STAT 37710: Homework 3

Caleb Derrickson

April 25, 2024

Collaborators: The TA's of the class, as well as Kevin Hefner, Alexander Cram, and Alice Yang.

Contents

Problem 1

An online algorithm, like the perceptron, is said to be conservative if it changes its hypothesis only when in makes a mistake. Let $\mathcal{C}$ be a concept class and A be a (not necessarily conservative) online algorithm which has a finite mistake bound M on $\mathcal{C}$. Prove that there is a conservative algorithm A' for $\mathcal{C}$ which also has mistake bound M.

Solution:

With our given A, we understand that it updates its hypothesis for any input in the concept class. Define the algorithm A' to be equivalent to A only when we encounter a mistake in prediction, else A' does nothing for correct predictions. Suppose we have a concept class of the form $\mathcal{C} = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$, where the y_i are the correct classification of the respective example x_i. We know that our original algorithm A, when every example in our concept class has been applied, will only make at most M mistakes (suppose without loss of generality $M \leq n$). Denote the subset S of $\mathcal{C}$ to be the examples which A' makes a mistake in classification. Note that, when we rerun our training, both algorithms A and A' will preform identically when run on S. Therefore, on this set, A' will make the same number of mistakes as A, the latter of which is bounded above by M. This implies that A' will itself make at most M mistakes on this set. This gives us that A' will make at most M mistakes. Therefore, the claim has been proven.

Problem 2

Give an example of a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ that is symmetric $(k(x, x') = k(x', x))$ and positive in the sense that $k(x, x') \geq 0$ for all $x, x' \in \mathcal{X}$, but is not positive semidefinite. Conversely, give an example of a kernel that is positive semidefinite, but does not satisfy $k(x, x') \geq 0$ for all $x, x' \in \mathcal{X}$.

Solution:

Let us first define the dataset $\mathcal{X}$ we are working with. For the first example, let $\mathcal{X} = \{0, 1\}$, which is just the set containing just zero and one. For the first example, let

$$k(x, x') = (x - x')^2 + (xx')^2.$$

This function is clearly symmetric and positive on our dataset. Define A to then be the Gram matrix of this function, that is

$$A = \begin{bmatrix} k(0,0) & k(0,1) \\ k(1,0) & k(1,1) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}.$$

In order for our function k to be psd, it is sufficient to show that this matrix is psd. Note that this is not the case, however, as one of its eigenvalues is negative. In fact, its eigenvalues are

$$\lambda_{1,2} = \frac{1}{2}(1 \pm \sqrt{5}).$$

Clearly showing that this matrix is not psd, we can explicitly write show an example of a vector $\mathbf{c}$ for which $\mathbf{c}^\intercal A\mathbf{c} < 0$. By explicitly writing this out, we see

$$\mathbf{c}^\intercal A\mathbf{c} = \begin{bmatrix} c_1 & c_2 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = c_2(c_2 + 2c_1).$$

Taking $c_2 = 1, c_1 = -1$, we get the above to equal -1. Therefore, A is doubly not psd.

For the second example, let our data set be $\mathcal{X} = \{-1, 1\}$ and our function to be simply $k(x, x') = xx'$. Forming the matrix A as in the above example gives us

$$A = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix},$$

which has eigenvalues $0, 2$. This makes A psd, which in turn makes the function k psd. However, k is clearly not positive for all $x, x' \in \mathcal{X}$, since $k(-1, 1) = -1 < 0$. Therefore, both examples have been given.

Problem 3

Given any function $\psi : \mathcal{X} \to \mathcal{X}'$, prove that if k' is a psd kernel on $\mathcal{X}'$, then $k(x, x') = k'(\psi(x), \psi(x'))$ is a psd kernel on $\mathcal{X}$.

Solution:

We will first show symmetry. Let $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$, then

$$k(\mathbf{x}, \mathbf{x}') = k'(\psi(\mathbf{x}), \psi(\mathbf{x}')) = k'(\psi(\mathbf{x}'), \psi(\mathbf{x})) = k(\mathbf{x}', \mathbf{x}).$$

The above holds since k' is given to be psd, in particular symmetric. Next, let the vector ξ be given as the following: suppose $c_1, ..., c_n \in \mathbb{R}$, then, we have

$$\xi = \sum_{i=1}^{n} c_i \psi(\mathbf{x}_i) \in \mathcal{X}'.$$

Since $\xi \in \mathcal{X}'$, we then have that

$$\langle \xi, \xi \rangle \geq 0 \implies \sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j k'(\psi(\mathbf{x}_i), \psi(\mathbf{x}_j)) \geq 0.$$

Since we have the given relation between the two kernels, we have that

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j k(\mathbf{x}_i, \mathbf{x}_j) \geq 0.$$

Therefore, we have that k is a psd kernel.

Problem 4

Prove that if k_1 and k_2 are two positive semi-definite (psd) kernels on a space $\mathcal{X}$, then

Problem 4, part a

The function, $k(x, x') := k_1(x, x') + k_2(x, x')$ is a psd kernel on $\mathcal{X}$;

Solution:

In order for the given function to be a psd kernel, we require k to be both symmetric and have the property that

$$\sum_{i,j} c_i c_j k(x_i, x_j) \geq 0.$$

Symmetry is simple, since

$$k(x, x') = k_1(x, x') + k_2(x, x') = k_1(x', x) + k_2(x', x) = k(x', x).$$

The above actions are justified since k_1, k_2 are psd, hence symmetric. Next, we write down the given sum in terms of k_1 and k_2.

$$\sum_{i,j} c_i c_j k(x_i, x_j) = \sum_{i,j} c_i c_j k_1(x_i, x_j) + \sum_{i,j} c_i c_j k_2(x_i, x_j).$$

Since both k_1 and k_2 are psd, the two sums on the left are positive for any choice $c_1, ..., c_n \in \mathbb{R}$. Hence, the right hand side is positive, so k is shown to be psd.

