Exercise 1

Deadline: 6.11.2025 16:00.
Ask questions in discord to #ask-your-tutor

In this exercise, you will implement and test classical generative modelling methods.

Regulations

Please implement your solutions in form of *Jupyter notebooks* (`*.ipynb` files), which can mix executable code, figures and text in a single file. They can be created, edited, and executed in the web browser, in the stand-alone apps VSCode and JupyterLab, or in the cloud via Google Colab and similar services. We especially recommend https://www.bwjupyter.de, a service dedicated to teaching at universities in Baden-Würtemberg. They provide a web-based, installation free environment. The registration has very low barriers – all you need is your University account (UniID) to verify your affiliation to UHD. If packages are missing, you can easily install them temporarily within your profile (see https://www.bwjupyter.de/english/65.php for help), but the "PyTorch"profile should already have most of what you will need.

Create a Jupyter notebook `generative-baselines.ipynb` for your solution and export the notebook to PDF as `generative-baselines.pdf`. Zip all files into a single archive `ex01.zip` and upload this file to MaMPF before the given deadline.
Moreover, please set your **Anzeigename/display name** and **Name in Uebungsgruppen/name in tutorials** in MaMPF to your real name, which should be identical to your name in `muesli` and make sure you **join the submission** of your team via the invitation code before the submission deadline. Check out https://mampf.blog/handing-in-homework-assignments for instructions. Also note that joining a submission takes a while when you do it for the first time (later it will be just a single click), so don't wait until the last minute before the deadline.

1 Two-dimensional data

Use the function `sklearn.datasets.make_moons()` to create 2-dimensional training data sets of varying sizes. Use noise level 0.1 and ignore the labels. Implement a class for each of the following models following the sklearn conventions, i.e. the classes provide methods `fit(...)`, `sample(...)`, and `score_samples(...)`. Do not use the pre-defined classes and training algorithms from sklearn (but you may use fundamental functions to sample from a Gaussian, to compute a distance matrix, and the like).

1. a two-dimensional histogram
2. a single Gaussian
3. a Gaussian mixture model (GMM)
4. a kernel density estimator (KDE) with a ~~Gaussian kernel~~

The single Gaussian should give the same results as a GMM with one component – this makes for a useful test. Sampling from the histogram should be implemented such that first a bin is choosen according to the discrete distribution of bin weights, followed by a uniform draw from the region of the bin.

Implement the maximum mean discrepancy (MMD^2) metric with squared exponential and inverse multi-quadratic kernels for evaluation:

$$\mathrm{se}(x,x') = \exp\left(-\frac{||x-x'||^2}{h^2}\right) \qquad \mathrm{imq}(x,x') = \frac{1}{\sqrt{1+\frac{||x-x'||^2}{h^2}}}$$

Here, $||x - x'||^2$ is the squared Euclidean distance between x and x'. Play with the bandwidth h of the MMD kernels to make the metric informative about model quality (note that MMD^2 may be negative when computed from a finite data sample). Make sure to vectorize the MMD computations to make them reasonably efficient.

Evaluate the accuracy of your models by estimating the MMD^2 between test datasets from `make_moons` and the data generated by each model. Visualize the accuracies as a function of model hyperparameters (histogram: bin size, GMM: number of components, KDE: kernel bandwidth) and training set size. Comment on your findings.

For a number of representative models (both good and bad ones), create two 2D plots that (i) visualize the numerical values of the learned density (using the method `score_samples(...)` of your classes, evaluated on a sufficiently dense grid), and (ii) visualize a generated dataset from the model (using method `sample(...)`). Make sure to specify `axis('equal')` in these plots. Comment on model strengths and weaknesses. Bonus: Add some representation of the model solution to your plots (e.g. the grid of the histogram, some selected mixture components of the GMM).

2 Higher-dimensional data

Repeat the same tasks with the digits dataset (`sklearn.datasets.load_digits()`). Use the models and algorithms from sklearn this time. Since histograms don't scale to higher dimensions, you must use a density forest instead, i.e. a collection of histograms with adaptive bin sizes. An implementation of a suitable class is provided under https://tinyurl.com/HD-GNN-density-forest-py (see the external link on the MaMPF page). The usage of this module is explained in the docstring of class `DensityForest`.

Again, check model accuracy by MMD^2 and visualize generated data for some representative models (you do not need to visualize the numerical density values – this is hard in 64 dimensions). In addition, train a `sklearn.ensemble.RandomForestClassifier` on the original dataset to distinguish the 10 digit classes. Use this classifier to check for the models which create recognisable output if the 10 digits are generated in equal proportions.