CSE 571 - Robotics
Homework 1 - Bayes Filters and Gaussian Processes

Due Monday April 20th @ 11:59pm

This homework involves three writing assignments and two programming assignments in Python. In the first programming problem, you will be implementing a 1D Gaussian process for predicting outputs given training data. In the second programming problem, you will be using multiple Gaussian processes to learn the dynamics of a cartpole system based on interactions with a cartpole simulator. The zip file containing the code for this homework can be found on the class website (`https://courses.cs.washington.edu/courses/cse571/20sp/`).

Useful reading material: Lecture notes, **Chapter 2 of Probabilistic Robotics, Thrun, Burgard and Fox (pdf shared with class)** and Chapter 2 of Gaussian Processes for Machine Learning, Rasmussen and Williams (Available online at: `http://www.gaussianprocess.org/gpml/chapters/`)

Collaboration: Students can discuss questions, but each student MUST write up their own solution, and code their own solution. We will be checking code/PDFs for plagiarism.

Late Policy: This assignment may be handed in up to 5 days late (April 25th @ 11:59pm), at a penalty of 10% of the maximum grade per day.

1 Writing assignments

1.1 Conditional Independence [5 points]

Let X,Y and Z be three random variables. Assuming that X and Y are conditionally independent given Z:

$$p(x, y|z) = p(x|z)p(y|z)$$

show that:

$$p(x|z) = p(x|z, y)$$
$$p(y|z) = p(y|z, x)$$

1.2 Bayes Filter [15 points]

A vacuum cleaning robot is equipped with a cleaning unit to clean the floor. The robot has a binary sensor to detect whether a floor tile is clean or dirty. However, neither the cleaning unit nor the sensor are perfect. From

previous experience, you know that the robot succeeds in cleaning a dirty floor tile with a probability of

$$\mathbb{P}(x_{t+1} = \text{clean} | x_t = \text{dirty}) = 0.6,$$

where x_t is the state of the floor tile at time t and x_{t+1} is the resulting state after the action has been applied. Activating the cleaning unit when the tile is clean will never make it dirty. Assume the robot always cleans at every time t (i.e. the transition probabilities model the fact that the robot is cleaning the floor tile).

The probability that the sensor indicates that the floor tile is clean although it is dirty is given by $p(z_t = \text{clean} | x_t = \text{dirty}) = 0.2$, and the probability that the sensor correctly detects a clean tile is given by $p(z_t = \text{clean} | x_t = \text{clean}) = 0.6$.

Unfortunately, you have no knowledge about the current state of the floor tile. However, after cleaning the tile, the robot's sensor indicates that it is clean. Compute the probability that the floor tile is now clean after the robot has vacuum-cleaned it. Assume a prior distribution at time t as $p(x_t = \text{clean}) = c$, where $0 \leq c \leq 1$. Then, plot $p(x_{t+1} = \text{clean} | z_{t+1} = \text{clean})$ for $0 \leq c \leq 1$.

1.3 Gaussian Conditioning [20 points]

Let X and Y denote two scalar random variables that are jointly Gaussian:

$$\begin{aligned} p(x, y) &= \mathcal{N}(\mu, \Sigma) \\ &= \frac{1}{2\pi\sqrt{|\Sigma|}} \exp\left\{ -\frac{1}{2} \left(\begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} \mu_X \\ \mu_Y \end{bmatrix} \right)^\intercal \Sigma^{-1} \left(\begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} \mu_X \\ \mu_Y \end{bmatrix} \right) \right\}, \end{aligned}$$

where $\mu = \begin{bmatrix} \mu_X & \mu_Y \end{bmatrix}^\intercal$ and $\Sigma = \begin{bmatrix} \sigma_X^2 & \sigma_{XY}^2 \\ \sigma_{XY}^2 & \sigma_Y^2 \end{bmatrix}$ are the mean and covariance, respectively. Show that conditioning on Y results in a Gaussian over X:

$$\begin{aligned} p(x \mid y) &= \mathcal{N}(\mu_{X|Y}, \sigma_{X|Y}^2) \\ &= \frac{1}{\sqrt{2\pi}\sigma_{X|Y}} \exp\left\{ -\frac{1}{2} \frac{(x - \mu_{X|Y})^2}{\sigma_{X|Y}^2} \right\} \end{aligned}$$

with $\mu_{X|Y} = \mu_X + \frac{\sigma_{XY}^2}{\sigma_Y^2}(y - \mu_Y)$ and $\sigma_{X|Y}^2 = \sigma_X^2 - \frac{\sigma_{XY}^4}{\sigma_Y^2}$.