Problem 4, part b

The function $k_\oplus((x_1, x_2), (x_1', x_2')) = k_1(x_1, x_1') + k_2(x_2, x_2')$ is a psd kernel on $\mathcal{X} \times \mathcal{X}'$.

Solution:

We will repeat the same process as in the previous example. $k_\oplus$ is symmetric, since

$$k_\oplus((x_1, x_2), (x_1', x_2')) = k_1(x_1, x_1') + k_2(x_2, x_2') = k_1(x_1', x_1) + k_2(x_2', x_2) = k_\oplus((x_1', x_2'), (x_1, x_2)).$$

Again, this works since k_1, k_2 are symmetric. Then, showing its sum is positive, we have

$$\sum_{i,j} c_i c_j k_\oplus((x_i^1, x_i^2), (x_j^2, x_j^2)) = \sum_{i,j} c_i c_j (k_1(x_i^1, x_j^1) + k_2(x_i^2, x_j^2)) = \sum_{i,j} c_i c_j k_1(x_i^1, x_j^1) + \sum_{i,j} c_i c_j k_2(x_i^2, x_j^2).$$

Since k_1, k_2 are psd, then the two sums on the right are positive. This implies that the sum on the left are positive. Therefore, the function $k_\oplus$ is a psd kernel.

Problem 4, part c

Given any function $\psi : \mathcal{X} \to \mathcal{X}'$, prove that if k' is a psd kernel on $\mathcal{X}'$, then $k(x, x') = k'(\psi(x), \psi(x'))$ is a psd kernel on $\mathcal{X}$.

Solution:

This is problem 3, just repeated. Please see my answer to problem 3 for my answer.

Problem 4, part d

Let $\alpha(\mathbf{x}, \mathbf{x}')$ be the angle between $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^n$. Prove that the cosine kernel $k_\angle(\mathbf{x}, \mathbf{x}') = \cos(\alpha(\mathbf{x}, \mathbf{x}'))$ is a psd kernel on $\mathcal{X} = \mathbb{R}^n$.

Solution:

First, we should define the angle between two vectors in $\mathbb{R}^n$ space. Note the inner product between $\mathbf{x}$ and $\mathbf{x}'$ can be written as

$$\langle \mathbf{x}, \mathbf{x}' \rangle = \|\mathbf{x}\| \|\mathbf{x}'\| \cos(\alpha),$$

where α denotes the angle between the two. Rewriting for θ, we have

$$\alpha(\mathbf{x}, \mathbf{x}') = \arccos\left(\frac{\langle \mathbf{x}, \mathbf{x}' \rangle}{\|\mathbf{x}\| \|\mathbf{x}'\|}\right).$$

This implies that, when plugging into $k_\angle$, we have that

$$k_\angle(\mathbf{x}, \mathbf{x}') = \frac{\langle \mathbf{x}, \mathbf{x}' \rangle}{\|\mathbf{x}\| \|\mathbf{x}'\|}.$$

This is clearly symmetric, since the inner product is symmetric. Furthermore, the cosine kernel defines the entry-wise normalized Gram matrix [1], which is indeed symmetric, hence positive semidefinite. Therefore, the claim has been shown.

[1] Instead of this entry-wise normalized Gram matrix, we can first normalize our data. Then the cosine kernel simply defines the Gram Matrix. Either way, the claim that this defines a symmetric matrix is not lost.

Problem 5

Recall that a training set $\{(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)\}$ is said to have edge γ over a set of weak classifiers H if for any distribution D over the training set, there is at least one weak learner $h \in H$ such that $\varepsilon_h = \sum_{i=1}^{m} D(i)\ell_{0/1}(h(x_i), y_i) \leq 1/2 - \gamma$.

Problem 5, part a

Using the inequality $\ell_{0/1}(z, 1) \leq e^{-z}$ prove that after t rounds of boosting the running hypothesis $\hat{h}(x) = \text{sgn}\left(\sum_{s=1}^{t} \alpha_s h_s(x)\right)$ satisfies

$$\ell_{0/1}(\hat{h}(x_i), y_i) \leq m \left(\prod_{s=1}^{t} Z_s\right) D_{t+1}(i)$$

for every example $i = 1, 2, ..., m$.

Solution:

The Distribution $D_{t+1}(i)$ is given by

$$D_{t+1}(i) = \frac{1}{m}\left(\prod_{s=1}^{t} Z_s\right)^{-1} \exp\left(-y_i \sum_{s=1}^{t} \alpha_s h_s(x_i)\right).$$

This is given in the slides. We can rearrange this to get

$$m\left(\prod_{s=1}^{t} Z_s\right) D_{t+1}(i) = \exp\left(-y_i \sum_{s=1}^{t} \alpha_s h_s(x_i)\right).$$

By the bound given, we have

$$\ell_{0/1}\left(y_i \sum_{s=1}^{t} \alpha_s h_s(x_i), 1\right) \leq m\left(\prod_{s=1}^{t} Z_s\right) D_{t+1}(i).$$

