mathbb{I}_{n\times n}\right)$
2 $\alpha = \text{lin-solve}\left(\boldsymbol{L}^\intercal, \text{lin-solve}\left(\boldsymbol{L}, \boldsymbol{y}\right)\right)$
3 $\overline{f_\star} = \boldsymbol{K}_{\boldsymbol{x}_\star \boldsymbol{X}} \alpha$
4 $\boldsymbol{v} = \text{lin-solve}\left(\boldsymbol{L}, \boldsymbol{K}_{\boldsymbol{x}_\star \boldsymbol{X}}\right)$
5 $\mathbb{V}[f_\star] = \boldsymbol{K}_{\boldsymbol{x}_\star \boldsymbol{x}_\star} - \boldsymbol{v}^\intercal \boldsymbol{v}$
6 return $\overline{f_\star}, \mathbb{V}[f_\star]$
```

- Line 1 can be incredibly slow as computing $\boldsymbol{K}_{\boldsymbol{XX}}$ and performing a Cholesky decomposition scale poorly as the number of inputs, n, grows.

Nystrom Approximation

- One technique to speed up the computation of $\boldsymbol{K_{XX}}$ is to use a Nystrom approximation.
- The Nystrom method we seek a matrix $\boldsymbol{Q} \in \mathbb{R}^{n \times k}$ that satisfies $\|\boldsymbol{A} - \boldsymbol{QQ}^*\boldsymbol{A}\|_F \leq \varepsilon$, where $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is positive semi definite matrix, to form the rank$-k$ approximation

$$\begin{aligned}\boldsymbol{A} &\simeq \boldsymbol{QQ}^*\boldsymbol{A} \\ &\simeq \boldsymbol{Q}\left(\boldsymbol{Q}^*\boldsymbol{AQ}\right)\boldsymbol{Q}^* \\ &= \boldsymbol{Q}\left(\boldsymbol{Q}^*\boldsymbol{AQ}\right)\left(\boldsymbol{Q}^*\boldsymbol{AQ}\right)^\dagger\left(\boldsymbol{Q}^*\boldsymbol{AQ}\right)\boldsymbol{Q}^* \\ &\simeq \left(\boldsymbol{AQ}\right)\left(\boldsymbol{Q}^*\boldsymbol{AQ}\right)^\dagger\left(\boldsymbol{Q}^*\boldsymbol{A}\right).\end{aligned}$$

- A matrix $\boldsymbol{Q}$ that satisfies the above conditions can be built using through a very popular column sampling technique.
- As the name suggests, the matrix $\boldsymbol{Q}$ essentially samples and rescales columns from $\boldsymbol{A}$ using a probability distribution $\{p_i\}_{i=1}^n$.
- When $\boldsymbol{Q}$ is constructed in this manner, it is usually referred to as a sketching matrix and denoted $\boldsymbol{S}$.

Random Fourier Feature Approximation

- The other technique investigated to speed up the computation of $\boldsymbol{K}_{\boldsymbol{XX}}$ is the Random Fourier Feature (RFF) approximation.
- The main idea is instead of using a kernel function to implicitly lift data into a higher dimensional feature space, an explicit feature map $\varphi : \mathbb{R}^d \to \mathbb{R}^D$ could be used to approximate k as $k(\boldsymbol{x}, \boldsymbol{y}) = \langle \Phi(\boldsymbol{x}), \Phi(\boldsymbol{y}) \rangle_{\mathbb{R}^N} \simeq \langle \varphi(\boldsymbol{x}), \varphi(\boldsymbol{y}) \rangle_{\mathbb{R}^D}$ where D is chosen so that $n \gg D$. Once $\varphi(\boldsymbol{x}_i)$ has been computed for each $\boldsymbol{x}_i$, every entry of the Gram matrix can be swiftly approximated as $\boldsymbol{K}_{ij} = \boldsymbol{K}_{ji} \simeq \langle \varphi(\boldsymbol{x}_i), \varphi(\boldsymbol{y}_j) \rangle_{\mathbb{R}^D}$.
- The RFF technique hinges on Bochners theorem which characterises positive definite functions (namely kernels) and states that any positive definite functions can be represented as

$$k(\boldsymbol{x} - \boldsymbol{y}) = k(\boldsymbol{x} - \boldsymbol{y}) = \int_{\mathbb{C}^d} \exp\left(i \langle \boldsymbol{\omega}, \boldsymbol{x} - \boldsymbol{y} \rangle\right) \mu_k\left(d\boldsymbol{\omega}\right)$$

  where μ_k is a positive finite measure on the frequencies of $\boldsymbol{\omega}$.

