Problem Set 3

Due April 26, 2024 at 11:59 PM Pacific

Turn in your problem set by Gradescope. Include any code you write in your solutions as well. Mark down how much time you spent on the homework set.

1. **Understanding Point Clouds and Sensor Data:** To familiarize you with working with 3D data (specifically point clouds), let's perform some simple data loading and processing with the `Open3D` (`https://www.open3d.org/docs/release/`) library in `Python`. Use the provided `terrain.ply`, `terrain-partial.ply`, and `terrain-noisy.ply` point cloud file. If you do not have python already installed, we recommend that you use Google Colab.

   (a) Load the point clouds using the `read-point-cloud` (`https://www.open3d.org/docs/latest/python_api/open3d.io.read_point_cloud.html`) function. It may be useful for you to look through the provided functions of the `PointCloud` class (`https://www.open3d.org/docs/release/python_api/open3d.geometry.PointCloud.html`).

   (b) Unfortunately, the point clouds have too many points, slowing down any algorithms we build on top of it. Let's uniformly downsample these point clouds (`https://www.open3d.org/docs/release/python_api/open3d.geometry.PointCloud.html#open3d.geometry.PointCloud.uniform_down_sample`). Downsample the number of points by 10 (if the later point cloud visualization and processing still takes too long on your computer or Google Colab, you can downsample up to 100). From here on, always use the downsampled point clouds unless explicitly noted otherwise.

   (c) Visualize the downsampled point clouds separately using the `draw` (`https://www.open3d.org/docs/release/python_api/open3d.visualization.draw_geometries.html`) function or plotly if in Google Colab to view the point cloud interactively. We recommend color-coding the points by the z dimension for additional clarity. Please ensure the axes are equal (i.e., the data is bounded and should be a long, flat rectangle rather than a cube). Include at least three views of the point clouds in your write-up.

   Note: If needed, you may use `matplotlib` to view the data, but you will have to set multiple views.

   (d) Your sensors will always have noise in them. Our job is to remove this noise as best as possible. Using the built-in methods of the `PointCloud` class, remove noise from `terrain-noisy.ply`. Once you've removed the noise, how do we quantify how good our denoised point cloud is? One way is through the one-way Chamfer Distance. Given two point clouds P_1 and P_2 (of size $|P_1|$ and $|P_2|$, respectively), the one-way Chamfer Distance is defined as

   $$CD(P_1, P_2) = \frac{1}{|P_1|} \sum_{x \in P_1} \min_{y \in P_2} ||x - y||_2^2. \tag{1}$$

   Report the Chamfer Distance between the partial terrain point cloud to the ground truth, the noisy point cloud to the ground truth, and the denoised point cloud to the ground truth.

   Hint: It is highly recommended to use `scipy.spatial.KDTree` to calculate these minimum distances. Make sure that P_2 is the ground-truth point cloud because we want to gauge how close our observed, smaller dataset is to the ground-truth (and not the other way around).

2. **Kernels in Gaussian Processes:** In class, you have seen Gaussian Processes (GPs) used to fit data of the form $y = f(x) + \epsilon$. In this problem, we ask you to implement one and investigate the impact of some parameters. Although a GP is a non-parametric model, the heart of a GP that defines its shape is the kernel, which contains parameters. Let's investigate the impact of several kernels to the fitting process by fitting a dataset `data.csv` produced by a time series on average stock prices.

   (a) Fill out the `compute-training-covariance` and `compute-posterior` function that takes in the training data and computes the training covariance and posterior statistics at sample points.

   (b) Fill out the `plot-gp` function to plot the posterior distribution (mean and two standard deviation).

(c) Implement a radial basis function kernel of the form $K(x_i, x_j) = \sigma^2 \exp[-\frac{1}{2\ell^2}(x_i - x_j)^T(x_i - x_j)]$ to fit the provided data. Use the `plot-gp` to plot the posterior on top of your training dataset as a scatter plot (and the full dataset as a lineplot). In addition, the dataset contains maximum and minimum daily stock prices. Plot these quantities as well and use them as a signal to tune your kernel parameters so that the maximum and minimum align with the standard deviation. Include this plot in your write-up. Comment on whether the maximum and minimum prices seem reasonable with respect to the posterior.

(d) Sometimes our data may also be periodic, in which case we have a stronger prior on the kernel. Implement an `exponential-sine-squared` kernel of the form $K(x_i, x_j) = \sigma^2 \exp[-\frac{1}{2\ell^2}\sin^2(\frac{\pi}{p}||x_i - x_j||^2)]$. Again, plot the posterior on top of your dataset, plot the maximum and minimum stock prices, and include the plot in your write-up.

(e) Sometimes our data may be explained by an underlying distribution that requires a more complicated kernel. Try multiplying the two kernels you've constructed into a third kernel, and perform the same procedure as (c) and (d). Which kernel performs the best for this data?

3. **Gaussian Processes for Sensors:** The typical filtering pipeline comprises a noisy **dynamics** model (i.e., $x_{t+1} = f(x_t, u_t) + w_t$) and a **sensor** model (i.e., $y_t = g(x_t, u_t) + v_t$). In this problem, we're asking you to fit a GP to sensor data, specifically to the provided terrain data retrieved from lidar from Problem 1.

(a) Using the GP functionality (radial basis kernel) you implemented in Problem 2, fit the **downsampled, partial, noisy** terrain data you saw in Problem 1. Tune the parameters (first try $\sigma = 1., \ell = 0.1$) to try to minimize the Mean Squared Error on the training data and report it.
Hint: We can approximately model gravity-aligned terrain data as $z = f(x, y)$.

(b) Using the (x, y) coordinates in the ground-truth point cloud, compute the mean elevation $\hat{z}$ as well as the standard deviation predicted by the GP. Report the Mean Squared Error on this test dataset.

(c) A total Mean Squared Error may not fully capture the spatial variability of the GP outputs across x and y. To better understand the GP, please generate at least the following visualizations from your computed $\hat{z}$ and standard deviation:

- the mean elevation terrain (the plot can be 3D, or it can be 2D with color-coding in $\hat{z}$)
- the standard deviation of the terrain (the plot can be 3D, or it can be 2D with color-coding in the standard deviation)
- the elevation error (the plot can be 3D, or it can be 2D with color-coding in $z - \hat{z}$

(d) Are the areas of high uncertainty where you would expect it to be?