This is equal to the given inequality above, we just need to massage the penalty function. Note that the $\ell_{0/1}$ loss function is given by

$$\ell_{0/1}(x, y) = \begin{cases} 0, & x = y \\ 1, & \text{o.w.} \end{cases}.$$

For our case, we have

$$\ell_{0/1}\left(y_i \sum_{s=1}^{t} \alpha_s h_s(x_i), 1\right) = \begin{cases} 0, & 1 = y_i \sum_{s=1}^{t} \alpha_s h_s(x_i) \\ 1, & \text{o.w.} \end{cases}$$

The equality $1 = y_i \sum_{s=1}^{t} \alpha_s h_s(x_i)$ can be rephrased as $1 = y_i \hat{h} \left|\sum_{s=1}^{t} \alpha_s h_s(x_i)\right|$. This gives us the size of the magnitude of the sum, but for the purposes of this problem, we can neglect it. We are primarily concerned with the equality $1 = y_i\hat{h}$, which when plugging into our loss function, we have

$$\ell_{0/1}(y_i, \hat{h}) = \begin{cases} 0, & y_i = \hat{h} \\ 1, & \text{o.w.} \end{cases}.$$

We can then write

$$\ell_{0/1}(y_i, \hat{h}) \leq m \left(\prod_{s=1}^{t} Z_s \right) D_{t+1}(i),$$

which is what we wanted to show.

Problem 5, part b

Use this to show that

$$\mathcal{E}_{\text{train}}(\hat{h}) \leq \prod_{s=1}^{t} 2\sqrt{\varepsilon_s(1-\varepsilon_s)}.$$

Solution:

By the definition of the training error,

$$\mathcal{E}_{\text{train}}(\hat{h}) = \frac{1}{m}\sum_{i=1}^{m} \ell_{0/1}(\hat{h}(x_i), y_i),$$

we can plug in our bound in part a to get

$$\mathcal{E}_{\text{train}}(\hat{h}) \leq \sum_{i=1}^{m} D_{t+1}(i) \prod_{s=1}^{t} Z_s = \left(\sum_{i=1}^{m} D_{t+1}(i)\right)\left(\prod_{s=1}^{t} Z_s\right).$$

Since the given distribution is discrete, summing over all its entries is equal to one. Hence

$$\mathcal{E}_{\text{train}}(\hat{h}) \leq \prod_{s=1}^{t} Z_s.$$

Note that for any s, $Z_s = 2\sqrt{\varepsilon_s(1-\varepsilon_s)}$. This can be substituted in to get

$$\mathcal{E}_{\text{train}}(\hat{h}) \leq \prod_{s=1}^{t} 2\sqrt{\varepsilon_s(1-\varepsilon_s)},$$

which is what we wanted to show.

Problem 5, part c

By plugging into the definition of the edge at round s, which is $\gamma_s = 1/2 - \varepsilon_s$ and using the inequality $1 - z \leq e^{-z}$ prove that the training error decreases exponentially,

$$\mathcal{E}_{\text{train}}(\hat{h}) \leq \exp\Bigl(-2\gamma^2 t\Bigr),$$

as stated in a Theorem in class.

Solution: Substituting in the given form for γ_s from the result of part b, we have

$$\mathcal{E}_{\text{train}}(\hat{h}) \leq \prod_{s=1}^{t} 2\sqrt{(1/2 - \gamma_s)(1/2 + \gamma_s)} = \prod_{s=1}^{t} \sqrt{1 - 4\gamma_s^s}.$$

By the given bound, we can say that

$$\mathcal{E}_{\text{train}}(\hat{h}) \leq \prod_{s=1}^{t} \sqrt{\exp\left(-4\gamma_s^2\right)} = \prod_{s=1}^{t} \exp\left(-2\gamma_s^2\right).$$

Since the negative exponential is a monotonically decreasing function, another upper bound can be made by instead substituting the minimum of the γ_s's in the exponential. In particular, we have

$$\mathcal{E}_{\text{train}}(\hat{h}) \leq \prod_{s=1}^{t} \exp\left(-2\gamma^2\right),$$

which after exponent properties, we get

$$\mathcal{E}_{\text{train}}(\hat{h}) \leq \exp\left(-2\gamma^2 \sum_{s=1}^{t}\right) = \exp\left(-2\gamma^2 t\right).$$

This then proves the theorem stated in class.

Problem 6

Recall that a Gaussian Process is a distribution over functions, not just over some finite collection of variables. Specifically, a GP $\mathcal{G}(\mu, k)$ on the real line is a dirstibution for which we fix n points $x_1, x_2, ..., x_n \in \mathbb{R}$ and draw a function f from $\mathcal{G}(\mu, k)$, the function values $f(x_1), f(x_2), ..., f(x_n)$ are jointly normally distributed with

$$\mathbb{E}(f(x_i)) = \mu(x_i),$$

$$\text{Cov}(f(x_i, f(x_j)) = k(x_i, x_j).$$

Here, μ and k are considered parameters of the GP, just like the vector mean vector $\boldsymbol{\mu}$ and the covariance matrix $\boldsymbol{\Sigma}$ are parameters of the normal distribution: μ can be any function $\mu : \mathbb{R} \to \mathbb{R}$, and $k\mathbb{R} \times \mathbb{R} \to \mathbb{R}$ can be any positive semi-definite kernel on $\mathbb{R}$. (It takes a little bit of theoretical work to show that for any $\mu, k, \mathcal{G}(\mu, k)$ really is a valid distribution over functions, and this is essentially unique.) For simplicity, in the following we will take $\mu(x) = 0$, and set k to be our favorite kernel, the Gaussian RBF kernel $k(x, x') = \exp\left(-\frac{1}{2\tau^2}(x - x')^2\right)$.