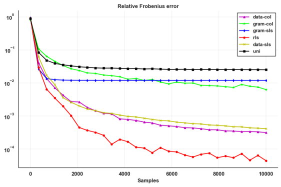
- The integral $k\left(\boldsymbol{x}-\boldsymbol{y}\right)=\int_{\mathbb{C}^d}\exp\left(i\langle\boldsymbol{\omega},\boldsymbol{x}-\boldsymbol{y}\rangle\right)\mu_k\left(d\boldsymbol{\omega}\right)$ can then be approximated via the following Monte Carlo estimate

$$\begin{aligned}
k\left(\boldsymbol{x}-\boldsymbol{y}\right) &= \int_{\mathbb{C}^d}\exp\left(i\langle\boldsymbol{\omega},\boldsymbol{x}-\boldsymbol{y}\rangle\right)p(\boldsymbol{\omega})\,d\boldsymbol{\omega}\\
&= \mathbb{E}_{\boldsymbol{\omega}\sim p(\cdot)}\left(\exp\left(i\langle\boldsymbol{\omega},\boldsymbol{x}-\boldsymbol{y}\rangle\right)\right)\\
&\simeq \frac{1}{D}\sum_{j=1}^{D}\exp\left(i\langle\boldsymbol{\omega}_j,\boldsymbol{x}-\boldsymbol{y}\rangle\right)\\
&= \sum_{j=1}^{D}\left(\frac{1}{\sqrt{D}}\exp\left(i\langle\boldsymbol{\omega}_j,\boldsymbol{x}\rangle\right)\right)\overline{\left(\frac{1}{\sqrt{D}}\exp\left(i\langle\boldsymbol{\omega}_j,\boldsymbol{y}\rangle\right)\right)}\\
&= \langle\varphi(\boldsymbol{x}),\varphi(\boldsymbol{y})\rangle_{\mathbb{C}^D}
\end{aligned}$$

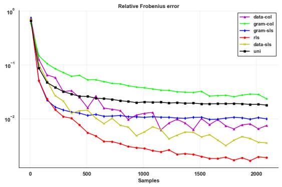
where $\boldsymbol{\omega}_i\overset{\text{iid}}{\sim}p(\cdot)$ using the feature map

$$\varphi(\boldsymbol{x})=\frac{1}{\sqrt{D}}\left[z\left(\boldsymbol{\omega}_1,\boldsymbol{x}\right),z\left(\boldsymbol{\omega}_2,\boldsymbol{x}\right),\ldots,z\left(\boldsymbol{\omega}_D,\boldsymbol{x}\right)\right]^{\mathsf{T}}$$

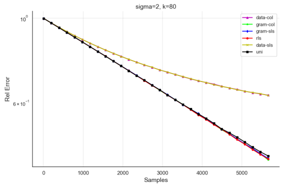
where for convenience of notation $z\left(\boldsymbol{\omega},\boldsymbol{x}\right)=\exp\left(i\langle\boldsymbol{\omega},\boldsymbol{x}\rangle\right)$.



(a) 3D-Spatial network

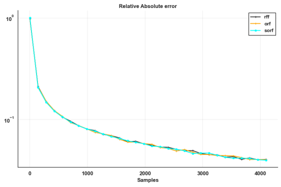


(b) Abalone

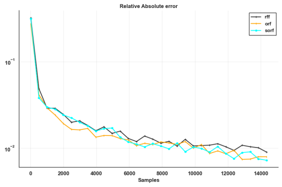


(c) Temperature

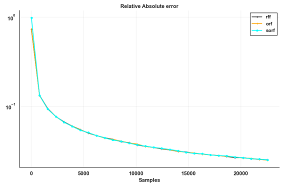
Figure: Comparison of Nystrom methods for various datasets.



(a) 3D-Spatial network

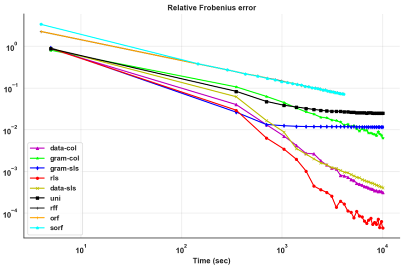


(b) Abalone

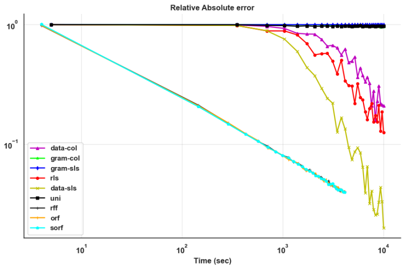


(c) Wine

Figure: Comparison of RFF methods for various datasets.



(a) Relative error



(b) Absolute error

Figure: Comparison between Nystrom and RFF approximations for the 3D-Spatial network data.

Moving Forward

- We saw that the Nystrom technique is better at producing approximations of the Gram matrix, $\boldsymbol{K_{XX}}$, with smaller *relative errors* while the RFF technique is better at producing approximations with smaller *absolute errors*.
- The question reamins which is better for GP prediction, lower relative error or absolute error (or perhaps some combination of the two)? This will ultimately determined which of the two techniques are more useful in Machine Learning applications.
- Recall, the other bottle neck in the GPR algorithm was the Cholesky decomposition.
- This can be avoided by employing faster linear system solvers. The two solvers we are interested in are the *Conjugate Gradient* (CG) and *Minimum Residual* (MINRES) methods used for solving positive semi-definite and symmeteric indefinite systems respectively.
- While MINRES can be applied to a wider class of linear systems, in certain scenarios it may perform better than the more widely used CG method. It will be interesting to see which of the two produced better predictions in the context of GPs.