1.3.1 Hints

- Use the definition of the conditional distribution and "complete the square" to get the answer
- Given $p(x, y)$ is jointly Gaussian, the marginal distribution of Y is also Gaussian: $p(y) = \mathcal{N}(\mu_Y, \sigma_Y^2)$
- For a 2x2 matrix positive definite matrix $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$, $A^{-1} = \frac{1}{|A|} \begin{bmatrix} d & -c \\ -b & a \end{bmatrix}$, $|A| = ad - bc$

2 Programming problems

2.1 Preliminaries

We highly suggest you install Anaconda (`https://www.anaconda.com/`) or Miniconda (`https://docs.conda.io/en/latest/miniconda.html`) to manage a Python virtual environment. Using such a software, managing software packages (e.g. requiring certain version numbers) is much easier, and you can create a different virtual environment for each project. See this tutorial for more information: `https://linuxhint.com/anaconda-python-tutorial/`.

In the source code, we provide a `requirements.txt` file with the required Python packages to run the code. To create a conda environment and install the required packages, simply run:

```
conda create -n cse571_20sp python=3.6
source activate cse571_20sp
cd <code directory>
pip install -r requirements.txt
```

The above commands create an anaconda environment `cse571_20sp` with a python3.6 interpreter and activate the environment. It then installs necessary packages for this homework assignment. After the commands have completed, you can run a Python script by typing `python <script name>.py`.

Python version. There are two major versions of Python: Python2 and Python3. Code written in Python2 may crash or produce incorrect results in Python3. **In this homework you must make sure your code is compatible with Python3**. We will test your code with a python3.6 interpreter.

OS compatibility. Our code should work in both MacOS and Linux. If you encounter any compatibility issue feel free to reach us. We recommend you have Linux installed because many robotics software (e.g. ROS) only works well in Linux, and you may want to use them for your project.

2.2 Gaussian Process predictions (1D) [25 points]

In this assignment, you will implement a 1D Gaussian process that predicts outputs based on noisy training data. You will be given (noisy) 1D training data pairs $D_{train} = \{(x_1, y_1), (x_2, y_2) \dots\}$. Your task is to predict the output for a set of test queries $D_{test} = \{\hat{x}_1, \hat{x}_2, ...\}$, conditioned on the training data.

The assignment requires you to implement two separate kernel functions, namely the:

- **Squared Exponential Kernel**: This is the kernel we discussed in class.

$$k(x_i, x_j) = \sigma_f^2 \exp(-\frac{(x_i - x_j)^T M (x_i - x_j)}{2})$$

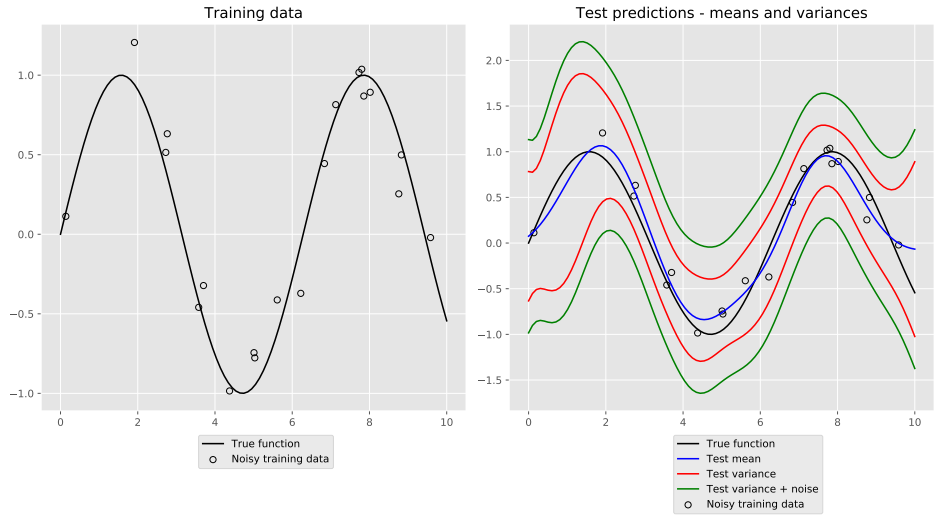

Figure 1: Gaussian process prediction using the Squared-Exponential kernel with default parameters.

where σ_f is a scale factor for the kernel and M is a metric measuring distance between two input vectors. In the 1D case, $M = \frac{1}{l^2}$ where l is the length scale of the kernel.

