Probabilistic Decision Making VU, WS25
Assignment 2
Bayesian Logistic Regression, Monte Carlo, Bayesian Networks

Thomas Wedenig
thomas.wedenig@tugraz.at

Teaching Assistant:	Leon Tiefenböck
Points to achieve:	25 Points
Deadline:	05.12.2025 23:59 (strict, no late submissions allowed)
Hand-in procedure:	You can work in groups of **at most two people**. **Exactly one** team member uploads four files to TeachCenter: **The report (.pdf)**, `bayesian_log_reg.py`, `monte_carlo.py`, `bayesian_nets.py`. The first page of the report must be the **cover letter**. Do not rename the Python files. Do not upload a folder. Do not zip the files. We do not accept submissions via means other than TeachCenter.
Plagiarism:	If detected, we grade *all involved parties* with "Ungültig aufgrund von Täuschung"

General Remarks

Your submission will be graded based on:

- Correctness (Is your code doing what it should be doing? Is your derivation correct?)
- The depth of your interpretations (Usually, only a couple of lines are needed.)
- The quality of your plots (Is everything clearly readable/interpretable? Are axes labeled? …)

Remarks:

- All results (i.e., plots, results of computations) should be included in your PDF report.
- Report all intermediate steps in pen & paper exercises—only presenting the final solution is insufficient.
- Your submission must run with Python 3.11.13 and the package versions listed in `requirements.txt`.
  - Check TeachCenter for instructions to setup a conda environment.
- Do not use any external packages except for the ones listed in `requirements.txt`.
- **Do not modify the function signatures** of the provided functions.
  - i.e., do not edit the function names and inputs
- Do not use Large Language Models (LLMs) to generate any part of your solution.
  - Evident LLM usage will be treated as plagiarism.

Failure to adhere to these rules may result in point deductions.

Task 1 — Bayesian Logistic Regression [14 Points]

Similar to Bayesian Linear Regression, we now consider *Bayesian Logistic Regression*: As in standard logistic regression, we wish to solve a *binary classification problem*, i.e., given a dataset with features and binary labels

$$\mathcal{D} = \left\{ \left(\mathbf{x}^{(i)}, y^{(i)} \right) \right\}_{i=1}^{N} \quad \text{with } \mathbf{x}^{(i)} \in \mathbb{R}^d, y^{(i)} \in \{-1, +1\},$$

we wish to predict the label y from the feature vector $\mathbf{x}$. In logistic regression, we also make use of a linear-in-the-parameters model:

$$f(\mathbf{x}) \;=\; \phi(\mathbf{x})^\top \boldsymbol{\theta},$$

where $\phi : \mathbb{R}^d \to \mathbb{R}^D$ is a (possibly non-linear) feature transformation and $\boldsymbol{\theta} \in \mathbb{R}^D$ again denote the parameters of the model. In all tasks, we will have $d = 2$ for the input data.

As y are binary labels, the likelihood $p(y \mid \mathbf{x}, \boldsymbol{\theta})$ must now be a Bernoulli distribution, and we model it as

$$p(y \mid \mathbf{x}, \boldsymbol{\theta}) = \begin{cases} \sigma(f(\mathbf{x})) & \text{if } y = 1 \\ 1 - \sigma(f(\mathbf{x})) & \text{if } y = -1 \end{cases} \qquad \text{where} \quad \sigma(z) = \frac{1}{1 + \exp(-z)}$$

where σ is called the *(logistic) sigmoid function*. Noticing that $\sigma(-z) = 1 - \sigma(z)$, we can also write the likelihood as

$$p(y \mid \mathbf{x}, \boldsymbol{\theta}) = \sigma(y \cdot f(\mathbf{x}))$$

We denote

$$X := \begin{bmatrix} (\mathbf{x}^{(1)})^\top \\ \vdots \\ (\mathbf{x}^{(N)})^\top \end{bmatrix}, \quad \phi(X) := \begin{bmatrix} \phi(\mathbf{x}^{(1)})^\top \\ \vdots \\ \phi(\mathbf{x}^{(N)})^\top \end{bmatrix}, \quad \mathbf{y} := (y^{(1)}, \ldots, y^{(N)})^\top.$$

As in Bayesian Linear Regression, we will place a *Gaussian prior* on the parameters $\boldsymbol{\theta}$:

$$p(\boldsymbol{\theta}) := \mathcal{N}(\boldsymbol{\theta}; \boldsymbol{\mu}, \Sigma)$$

In contrast to Bayesian Linear Regression, the posterior $p(\boldsymbol{\theta} \mid \mathbf{y}, X)$ cannot be computed analytically. We will thus approximate it using a (1) Laplace Approximation, and a (2) brute-force Monte-Carlo based approach.