In the Bayesian framework, GPs are used as a prior for the function $\hat{f}$ that we are trying to estimate. The beauty of GPs is that there are several complicated looking things that one can do with them in a very simple way:

1. We can draw function from the prior $\mathcal{G}(\mu, k)$.
2. In a regression setting, if we assume that the observed data $S = \{(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)\}$ are distributed around f according to a second univariate Gaussian:

$$y = f(x) + \eta \quad \eta \sim \mathcal{N}(0, \sigma^2), \tag{1}$$

   we can invoke Bayes' rule

$$p(f|S) = \frac{p(S|f)p(f)}{p(S)}$$

   to get the posterior distribution over f. Miraculously, in the case the posterior also turns out to be a Gaussian Process, $\mathcal{G}(\mu', k')$.
3. We can draw further functions from this update GP $\mathcal{G}(\mu', k')$, or just use its mean $\hat{f} = \mu'$ as our regression estimate (which will be the same as doing Ridge Regression), and k' as a measure of uncertainty about $\hat{f}$.

In this problem, you are asked to do the following: Assume that for $x_1, x_2, ..., x_n \in \mathbb{R}$, the function values $f(x_1), ..., f(x_n)$ are known, and we want to estimate the value of f at some point x. Define the Gram matrix $\mathbf{K} \in \mathbb{R}^{n \times n}$, the vector $\mathbf{k}_x \in \mathbb{R}^n$, and the scalar κ_x as

$$\mathbf{K}_{i,j} = k(x_i, x_j)$$

$$[\mathbf{k}_x]_i = k(x, x_i)$$

$$\kappa_x = k(x, x),$$

and let $\mathbf{f} = (f(x_1), ..., f(x_n))^\intercal$. Assume for simplicity that $\mu(x) = 0$. The key to using GP's is that since $(f(x_1), ..., f(x_n), f(x))$ are jointly normal with covariance matrix

$$\mathbf{K}^{(n+1)} = \left[\begin{array}{c|c} \mathbf{K} & \mathbf{k}_x \\ \hline \mathbf{k}_x^\intercal & \kappa_x \end{array}\right],$$

the distribution of $f(x)$ given $(f(x_1), ..., f(x_n))$ is also normal.

Problem 6, part a

[2] Show that the mean and variance of $f(x)$ given $(f(x_1), ..., f(x_n))$ is

$$\mathbb{E}(f(x)) = \mathbf{k}_x^\intercal \mathbf{K}^{-1}\mathbf{f}$$

$$\text{Var}(f(x)) = \kappa_x - \mathbf{k}_x^\intercal \mathbf{K}^{-1}\mathbf{k}_x.$$

Solution:

We will investigate the probability of the $f(x)$ given $\mathbf{f} \equiv (f(x_1), ..., f(x_n))$. Note that Bayes rule tells us

$$p(f(x)|\mathbf{f}) = \frac{p(\mathbf{f}|f(x))p(f(x))}{p(\mathbf{f})} = \frac{p(\mathbf{f}, f(x))}{p(\mathbf{f})}$$

Define $\mathbf{f}' = (\mathbf{f}^\intercal, f(x))^\intercal$. By the properties of GPs, (this is given as i), we can draw functions from the prior. Hence,

$$p(f(x)|\mathbf{f}) = \frac{(2\pi)^{-(n+1)/2}|\mathbf{K}_{n+1}|^{-1}\exp\left(-\frac{1}{2}\mathbf{f}'^\intercal\mathbf{K}^{-1}\mathbf{f}'\right)}{(2\pi)^{-n/2}|\mathbf{K}_n|^{-1}\exp\left(-\frac{1}{2}\mathbf{f}^\intercal\mathbf{K}^{-1}\mathbf{f}'\right)} = \frac{1}{\sqrt{2\pi}}\frac{|\mathbf{K}|}{|\mathbf{K}_{n+1}|}\exp\left(-\frac{1}{2}\left[\mathbf{f}'^\intercal\mathbf{K}_{n+1}^{-1}\mathbf{f}' - \mathbf{f}^\intercal\mathbf{K}^{-1}\mathbf{f}\right]\right)$$

We will first investigate and simplify the exponential term. Taking the first term, we will expand it to get

$$\begin{bmatrix}\mathbf{f}^\intercal & f(x)\end{bmatrix}\mathbf{K}_{n+1}^{-1}\begin{bmatrix}\mathbf{f} \\ f(x)\end{bmatrix} = \begin{bmatrix}\mathbf{f}^\intercal & f(x)\end{bmatrix}\begin{bmatrix}(\mathbf{K} - \frac{1}{\kappa_x}\mathbf{k}_x\mathbf{k}_x^\intercal)^{-1} & -(\mathbf{K} - \frac{1}{\kappa_x}\mathbf{k}_x\mathbf{k}_x^\intercal)^{-1}\frac{1}{\kappa_x}\mathbf{k}_x \\ \frac{\mathbf{k}_x^\intercal}{\kappa_x}(\mathbf{K} - \frac{1}{\kappa_x}\mathbf{k}_x\mathbf{k}_x^\intercal)^{-1} & \frac{1}{\kappa_x} + \frac{\mathbf{k}_x^\intercal}{\kappa_x}\left(\mathbf{K} - \frac{1}{\kappa_x}\mathbf{k}_x\mathbf{k}_x^\intercal\right)^{-1}\frac{\mathbf{k}_x}{\kappa_x}\end{bmatrix}\begin{bmatrix}\mathbf{f} \\ f(x)\end{bmatrix} \quad (2)$$