- **Matern Kernel**: This kernel is used commonly in many machine learning applications.

$$k(x_i, x_j) = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu} r}{l} \right)^{\nu} K_\nu \left(\frac{\sqrt{2\nu} r}{l} \right)$$

where ν and l are (positive) parameters of the kernel and $r = |x_i - x_j|$. K_ν is a modified bessel function and Γ is the gamma function. Good parameters settings for ν are 0.25 - 3.

The code for this section is in the file **gp1d.py** in the zip file. You are encouraged to see how the Gaussian process mean and uncertainty changes as more training points are given. You are also encouraged to play with the kernel parameters to see their effect on the GP. The code displays the training data and the predicted mean function and the respective variances (supplied by you) for the test queries. You can find the equations for the mean and variance predictions from the lecture slides. Fig. 1 shows results from a correct implementation of the SE kernel (using the default parameters).

2.3 Learning the dynamics of a cartpole using GPs [35 points]

In this assignment, you will implement Gaussian processes to approximate the dynamics of a cartpole system. A cartpole is an underactuated system with a pendulum fixed to a cart and is controlled by applying a force on the cart. We represent the state x and control u of the cartpole system as follows:

```
Cartpole state:
     dθ          [rad/s] angular velocity of the pendulum
```

```
     dp        [m/s]   velocity of cart
     θ         [rad]   angle of the pendulum
     p         [m]     position of cart
Control:
     u         [N]     force on cart
```

Our task is to learn a model of the cartpole dynamics:

$$x_{t+1} = f(x_t, u_t)$$

from data using Gaussian processes. We do this by interacting with a cartpole simulator which, over time provides us with the proper training data needed for the GP. We make two important modifications to the learning problem:

- First, we use an augmented state ($\hat{x}$) instead of the actual state (x) as input to the GP. The augmented state replaces the pendulum angle θ with the pair $[sin(\theta), cos(\theta)]$ in order to avoid the wrap-around issue with angles.
- Second, we predict delta-values of the state (dx_t), rather than the state directly. This reduces the difficulty of the problem as the model needs to only capture changes in state, centers the predictions (approximately) around zero and prevents wrap around issues when predicting θ.

We now have the following inputs and outputs to the GP:

```
GP dynamics model input:
     [dθ_t, dp_t, sin(θ_t), cos(θ_t), p_t, u_t]
GP dynamics model prediction:
     [Δdθ_{t+1}, Δdp_{t+1}, Δθ_{t+1}, Δp_{t+1}] # delta-state, not next state
```

The input of the GP is $6D$ while the output is $4D$. Since a GP only predicts a 1-dimensional output, we will train 4 separate GPs for this problem.

2.3.1 Learning the dynamics model

Let us now look at the training process for the Gaussian Process dynamics model. The system is setup to train in epochs, using data from the cartpole simulator to train a model that becomes more accurate with each epoch. Each epoch proceeds as follows:

1. At the start of an epoch, a random initial state x_0 and a sequence of H controls are chosen (either randomly or via a preset policy) $U = \{u_0, u_1, \ldots, u_{H-1}\}$. The code propagates these controls through the cartpole simulator to generate a rollout (trajectory) $\{x_0, u_0, \ldots, x_{H-1}, u_{H-1}, x_H\}$.
2. During this epoch, we will compute our GP predictions $\hat{x}_{t+1} = f_{\text{GP}}(x_t, u_t)$ for every t, and (visually) compare this with the actual state x_{t+1} to evaluate how well our GP captures the dynamics of the cartpole.

Additionally, the ground truth trajectory will be added to the training dataset for the GP **at the end of the current epoch**.

3. To (visually) evaluate how well our GP model is able to predict the dynamics of the cartpole, we make predictions using the GP model, conditioned on the simulator rollouts from previous epochs. ***You need to implement this step***. Alg 1 shows pseudocode for generating the rollout. We compute the mean and variance of the predicted next state using the GP and propagate the mean prediction across time. We call this a *Mean-Rollout* since we discard the actual next state distribution and rollout only the mean.

