TME 1 - Density Estimation

The goal of this TME is to familiarize yourself with density estimation algorithms : histogram estimation and kernel methods. We will use two classes (prototypes provided in the supplied code) - `Histogram` and `KernelDensity` - which inherit from the (abstract) class `Density`. The `fit(x)` function allows learning the estimator on the provided data, and the `predict(data)` function predicts the density for each point in `data`.

Complete the `Density` class with a `score(data)` method that returns the log-likelihood of the data `data` for the estimator. To handle points corresponding to zero density, it is customary to add a very small value (e.g., 10^{-10}) to each likelihood before taking the log (why ?).

Data : Google Places API, Points of Interest in Paris

Download the data archive from the course website. It contains a dictionary with information on different types of points of interest (POI) in Paris. Each key corresponds to a POI type (restaurant, bar, ...), and the associated value is a dictionary where the Google Maps ID serves as the key, and the information about the POI (GPS coordinates, rating, name, types, price level) serves as the value.

The provided code allows you to load the matrix of GPS coordinates and visualize the data.
Display different POIs on the map, such as restaurants and bars.

Histogram Method

Implement the methods for the `Histogram` class. Ideally, your class should handle data of any dimension. For the purposes of the TME, we will use only 2D data, so you can limit your methods to 2D functionality. The `steps` parameter in the constructor should allow specifying the number of discretization intervals for each dimension.
Hints :

- To estimate the histogram, you can use `np.histogramdd` ;
- For prediction, there is no `numpy` function to determine the indices in the histogram for a given point. You can use an intermediate function `to_bin(x)` to calculate these indices based on the discretization step and the histogram's min and max bounds.

Estimate the geographical density distribution of a type of POIs using the histogram method for a given discretization step.
Verify that you have a valid density : the function `get_density2D(f,data,steps)` evaluates a 2D problem estimator f on a grid defined by the data's min and max bounds for each dimension, with each dimension divided into `steps` intervals. It returns the estimation for each grid point, the discretization used for the first axis, and the second axis. What should your estimator satisfy on this grid if there are no errors ?
For some values of the histogram discretization step, display the obtained densities using the `show_density(f,data,ste` function. Which discretization seems best ?
Use a split into training and testing sets (or cross-validation) to estimate the likelihood on training and test data. What is the best empirical hyperparameter for the discretization step ?
Repeat the same experiments for a rarer POI type (e.g., "night_club"). Is the best hyperparameter of the same order as before ?

Kernel Methods

The kernel method is defined as :

$$f(\mathbf{x}_0) = \frac{1}{|\mathcal{X}|\sigma^d} \sum_{\mathbf{x}\in\mathcal{X}} \phi\left(\frac{\mathbf{x}_0 - \mathbf{x}}{\sigma}\right)$$

where ϕ is a kernel, σ is a hyperparameter, $\mathcal{X}$ is the training set, and d is the dimension of $\mathcal{X}$.

Implement the uniform kernel : the function `kernel_uniform(x)` that returns 1 if $|x^i| \leq \frac{1}{2}$ for all dimensions i, 0 otherwise. Your implementation should accept a matrix of examples `x` as input (not just a single data point) and return the function value for each example.

Implement the Gaussian kernel : the function `kernel_gaussian(x)` that returns $(2\pi)^{-d/2}e^{-\frac{1}{2}\|x\|^2}$.

Complete the `KernelDensity` class : the constructor takes the kernel function to be applied as an argument, the `fit(x)` function only records the training data passed as a parameter, and the `predict(data)` function must return the density estimation for each point in `data`.

Repeat the same experiments as for histograms :

- Verify that you obtain a valid density.
- Experiment with different kernels and hyperparameters, and visualize the results.
- Calculate the test likelihood based on the parameterization and plot likelihood curves for training and test data based on the parameterization.

Nadaraya-Watson Regression

The Nadaraya-Watson estimator enables classification and regression using methods very similar to kernel methods. Given a training set $\{\mathbf{x}_i, y_i\}_{i=1}^n \in \mathbb{R}^d \times \mathbb{R}$, the Nadaraya estimator is defined as $f(\mathbf{x}) = \sum_{i=1}^n \frac{y_i \phi(\frac{\mathbf{x}-\mathbf{x}_i}{\sigma})}{\sum_{j=1}^n \phi(\frac{\mathbf{x}-\mathbf{x}_j}{\sigma})}$.

Implement the `Nadaraya` class corresponding to this estimator. Test your implementation to predict a POI's rating based on its location. Vary the parameterization and measure performance using the mean squared error.

Bonus : Smoothing, COVID, and Pandas

The Nadaraya-Watson estimator is often used to "smooth" sequential data (e.g., temporal data ; moving averages are a special case of the estimator with a uniform kernel).

Download the open data on COVID in France in CSV format. Use the `pandas` module to extract the daily number of infections since the beginning of the pandemic (you can do this in almost one line, making it a simple exercise in using pandas). Plot the raw data, then experiment with different kernels and parameterizations to visualize the smoothing applied to the curve.