The Expansion seen above was performed using the Sherman Morrison Woodberry formula. You will notice that there is a repeating term which shows up in each entry of the expansion, for the sake of legibility I will

[2]This is gonna get really ugly, really quickly.

rewite this. Implementing the Sherman-Morrison formula to this term, we get

$$\left(\mathbf{K} - \frac{1}{\kappa_x}\mathbf{k}_x\mathbf{k}_x^\intercal\right)^{-1} = \mathbf{K}^{-1} + \frac{1}{\kappa_x}\left(\frac{\mathbf{K}^{-1}\mathbf{k}_x\mathbf{k}_x^\intercal\mathbf{K}^{-1}}{1-\mathbf{k}_x^\intercal\mathbf{K}^{-1}\mathbf{k}_x}\right) = \mathbf{K}^{-1} + \frac{\mathbf{K}^{-1}\mathbf{k}_x\mathbf{k}_x^\intercal\mathbf{K}^{-1}}{\kappa_x - \mathbf{k}_x^\intercal\mathbf{K}^{-1}\mathbf{k}_x} \tag{3}$$

I will define the denominator in the above to be equal to λ. For the simplification of the expansion of $\mathbf{K}_{n+1}^{-1}$, I will denote its block entries as A, B, C, D, where each entry is labelled in the standard sense. Note that (3) is equal to A. I will simplify each term separately below:

$B:$

$$= -\left(\mathbf{K}^{-1} - \frac{\mathbf{K}^{-1}\mathbf{k}_x\mathbf{k}_x^\intercal\mathbf{K}^{-1}}{\kappa_x - \mathbf{k}_x^\intercal\mathbf{K}^{-1}\mathbf{k}_x}\right)\frac{\mathbf{k}_x}{\kappa_x} \qquad \text{(Applying (3.))}$$

$$= -\frac{\mathbf{K}^{-1}\mathbf{k}_x}{\kappa_x} + \frac{\mathbf{K}^{-1}\mathbf{k}_x\mathbf{k}_x^\intercal\mathbf{K}^{-1}\mathbf{k}_x}{\kappa_x\lambda} \qquad \text{(Rearranging.)}$$

$$= \frac{-\mathbf{K}^{-1}\mathbf{k}_x(\kappa_x - \mathbf{k}_x^\intercal\mathbf{K}^{-1}\mathbf{k}_x) + \mathbf{K}^{-1}\mathbf{k}_x\mathbf{k}_x^\intercal\mathbf{K}^{-1}\mathbf{k}_x}{\kappa_x\lambda} \qquad \text{(Combining fractions.)}$$

$$= -\frac{\mathbf{K}^{-1}\mathbf{k}_x}{\lambda} \qquad \text{(Simplifying.)}$$

$C:$

$$\frac{\mathbf{k}_x^\intercal}{\kappa_x}\left(\mathbf{K}^{-1} + \frac{\mathbf{K}^{-1}\mathbf{k}_x\mathbf{k}_x^\intercal\mathbf{K}^{-1}}{\kappa_x - \mathbf{k}_x^\intercal\mathbf{K}^{-1}\mathbf{k}_x}\right) \qquad \text{(Applying (3.))}$$

$$= \frac{\mathbf{k}_x^\intercal\mathbf{K}^{-1}}{\kappa_x} + \frac{\mathbf{k}_x^\intercal\mathbf{K}^{-1}\mathbf{k}_x\mathbf{k}_x^\intercal\mathbf{K}^{-1}}{\kappa_x\lambda} \qquad \text{(Rearranging.)}$$

$$= \frac{-\mathbf{k}_x^\intercal\mathbf{K}^{-1}(\kappa_x - \mathbf{k}_x^\intercal\mathbf{K}^{-1}\mathbf{k}_x) + \mathbf{k}_x^\intercal\mathbf{K}^{-1}\mathbf{k}_x\mathbf{k}_x^\intercal\mathbf{K}^{-1}}{\kappa_x\lambda} \qquad \text{(Combining fractions.)}$$

$$= -\frac{\mathbf{k}_x^\intercal\mathbf{K}^{-1}}{\lambda} \qquad \text{(Simplifying.)}$$

$D:$

$$\frac{1}{\kappa_x} + \frac{\mathbf{k}_x^\intercal}{\kappa_x}\left(\mathbf{K}^{-1} + \frac{\mathbf{K}^{-1}\mathbf{k}_x\mathbf{k}_x^\intercal\mathbf{K}^{-1}}{\kappa_x - \mathbf{k}_x^\intercal\mathbf{K}^{-1}\mathbf{k}_x}\right)\frac{\mathbf{k}_x}{\kappa_x} \qquad \text{(Applying (3).)}$$

$$= \frac{1}{\kappa_x} - \frac{\mathbf{k}_x^\intercal\mathbf{K}^{-1}\mathbf{k}_x}{\kappa_x\lambda} \qquad \text{(Using } B\text{.)}$$

$$= \frac{\kappa_x - \mathbf{k}_x^\intercal \mathbf{K}^{-1}\mathbf{k}_x + \mathbf{k}_x^\intercal \mathbf{K}^{-1}\mathbf{k}_x}{\kappa_x \lambda} \qquad \text{(Combining fractions.)}$$

$$= \frac{1}{\lambda} \qquad \text{(Simplifying.)}$$

According to the calculations above, we can rewrite (2) as the following:

$$\begin{bmatrix} \mathbf{f}^\intercal & f(x) \end{bmatrix} \mathbf{K}_{n+1}^{-1} \begin{bmatrix} \mathbf{f} \\ f(x) \end{bmatrix} = \begin{bmatrix} \mathbf{f}^\intercal & f(x) \end{bmatrix} \begin{bmatrix} \mathbf{K}^{-1} + \frac{\mathbf{K}^{-1}\mathbf{k}_x\mathbf{k}_x^\intercal\mathbf{K}^{-1}}{\lambda} & -\frac{\mathbf{K}^{-1}\mathbf{k}_x}{\lambda} \\ -\frac{\mathbf{k}_x^\intercal\mathbf{K}^{-1}}{\lambda} & \frac{1}{\lambda} \end{bmatrix} \begin{bmatrix} \mathbf{f} \\ f(x) \end{bmatrix}$$

After all the matrix vector multiplication, this equates to

$$\mathbf{f}^\intercal\mathbf{K}^{-1}\mathbf{f} + \frac{1}{\lambda}\mathbf{f}^\intercal\mathbf{K}^{-1}\mathbf{k}_x\mathbf{k}_x^\intercal\mathbf{K}^{-1}\mathbf{f} - \frac{1}{\lambda}\mathbf{f}\mathbf{K}^{-1}\mathbf{k}_x f(x) - \frac{1}{\lambda}\mathbf{k}_x^\intercal\mathbf{K}^{-1}\mathbf{f} + \frac{f(x)^2}{\lambda}$$

Note that in $p(f(x)|\mathbf{f})$, we are subtracting away $\mathbf{f}^\intercal\mathbf{K}^{-1}\mathbf{f}$, so the first term in the above expression is subtracted away. After grouping, we have

$$\frac{1}{\lambda}\Big(f(x) - \mathbf{f}^\intercal\mathbf{K}^{-1}\mathbf{k}_x\Big)^2 \tag{4}$$

Next, we'll simplify the normalization constant in $p(f(x)|\mathbf{f})$. This equates to finding the determinant of $\mathbf{K}_{n+1}$. In this form, the determinant of a block matrix is given as

$$\det\left(\begin{bmatrix} A & B \\ C & D \end{bmatrix}\right) = \det(A)\det\Big(D - CA^{-1}B\Big),$$

so the determinant is given as

$$\det(\mathbf{K}_{n+1}) = \det(\mathbf{K})\det\Big(\kappa_x - \mathbf{k}_x^\intercal\mathbf{K}^{-1}\mathbf{k}_x\Big) = \lambda\det(\mathbf{K})$$

Therefore, $p(f(x)|\mathbf{f})$ simplifies to

$$p(f(x)|\mathbf{f}) = \frac{1}{\sqrt{2\pi|\lambda|^2}}\exp\left(-\frac{1}{2\lambda}\Big(f(x) - \mathbf{f}^\intercal\mathbf{K}^{-1}\mathbf{k}_x\Big)^2\right)$$

Therefore, the mean of $f(x)$ is $\mathbf{f}^\intercal\mathbf{K}^{-1}\mathbf{k}_x$ and the variance is λ.

Problem 6, part b

Similarly, show that if $y_1, ..., y_n$ are distributed around $x_1, ..., x_n$ according to (1), then given $\mathbf{y} = (y_1, y_2, ..., y_n)$,

$$p(f(x)|y_1, ..., y_n) \sim \mathcal{N}\left(\mathbf{k}_x^\intercal(\mathbf{K} + \sigma^2\mathbb{I})^{-1}\mathbf{y}, \kappa_x - \mathbf{k}_x^\intercal(\mathbf{K} + \sigma^2\mathbb{I})^{-1}\mathbf{k}_x\right)$$

Solution:

According to (1), the y_i's are distributed around $f(x)$ according to a normal distribution with variance σ^2. Thus,

$$\begin{bmatrix} y_1 \\ \vdots \\ y_n \\ f(x) \end{bmatrix} = \begin{bmatrix} f(x_1) \\ \vdots \\ f(x_n) \\ f(x) \end{bmatrix} + \begin{bmatrix} \eta_1 \\ \vdots \\ \eta_n \\ 0 \end{bmatrix}$$

We can then repeat the process done in the previous part, except with using $\mathbf{y}$ over $\mathbf{f}$ to get the form above. Note that the covariance should be replaced with $\mathbf{K} + \sigma^2\mathbb{I}$, due to the distribution of the y_i's.

Problem 6, part c

Let $\mu(x) = 0$ and $k(x, x') = \exp\left(-\frac{1}{(2\tau)^2}(x - x')^2\right)$ with $\tau^2 = 0.12$. Draw 20 different samples from $\mathcal{G}(\mu, k)$ and plot them, restricted to the unit interval $[0, 1]$ on the x axis. For this, all you need to do is let $z_1, ..., z_n$ be a sufficient number of equispaced points on $[0, 1]$, and plot the line connecting $f(z_1), ..., f(z_N)$, where $f \sim \mathcal{G}(\mu, k)$.

Solution:

The results are shown in Figure 1.

