Machine Learning

Assignment 1

Florian Thaler

March 18, 2025

Deadline: April 15, 2025
Submission: There are two different types of tasks: the **Pen and paper** category and the **Implementation** category. The former requires a written elaboration (calculations, explanations of results, observations, etc.), while the latter involves the implementation of specified functions and code blocks in the provided template files. Please use only these template files for the implementation. Additionally, please use only the packages included in the template files to solve the tasks.

Summarise all results of the 'Pen and paper' tasks in a PDF file named **report.pdf**. Upload this file and all your implementations to the TeachCenter. Please do not zip the files.

1 Pseudorandom number generation

In this exercise we consider methods for the generation of pseudorandom numbers of continuous distributions.

1.1 Inversion method (12P)

Let $F : \mathbb{R} \to [0, 1]$. The generalised inverse (or quantile function) $F^{(-1)} : [0, 1] \to \mathbb{R} \cup \pm\{\infty\}$ of F is defined by

$$F^{(-1)}(y) = \inf\{x \in \mathbb{R} \ : \ y \leq F(x)\}.$$

For $U \sim U(0, 1)$ one can prove that the CDF of the random variable $F^{(-1)}(U)$ equals F, or in other words: If the random variable X has CDF F, then X and $F^{(-1)}(X)$ are equally distributed - short $X \overset{d}{=} F^{(-1)}(U)$. This is the main principle the inversion method relies on.

```
Data: Samples $u_k$ of independent random variables $U_k \sim U(0,1)$, $k = 1, \dots, n$, generalised inverse
      $F^{(-1)}$
for $k = 1$ to $n$ do
    $x_k = F^{(-1)}(u_k)$
end
```

Algorithm 1: Inversion method

The sequence $\{x_k\}$ generated by this algorithm is an iid sample of F.

1. [**Pen and paper**] For each probability density function (PDF) compute its normalising constant c and find the generalised inverse of its CDF.
   a) $f(\cdot|a, b) : x \in \mathbb{R} \mapsto c\chi_{[a,b]}(x)$, $a, b \in \mathbb{R}, a < b$.
   b) $g(\cdot|\lambda) : x \in \mathbb{R} \mapsto ce^{-\lambda x}\chi_{[0,\infty)}(x)$, $\lambda \in (0, \infty)$
   c) $h(\cdot|\alpha, \beta) : x \in \mathbb{R} \mapsto c\frac{1}{x \log(\beta/\alpha)}\chi_{[\alpha,\beta]}(x)$, $\alpha, \beta \in (0, \infty), \alpha < \beta$

   Hint You may use the fact that on intervals where the CDF is continuous and strictly monotonically increasing, its generalised inverse is simply its inverse.

2. [**Implementation**] This task is about the implementation of the inversion method to sample from the distributions corresponding to the densities

$$f(\cdot|-1,1),\ g(\cdot|3),\ h(\cdot|3,5).$$

For this purpose implement the functions

- *pdf_f(...), pdf_g(...), pdf_h(...)*
- *inverse_cdf_f(...), inverse_cdf_g(...), inverse_cdf_h(...)*
- *generate_samples(...)*
- *visualise(...)*

in the template file **inversion_method.py**.

3. [**Pen and paper**] Use your implementation to generate random samples of size $n \in \{10^3, 10^4, 10^5\}$ of the distributions with densities $f(\cdot|-1,1)$, $g(\cdot|3)$, $h(\cdot|3,5)$. For each sample size n and each distribution provide plots illustrating your results.

4. [**Pen and paper**] Consider applying the inversion method to sample from the standard normal distribution. Unfortunately the CDF of $N(0,1)$ does not have a closed form, which makes the explicit computation of its generalised inverse difficult. Assume that there is a function *cdf_standard_normal()* which computes for a given input the CDF of $N(0,1)$ reasonably well[1]. Provide a pseudocode of an algorithm, which uses the inversion method and the function *cdf_standard_normal()* to generate samples from the standard normal distribution. Would you consider this an efficient nethod for sampling from the standard normal distribution?

1.2 Box-Muller method

Let us consider a method designed specifically for the task of sampling from a two-dimensional normal distribution. The method is based on the following statement:

Lemma (Box-Muller)**.** *Let $U, V \sim U(0,1)$ be stochastically independent random variables. Define $R = \sqrt{-2\log(U)}$ and $W = 2\pi V$. Then $X = R\cos(W)$, $Y = R\sin(W)$ are stochastically independent, standard normal distributed random variables.*

1. [**Pen and paper**] Prove the Box-Muller Lemma.