All code you write should go into `bayesian_log_reg.py`. We will use `jax` and the `numpy`-like API it exposes (`jax.numpy`, called `jnp` in the code) to implement these tasks, since we will leverage automatic differentiation in some tasks. Carefully read the provided code skeleton and become familiar with it.

Task 1.1 [0.5 points] We start with `classify_linear_data`, where you will work on data X which can be almost perfectly seperated with an affine decision function. In `classify_linear_data`, define an appropriate feature transformation ϕ (called `phi_fn` in the code) for this dataset. This will be used to compute $\phi(X)$, so assume the input to `phi_fn` is a matrix of shape (N, d) and the output is a matrix of shape (N, D). Pick parameters $\boldsymbol{\mu}$ and Σ for the Gaussian prior $p(\theta)$ (e.g., standard Gaussian prior with mean $\mathbf{0}$ and covariance I).

Task 1.2 [1.5 points]

- *Prior*: Implement `log_prior`, which should return $\log p(\boldsymbol{\theta})$. Make use of the already imported function `logpdf_mvnorm`.
  - On pen and paper, show that for a Gaussian prior $p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}; \boldsymbol{\mu}, \Sigma)$ and any points $\mathbf{x}_*^{(1)}, \ldots, \mathbf{x}_*^{(N_*)}$ collected in the matrix $X_* \in \mathbb{R}^{N_* \times d}$, the induced prior over the vector of latent function values $\mathbf{f}_* := \phi(X_*)\boldsymbol{\theta}$ is Gaussian. Write down its mean and covariance in terms of $\boldsymbol{\mu}$, Σ, and X_*.

- *Likelihood*: In your report, write down $p(\mathbf{y} \mid X, \boldsymbol{\theta})$ by making use of the i.i.d. assumption of the $(\mathbf{x}, y)$ pairs. Then, write down $\log p(\mathbf{y} \mid X, \boldsymbol{\theta})$ and implement it in `log_likelihood`.
- *Posterior*: In your report, write down the posterior $p(\boldsymbol{\theta} \mid \mathbf{y}, X)$ as a function of prior and likelihood using Bayes' law.
  - Is the posterior Gaussian? Why/why not?

  Dropping the normalization constant, we get the unnormalized posterior $\tilde{p}(\boldsymbol{\theta} \mid \mathbf{y}, X)$, which is proportional to the true posterior. Implement $\log \tilde{p}(\boldsymbol{\theta} \mid \mathbf{y}, X)$ in `log_unnormalized_posterior` by making use of `log_prior` and `log_likelihood`.

Task 1.3 [1 point] For plotting, it is convenient to have a function that takes a `jnp.array X_grid` $\in \mathbb{R}^{N_* \times d}$ and returns a `jnp.array` of probabilities, i.e.,

$$\texttt{X\_grid} \mapsto [\, p(y = 1 \mid \mathbf{x}^{(1)}, \boldsymbol{\theta}),\ \ldots,\ p(y = 1 \mid \mathbf{x}^{(N_*)}, \boldsymbol{\theta}) \,]$$

where $\mathbf{x}^{(i)}$ is the i-th row of `X_grid`. Implement `get_likelihood_per_x_fn`, which should return this *function* (that takes `X_grid` as input and returns the probabilities).

Task 1.4 [1.5 point] In `bayesian_logistic_regression`, sample `num_samples_plots` times out of the Gaussian prior $p(\boldsymbol{\theta})$. For each sample $\boldsymbol{\theta}$, visualize the model probabilities $p(y = 1 \mid \mathbf{x}, \boldsymbol{\theta})$ using `plot_prob_fns`, where this function accepts a list of functions that accept `X_grid` and return the probabilities. Include this plot in your report and briefly explain what you see.

Task 1.5 [1.5 points] Implement `get_map_estimate`, which receives the negative log of the unnormalized posterior, i.e., a function $\boldsymbol{\theta} \mapsto -\log \tilde{p}(\boldsymbol{\theta} \mid \mathbf{y}, X)$ and returns the maximum a posteriori (MAP) estimate $\boldsymbol{\theta}_{\text{MAP}} := \text{argmax}_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta} \mid \mathbf{y}, X)$.