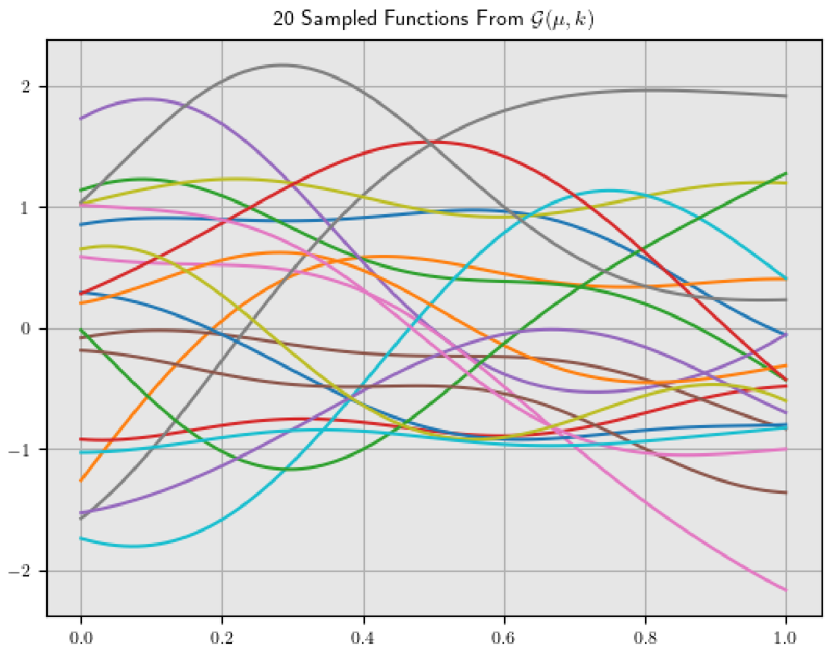


Figure 1: 20 Equispaced points drawn from given kernel.

Problem 6, part d

Now apply GP regression to the dataset gp.dat. Your prior should be $\mathcal{G}(\mu, k)$ as before. Given the data in gp.dat (the first column are the x values and the second column are the y values, plot the posterior mean

$$\mu'(x) = \mathbf{k}_x^\intercal \left(\mathbf{K} + \sigma^2 \mathbb{I}\right)^{-1} \mathbf{y}$$

and the two standard deviation bounds around it

$$s'_\pm = \mathbf{k}_x^\intercal \left(\mathbf{K} + \sigma^2 \mathbb{I}\right)^{-1} \mathbf{y} \pm 2\sqrt{\kappa_x - \mathbf{k}_x^\intercal \left(\mathbf{K} + \sigma^2 \mathbb{I}\right)^{-1} \mathbf{k}_x}.$$

Solution:

I have implemented GP Regression (this time, in Python. In retrospect, it wasn't that difficult, so I could have maybe done it in C++). My plot is shown below. Note that I was a little confused on what exactly I should set the σ value to be. It just so happens that $\sigma = 1$ looks perfectly reasonable for this data set, so I did not touch it. My plot is shown below, which gives the mean and two deviations away from the mean for the data in gp.dat.

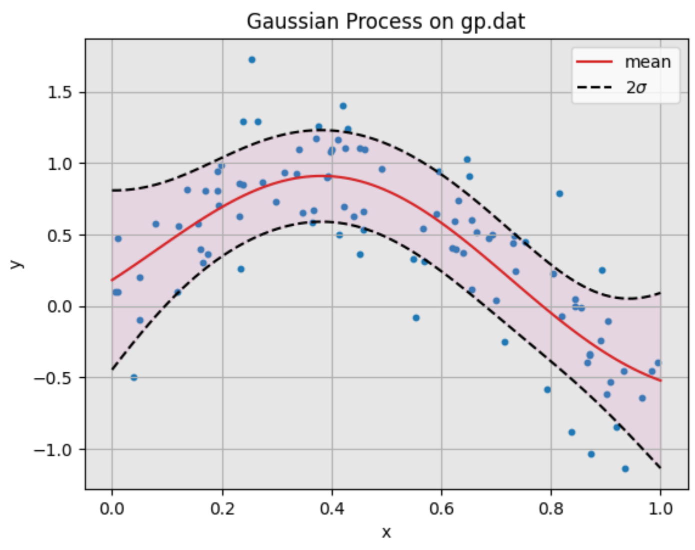


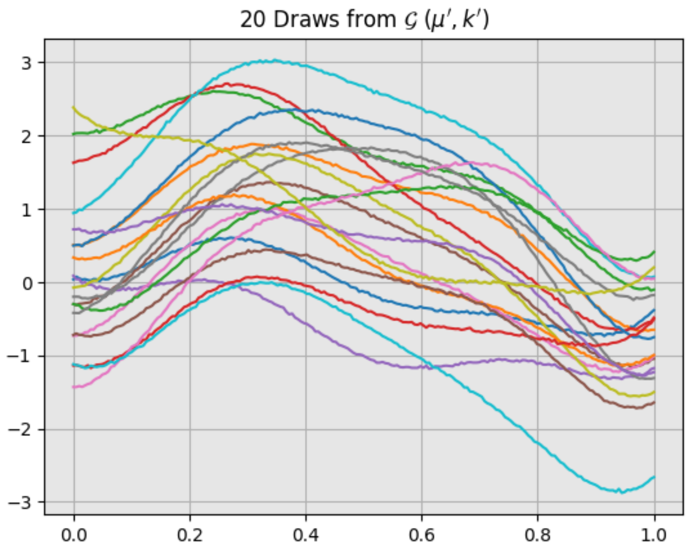
Figure 2: GP Process on gp.dat

Problem 6, part e

Plot 20 samples from the posterior GP $\mathcal{G}(\mu', k')$. Similarly to part (c).

Solution:

My results are below. The entirety of my code will follow.