   Hint Compute the PDFs of R, Z and use the independence of U, V to obtain the joint PDF $f_{R,W}$. Recall the polar coordinate transform T and observe that $(X,Y) = f(R,W)$. Finally apply the transformation theorem for PDFs to obtain the joint PDF of X and Y. Compare the result with the PDF of the two-dimensional standard normal distribution.

2. Let $U_k, V_k \sim U(0,1)$, $k \in \mathbb{N}$ be random variables such that the set $\{U_k : k \in \mathbb{N}\} \cup \{V_k : k \in \mathbb{N}\}$ is stochastically independent. Then the sequence of random vectors $\{(X_k, Y_k)\}_{k\geq 1}$ generated by the Box-Muller scheme is a sequence of iid, standard normal distributed random vectors. Thus, similarly as for the inversion method we reduce the problem of the generation of a random sample from the two-dimensional standard normal distribution to the problem of sampling from the one dimensional uniform distribution on $(0,1)$.

   a) [**Implementation**] Let us now implement the Box-Muller scheme. For this purpose implement the functions
      - *box_muller_scheme(...)*
      - *visualise(...)*

      in the template file **box_muller_method.py**.

   b) [**Pen and paper**] Apply your implementation of the Box-Muller scheme and generate a sample of size $n = 1000$. Provide an illustration of your result.

[1]The Python package **scipy** provides such a function - see https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.norm.html.

c) [**Pen and paper**] Let

$$\mu_1 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, \mu_2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \Sigma_1 = \begin{bmatrix} 1/100 & 0 \\ 0 & 1 \end{bmatrix}, \Sigma_2 = \begin{bmatrix} 5 & 6 \\ 6 & 9 \end{bmatrix}.$$

Now apply your implementation to sample from $N_2(\mu_j, \Sigma_j), j = 1, 2$. Use the already implemented function *marginal_max_likelihood_estimators(...)* to compute the maximum likelihood estimators of the expected value and the variance of the marginal distributions. Compare the estimations with the true values. Additionally provide plots illustrating the generated samples.

Hint Recall the linear transformation theorem for the multivariate normal distribution and use the already implemented function *affine_linear_transformation(...)*. You may want to use the Cholesky method to find a suitable factorisation of Σ_2 - use the function *numpy.linalg.cholesky(...)* to compute it.

3. Consider the following rejection-type algorithm

```
Data: Samples $u_k, v_k$ of independent random variables $U_k, V_k \sim U(0,1)$, $k = 1, \ldots, n$
for $k = 1$ to $n$ do
    $\bar{u}_k = 2u_k - 1$, $\bar{v}_k = 2v_k - 1$
    $r_k = \bar{u}_k^2 + \bar{v}_k^2$
    if $r_k < 1$ then
        $x_k = \frac{\bar{u}_k}{\sqrt{r_k}}\sqrt{-2\log(r_k)}$, $y_k = \frac{\bar{v}_k}{\sqrt{r_k}}\sqrt{-2\log(r_k)}$
    end
end
```

Algorithm 2: Marsaglia-Bray algorithm

Note that this algorithm renounces on the application of sine and cosine, which makes it computationally less expansive than the Box-Muller scheme.

a) [**Pen and paper**] Explain why this algorithm generates random samples from the two-dimensional standard normal distribution. You don't need to give a comprehensive and rigourous proof here - it is enough to point out why the algorithm samples from the desired distribution.

Hint First note that due to the rejection step the algorithm generates (independent) realisations of random variables $\bar{U}, \bar{V}$, where $(\bar{U}, \bar{V})$ is uniformly distributed on the unit disk. Then, applying the inverse polar coordinate transform on to $(\bar{U}, \bar{(V)})$ infer that the squared radial component is uniformly distributed on $(0, 1)$. What can you say about the distribution of $\bar{U}^2 + \bar{V}^2$? Finally recall the definitions of sine and cosine.

b) [**Implementation**] Implement the function *marsaglia_bray_scheme(...)* in the template file **box_muller_method.py**.

c) [**Pen and paper**] Apply your implementation to generate a sample of size $n = 1000$ of $N_2(\mu_j, \Sigma_j), j = 1, 2$ with μ_j, Σ_j as before. Compare the results in terms of maximum likelihood estimates of the expected value and the variance of the marginal distributions with the results obtained from the Box-Muller scheme.

