GP 101

Code

Overview

This notebook provides a quick start guide to building a Gaussian process model using only `numpy`, `scipy`, `pandas`, and `matplotlib`.

▼ Code

```
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal
from scipy.linalg import cholesky, solve_triangular
import seaborn as sns
```

Data

In this tutorial, we will use the Olympic gold dataset that we have used quite a few times in Lecture. First, we shall use `pandas` to retrieve the data.

▼ Code

```
df = pd.read_csv('data/data100m.csv')
df.columns=['Year', 'Time']
N = df.shape[0]
```

The three code blocks below define the kernel, a utility function for *tiling*, and the posterior calculation. To clarify, this is the predictive posterior distribution evaluated at some *test* locations, $\mathbf{X}_{*}$. The expression for both the predictive posterior mean and covariance are given by:

$$\mathbb{E}\left[\mathbf{y}_{*}|\mathbf{X}_{*}\right] = \mathbf{K}\left(\mathbf{X}_{*}, \mathbf{X}\right)\left[\mathbf{K}\left(\mathbf{X}, \mathbf{X}\right) + \sigma_{n}^{2}\mathbf{I}\right]^{-1}\mathbf{y}$$
$$Covar\left[\mathbf{y}_{*}|\mathbf{X}_{*}\right] = \mathbf{K}\left(\mathbf{X}_{*}, \mathbf{X}_{*}\right) - \mathbf{K}\left(\mathbf{X}_{*}, \mathbf{X}\right)\left[\mathbf{K}\left(\mathbf{X}, \mathbf{X}\right) + \sigma_{n}^{2}\mathbf{I}\right]^{-1}\mathbf{K}\left(\mathbf{X}, \mathbf{X}*\right)$$

▼ Code

```
def kernel(xa, xb, amp, ll):
    Xa, Xb = get_tiled(xa, xb)
    return amp**2 * np.exp(-0.5 * 1./ll**2 * (Xa - Xb)**2 )

def get_tiled(xa, xb):
    m, n = len(xa), len(xb)
    xa, xb = xa.reshape(m,1) , xb.reshape(n,1)
    Xa = np.tile(xa, (1, n))
    Xb = np.tile(xb.T, (m, 1))
    return Xa, Xb

def get_posterior(amp, ll, x, x_data, y_data, noise):
    u = y_data.shape[0]
    mu_y = np.mean(y_data)
    y = (y_data - mu_y).reshape(u,1)
    Sigma = noise * np.eye(u)

    Kxx = kernel(x_data, x_data, amp, ll)
    Kxpx = kernel(x, x_data, amp, ll)
    Kxpxp = kernel(x, x, amp, ll)

    # Inverse
    jitter = np.eye(u) * 1e-12
    L = cholesky(Kxx + Sigma + jitter)
    S1 = solve_triangular(L.T, y, lower=True)
    S2 = solve_triangular(L.T, Kxpx.T, lower=True).T

    mu = S2 @ S1  + mu_y
    cov = Kxpxp - S2 @ S2.T
    return mu, cov
```

▼ Code

```
Xt = np.linspace(1890, 2022, 200) # test data locations (years)

# Hyperparameters (note these are not optimized!)
length_scale = 7.0
amplitude = 0.8


noise_variance = 0.1
mu, cov = get_posterior(amplitude, length_scale, Xt, df['Year'].values, df['Time'].values, noise_v
```

▼ Code

```
Xt = Xt.flatten()
mu = mu.flatten()
std = np.sqrt(np.diag(cov)).flatten()
```

```
fig = plt.figure(figsize=(8, 5))
plt.plot(Xt, mu, '-', label=r'$\mu$', color='navy')
plt.fill_between(Xt, mu+std, mu-std, color='blue', alpha=0.2, label=r'$\sigma$')
plt.plot(df['Year'].values, df['Time'].values, 'go', label='Data', ms=8)
plt.xlabel('Years')
plt.ylabel('Winning times')
plt.legend()
plt.show()
```