Algorithm 1 Mean-Rollout using GP (f_{GP}) - TO BE IMPLEMENTED

Inputs: Initial state: x_0, Training inputs: X, Training targets: Y.
Output: Future states: $\{x_1, x_2, \ldots, x_H\}$
for $t = 0 : H-1$ **do**
  $\hat{x}_t = g(x_t)$ ▷ Create augmented state
  for $k = 0 : 3$ **do** ▷ Predict using separate GP per output dimension
    $(\mu_t[k], \sigma_t[k]) = f_{GP}[k](\hat{x}_t, u_t)$ ▷ Predict delta-state mean and variance. $a[k] = k$th element of a
    $x_{t+1}[k] = x_t[k] + \mu_t[k]$ ▷ Compute next state's mean

4. The *Mean-Rollout* captures how well the GP does in expectation. In this step, we would like to see how the variance of the GP behaves as we predict across a long time horizon. Unlike the previous case, we will now make use of the complete Gaussian distribution predicted by the GP. Instead of using the mean delta-state, we will sample from the distribution around the mean, based on the predicted variance. This will result in trajectories that capture the uncertainties in the predictions over longer horizons. ***Once again, you need to implement this step***. Alg 2 shows the pseudocode for generating sampled rollouts. Ideally, to fully represent the uncertainty, we would generate multiple samples from the state distribution at each timestep of each trajectory. Unfortunately, this would result in the number of samples increasing exponentially over time. To avoid this, we only sample once from the distribution at each timestep and repeat this N times to generate the rollouts. In practice, these samples will be near the Mean-Rollout initially and slowly move away as the uncertainty increases over time.

Algorithm 2 Sampled Rollouts using GP (f_{GP}) - TO BE IMPLEMENTED

Inputs: Initial state: x_0, Training inputs: X, Training targets: Y.
Output: Sampled Future states (N samples per timestep): $\{x_1^0, x_2^0, \ldots, x_H^0\}, \{x_1^1, x_2^1, \ldots, x_H^1\}, \ldots, \{x_1^{N-1}, x_2^{N-1}, \ldots, x_H^{N-1}\}$
for $j = 0 : N-1$ **do** ▷ Generate N sampled trajectories
  for $t = 0 : H-1$ **do**
    $\hat{x}_t^j = g(x_t^j)$ ▷ Create augmented state. $\forall j,\ x_0^j = x_0$
    for $k = 0 : 3$ **do** ▷ Predict using separate GP per output dimension
      $(\mu_t^j[k], \sigma_t^j[k]) = f_{GP}[k](\hat{x}_t^j, u_t)$ ▷ Predict delta-state mean and variance. $a[k] = k$th element of a
      $s \sim \mathcal{N}(\mu_t^j[k], \sigma_t^j[k])$ ▷ Sample a delta-state from the predicted Gaussian
      $x_{t+1}^j[k] = x_t^j[k] + s$ ▷ Compute next sampled state

5. Finally, at the end of each epoch, we compare the predictions from the GP (Mean/Samples) with the ground-truth from the cartpole simulator. The system displays the mean trajectory, the N rolled out trajectories and plots the GP and simulator predictions with uncertainties. Also, the predictions from the cartpole simulator are added to the training data from the previous epoch. We use the augmented state and control pair $[\hat{x}_t, u_t]$ as the training inputs with the delta-states dx_t as the targets.

2.3.2 Remarks:

A few points to note:

- The output is 4-dimensional. You need to create a separate GP per output dimension.
- The kernel to use for this problem is the Squared Exponential kernel. Unlike the previous problem, the kernel inputs are 6-dimensional. There is a different length scale for each dimension of the input (6 values per GP) and for each GP (4 GPs in total).
- The hyper-parameters (for each GP) are given to you for this task. You are encouraged to modify the hyper-parameters and see how the system behaves.
- With a successful implementation, you will see that the system does quite poorly at the start (the rollouts are all over the place) and starts to improve very fast. The predictions should match well against the simulator with low variance. One case where the system fails to do well even with a lot of training data is when the pendulum is upright as even a small force there can cause large changes in the state.
- When sampling from a Gaussian to sample GP rollouts, please use np.random.RandomState.normal(). See the code for more details. This will allow our autograders to fix the random seed and grade your results accurately.
- When implementing mathematical code in Python/Numpy, try to minimize the amount of for loops. If a for loop can be replaced with vectorized computation, it will greatly increase the efficiency of the code. Here is a nice tutorial about this and other advantages of Numpy: `https://realpython.com/numpy-array-programming`.

The code to be modified for this part of the homework is in **cartpole_test.py**.

You can find a video of a correct implementation for the first 12 epochs at `https://courses.cs.washington.edu/courses/cse571/20sp/homeworks/cartpole.mp4`. Your results might be slightly different based on how you sample from the gaussian for the 10 rollouts. The results should have the general trend of high variance for the first few epochs (evidenced by the rollouts - transparent cartpoles moving all over) and lower variance as you get more data (tighter clumping of the rollouts).

3 Submission

You will be using Gradescope `https://www.gradescope.com/` to submit the homework. Please submit the written assignment answers as a PDF. For the code, please **submit gp1d.py and cartpole_test.py**.