2 Monte-Carlo estimation (9P)

Consider a real-valued random variable Y and a real-valued function f. Then the Monte-Carlo estimation M_n of the expected value $E(f(Y))$ is given as

$$\frac{1}{n}\sum_{j=1}^{n} f(y_j),$$

where $y_j, j = 1, \ldots, n$ are assumed to be stochastically independent realisations of Y.

1. [**Pen and paper**] Use the law of large numbers to argue under which assumptions and why the Monte-Carlo method generates good approximations of expected values for large n.

2. [**Implementation**] Let $(X_1, X_2) \sim N_2(\mu, \Sigma)$, where

$$\mu = \begin{bmatrix} 2 \\ 3 \end{bmatrix}, \text{ and } \Sigma = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}.$$

In the template file **monte_carlo_method.py** implement the functions

- *f(...)*
- *monte_carlo(...)*

for the approximation of the expected value and the variance of $2X_1 + X_2$.

3. [**Pen and paper**] Apply the Monte-Carlo method for each sample size $n \in \{10^3, 10^4, 10^5\}$ and for samples generated by the Box-Muller scheme and the Marsaglia-Bray scheme. Compare the results with the exact values. How does the sample size n affect the approximation quality?

4. Let $U_1, \dots, U_n$ be independent $U(0,1)$-distributed random variables and let $Y_j = f(U_j)$, $j = 1, \dots, n$. Given an error bound ε, by means of the Chebyshev's inequality one can prove that

$$P(|\frac{1}{n}\sum_{j=1}^{n} Y_i - E(Y_1)| > \varepsilon) \leq \frac{Var(Y_1)}{n\varepsilon^2}. \tag{1}$$

Thus, the error of the Monte-Carlo method for the approximation of $E(Y_1)$ is basically governed by the ratio $Var(Y_1)/n$. To reduce the error, we may increase the sample size and/or sample from a distribution whose variance is less than $Var(Y_1)$. Let us focus on the latter, i.e. on a *variance reduction* technique.

a) Let Z be a random variable whose expexted value is well known or easy to compute[2]. Let $a \in \mathbb{R}$ and let $T = Y + a(Z - E(Z))$. Instead of applying the Monte-Carlo method to Y we apply the Monte-Carlo method to T. Since $E(T) = E(Y)$, this gives us an approximation of the expected value of Y.

i. [**Pen and paper**] We aim to choose the parameter a in such a way, that $Var(T) \ll Var(Y)$. According to (1), this improves the approximation quality of our estimate.

Find a such that $Var(T)$ is minimal. Interpret the result.

Hint Compute the $Var(T)$ for the optimal parameter a. Recall the definition and the meaning of the correlation coefficient.

ii. [**Implementation**] Let $U \sim U(0,1)$, and let $g : x \in \mathbb{R} \mapsto x(1-x)$. Let further the random variable Z be defined via $Z = h(U)$, where $h : \mathbb{R} \to \mathbb{R}$ is defined as

$$h(x) = \begin{cases} x/2 & \text{if } 0 \leq x < 0.5 \\ (1-x)/2 & \text{if } 0.5 \leq x \leq 1 \end{cases}$$

Implement the function *monte_carlo_variance_reduction(...)* in the template file **monte_carlo_method.py** for the approximation of the expected value of $g(U)$.

iii. [**Pen and paper**] Apply your implementation to approximate the expected value of $g(U)$. Compare the approximates with the exact value. What do you observe?

3 Bayesian image denoising (9P)

Let us consider the problem of removing noise from images. Let $\mathcal{X} = \{x_1, \dots, x_n\} \subset \mathbb{R}^d$ be a set of clean, noise-free images which we will use as training dataset. Let $\{y_1, \dots, y_m\} \subset \mathbb{R}^d$ be a sample

[2] Z is called control variate.

of noisy images. We assume that each noisy image y is the sum of a clean image x and Gaussian noise, i.e.

$$y = x + \sigma z,$$

where z is a realisation of the d-dimensional standard normal distribution and $\sigma \in \mathbb{R}_{>0}$ denotes the so-called noise level. Interpreting x, y as realisations of the random variables X and Y respectively, this assumption can be reformulated as

$$Y = X + Z, Z \sim N(0, \Sigma),\ \Sigma = \operatorname{diag}(\sigma^2, \dots, \sigma^2). \tag{2}$$