In [ ]:
```python
import numpy as np
import matplotlib.pyplot as plt
import scipy
import math
```

In [ ]:
```python
def RBFkernel(x1: float, x2: float, sigma_sq: float = 1):
    """
    RBF kernel.
    """
    pow = -1/(2*sigma_sq) * (x1 - x2)**2
    return math.exp(pow)


def cov_mat(x1: np.ndarray, x2: np.ndarray, ker: callable, sigma_sq: float = 1) -> np.ndarray:
    """
    Returns the Covraiance matrix for the given kernel.
    """
    n = max(x1.shape)
    cov = np.zeros((n, n))
    for i in range(n):
        for j in range(n):
            cov[i][j] = ker(x1[i], x2[j], sigma_sq)

    return cov
```

In [ ]:
```python
plt.figure(figsize=[9, 12])
plt.rcParams.update({
    'font.size': 8,
    'text.usetex': True,
    'text.latex.preamble': r'\usepackage{amsfonts}'
})
```

In [ ]:
```python
n_points = 100
n_draws = 20
interval = (0, 1)
xs = np.linspace(interval[0], interval[1], n_points)#.reshape(-1, 1)
Si = cov_mat(xs, xs, ker = RBFkernel, sigma_sq=0.12)
mean = np.zeros(n_points)
ys = np.random.multivariate_normal(mean=mean, cov=Si, size=n_draws)#.reshape(-1, 1)

for i in range(n_draws):
    plt.plot(xs, ys[i])

plt.grid()
plt.gca().set_facecolor((0.9, 0.9, 0.9))
plt.title(r"20 Sampled Functions From $\mathcal{G}(\mu, k)$")
```

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy
import math
```

In [ ]:
```
def RBFkernel(x1: float, x2: float, sigma_sq: float = 1):
    """
    RBF kernel.
    """
    pow = -1/(2*sigma_sq) * (x1 - x2)**2
    return math.exp(pow)



def cov_mat(x1: np.ndarray, x2: np.ndarray, ker: callable, sigma_sq: float = 1) -> np.ndarray:
    """
    Returns the Covraiance matrix for the given kernel.
    """
    n = max(x1.shape)
    cov = np.zeros((n, n))
    for i in range(n):
        for j in range(n):
            cov[i][j] = ker(x1[i], x2[j], sigma_sq)

    return cov

def GPRegression(x1: np.ndarray, y1: np.ndarray, n_points: int,
                 interval : tuple, sigma_sq: float) -> tuple:
    """
    GP Regression with specified parameters. Returns mean and
    standard deviation (for each point).
    """
    n = x1.__len__()

    x2 = np.linspace(interval[0], interval[1], n_points)#.reshape(-1, 1)
    cov = cov_mat(x1, x1, RBFkernel, sigma_sq=0.12)

    mean = np.zeros(n_points)
    stdev = np.zeros(n_points)
    i = 0
    while i < n_points:

        # First generate the mean, then stdev
        # Form k_x
        k_x = np.zeros(n)
        for j in range(n):
            k_x[j] = RBFkernel(x2[i], x1[j], 0.12)
        # Set kappa_x
        kappa_x = RBFkernel(x2[i], x2[i])
        # Make K + si_sq*I
        K_mod = cov + sigma_sq*np.eye(n)
        mean_right = scipy.linalg.solve(K_mod, y1, assume_a='pos')
        mean[i] = np.dot(k_x, mean_right)

        # Now do stdev
        stdev_right = scipy.linalg.solve(K_mod, k_x, assume_a='pos')
        if kappa_x - np.dot(k_x, stdev_right) < 0:
            print("Error, standard deviation too small. Increasing...")
            i = 0
            sigma_sq = sigma_sq*2
            continue

        stdev[i] = kappa_x - np.dot(k_x, stdev_right)
        stdev[i] = np.sqrt(stdev[i])
        i+=1
    print("Ending sigma: ", sigma_sq)
    return (mean, stdev)
```

In [ ]:
```
# Fetching data
data = np.genfromtxt("gp.dat")
x1 = data[:, 0]
y1 = data[:, 1]

#Desired output
n_points = 200
x2 = np.linspace(0, 1, n_points)

interval = (0, 1)
sigma_sq = 1
# Get Mean, stdev
mean, stdev = GPRegression(x1, y1, n_points, interval, sigma_sq)
```

In [ ]:
```
plt.scatter(x1, y1, s=10)
plt.plot(x2, mean, color='tab:red', label= "mean")
plt.plot(x2, mean+2*stdev, color='k', linestyle='--', label = "2$\sigma$")
plt.plot(x2, mean-2*stdev, color='k', linestyle='--')
plt.fill_between(x2, mean+2*stdev, mean, alpha=0.15, color='tab:pink')
plt.fill_between(x2, mean-2*stdev, mean, alpha=0.15, color='tab:pink')
plt.gca().set_facecolor((0.9, 0.9, 0.9))
plt.grid(True)
plt.title("Gaussian Process on gp.dat")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.show()
```

In [ ]:

```
n_draws = 20
K_mod = cov_mat(x2, x2, ker=RBFkernel, sigma_sq=0.12) + sigma_sq*np.eye(x2.__len__())
ys = np.random.multivariate_normal(mean=mean, cov=K_mod, size=n_draws)#.reshape(-1, 1)

for i in range(n_draws):
    plt.plot(x2, ys[i])
plt.gca().set_facecolor((0.9, 0.9, 0.9))
plt.grid(True)
plt.title(r"20 Draws from $\mathcal{G}\ (\mu', k')$")
plt.show()
```