We approach the denoising problem from a Bayesian perspective. Given a noisy image y, our goal is to obtain a denoised version x^* of it by means of the maximum a posteriori (MAP) estimate:

$$x^* \in \operatorname*{argmax}_{x \in \mathcal{X}} f_{Y|X=x}(y) P(X = x), \tag{3}$$

or equivalently

$$x^* \in \operatorname*{argmax}_{x \in \mathcal{X}} \log(f_{Y|X=x}(y)) + \log(P(X = x)).$$

The function $f_{Y|X=x}$ denotes the conditional density of Y given the observation x, or in other words, the likelihood function of our model (2). The model assumption allows to derive a formula for the conditional density - see task 1 in section 3.2. The latter term in (3), i.e. the prior, is approximated using kernel density estimation (KDE).

3.1 Kernel density estimation

1. [**Pen and paper**] Let $\mathcal{X} = \{x_1, \dots, x_n\} \subset \mathbb{R}^d$ denote a data sample - not necessarily a set of images. Let $k : \mathbb{R}^d \to \mathbb{R}$ and let $h > 0$. The KDE w.r.t. the kernel k is defined via

$$f_h(x) = \frac{1}{n} \sum_{j=1}^{n} (1/h)^d k(\frac{x - x_j}{h}).$$

   Assuming that k is a PDF, prove that f_h defines a PDF on $\mathbb{R}^d$.

2. [**Pen and paper**] The parameter h, the so-called bandwidth, has a major influence on the KDE. Let us study its impact in the case where $d = 1$ and k corresponds to the density of the standard normal distribution[3]. In *kernel_density_estimation.py* a Gaussian-KDE is used to approximate the density of a data sample $\mathcal{X} = \{x_1, \dots, x_7\}$. Apply the function *gaussian_kde(...)* to each $h \in \{0.03, 0.1, 0.3, 1.0\}$ and use the function *visualise_kde(...)* to visualise the resulting KDEs. Interpret the results:
   - What do you observe for small and large values of h? What happens in the limiting cases $h \to 0^+$, $h \to \infty$?
   - How does the bandwidth h influence the sampling process if we wanted to sample from the distribution with density f_h?

3.2 Denoising MNIST data

Let us now come back to the denoising task.

1. [**Pen and paper**] Let X, Z be stochastically independent and let x be a realisation of X such that $P(X = x) > 0$. Then for the conditional density $f_{Y|X=x}$ it holds that

$$f_{Y|X=x}(y) = \frac{1}{(2\pi\sigma^2)^{d/2}} \exp((-1/2\sigma^2)\|y - x\|_2^2)$$

   Prove this statement for univariate random variables X, Y, Z.

   Hint Simplify the probability $P(Y \leq y \cap X = x)$ using the assumption on Z and use the independence assumption to compute the conditional probability $P(Y \leq y | X = x)$. From this expression derive the sought density.

[3]In this case we refer to it as Gaussian-KDE

2. [**Pen and paper**] Let p_h denote the Gaussian-KDE of the prior. Verify analytically that

$$\log f_{Y|X=x}(y) = -\frac{d}{2}\log(2\pi\sigma^2) - \frac{\|y-x\|_2^2}{2\sigma^2}$$

and that

$$\log p_h(x) = \log(\sum_{j=1}^{n}(\exp(-\log(n) - \frac{d}{2}\log(2\pi) - d\log(h) - \frac{\|x-x_j\|_2^2}{2h^2})).$$

3. [**Implementation**] This task is about the implementation of the denoising scheme introduced and discussed above. In the template file *bayesian_denoising.py* implement the following functions

   - *log_gaussian_kde(...)*
   - *log_likelihood(...)*

   Do not use any loops, or iterators in your implementation.

4. [**Pen and paper**] Let us apply the implementation to the sign language MNIST dataset[4]. We consider in this example only the set of images representing the letter A.
   Use a training sample of size $n = 1000$ and a test sample of size $n = 10$. Test your application for the bandwidth $h = 0.1$ and each noise level $\sigma \in \{0.05, 0.1, 0.25\}$. Provide for each scenario a single plot showing all the noisy test images and the corresponding denoised images. How does your model perform? Do you observe any differences w.r.t to different values of σ?

[4]See for example `https://github.com/namas191297/sign_language_mnist_cnn`.