- Note that we cannot analytically compute $\log p(\boldsymbol{\theta} \mid \mathbf{y}, X)$, but only $\log \tilde{p}(\boldsymbol{\theta} \mid \mathbf{y}, X)$. How can we still find $\boldsymbol{\theta}_{\text{MAP}}$? In other words, show that

$$\boldsymbol{\theta}_{\text{MAP}} := \operatorname*{argmax}_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta} \mid \mathbf{y}, X) = \operatorname*{argmax}_{\boldsymbol{\theta}} \log \tilde{p}(\boldsymbol{\theta} \mid \mathbf{y}, X)$$

Use gradient descent with a suitable step size and a fixed, suitable number of iterations[1]. Pick a sensible initial point `theta_init` in the `bayesian_logistic_regression` function. The function `neg_log_unnorm_posterior` is already provided for you, but note the use of `jax.jit`: Briefly comment on what it does. What happens if you remove it?

Task 1.6 [1.5 points] Since the posterior cannot be computed analytically, we will approximate it using a *Laplace Approximation*, where we approximate the posterior with a Gaussian:

$$p(\boldsymbol{\theta} \mid \mathbf{y}, X) \approx \mathcal{N}(\boldsymbol{\theta}; \boldsymbol{\theta}_{\text{MAP}}, \Sigma_{\text{Laplace}}) \quad \text{where} \quad \Sigma_{\text{Laplace}} = \left(-\nabla^2_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta} \mid \mathbf{y}, X) \big|_{\boldsymbol{\theta} = \boldsymbol{\theta}_{\text{MAP}}} \right)^{-1}$$

where Σ_{Laplace} is the covariance of the Laplace approximation.

Implement `laplace_approximation`, which receives the MAP estimate $\boldsymbol{\theta}_{\text{MAP}}$ and the negative log of the unnormalized posterior, and returns the parameters of the *Laplace approximation* of the posterior, i.e., $\boldsymbol{\theta}_{\text{MAP}}$ and Σ_{Laplace}. Report Σ_{Laplace} in your report and briefly interpret it.

[1] Note: The function we minimize here is convex in $\boldsymbol{\theta}$, and there are much more efficient ways to optimize such smooth convex functions (e.g. Newton's method, or (L)-BFGS). We use plain gradient descent mostly for simplicity.

Task 1.7 [3 points] Consider an arbitrary test point $\mathbf{x}_* \in \mathbb{R}^d$ for which we want to make a prediction for its label y_*. We should thus consider the *posterior predictive*

$$p(y_* = 1 \mid \mathbf{x}_*, X, \mathbf{y}) = \int \underbrace{p(y_* = 1 \mid \mathbf{x}_*, \boldsymbol{\theta})}_{\text{Likelihood}} \underbrace{p(\boldsymbol{\theta} \mid \mathbf{y}, X)}_{\text{Posterior}} \, \mathrm{d}\boldsymbol{\theta}$$

which integrates out the unknown parameters $\boldsymbol{\theta}$.

Even after approximating the posterior with a Gaussian, we still cannot compute this analytically. We will thus approximate it in two ways: (1) using a *Monte Carlo approach*, and (2) using a *MacKay's analytic approximation* [1].

- *Monte Carlo Approach*: Sample `num_samples_mc` times out of the Laplace approximation of the posterior and compute a Monte Carlo estimate of the posterior predictive (function `posterior_pred_mc`).
- *MacKay's analytic approximation*: Assuming $\boldsymbol{\theta} \sim \mathcal{N}(\boldsymbol{\theta}; \boldsymbol{\theta}_{\text{MAP}}, \Sigma_{\text{Laplace}})$, what's the distribution of $\phi(\mathbf{x}_*)^\top \boldsymbol{\theta}$ for arbitrary test point $\mathbf{x}_* \in \mathbb{R}^d$? Write it down and analytically compute its parameters. Then, use the following approximation

  $$\mathbb{E}_{z \sim \mathcal{N}(m, s^2)}[\sigma(z)] \approx \sigma\left(\frac{m}{\sqrt{1 + \pi s^2/8}}\right)$$

  to approximate the posterior predictive (function `posterior_pred_mackay`).

Include all plots in your report. Compare the two approximations, and compare them with the plot obtained via the MAP estimate. Comment on the results.

Task 1.8 [1.5 points] For low-dimensional problems (e.g., $D \leq 3$), we can also (numerically) sample from the posterior $p(\boldsymbol{\theta} \mid \mathbf{y}, X)$ directly: To this end, we will discretize the parameter space and sample from the posterior in a grid.

Create such a grid in `brute_force_posterior_sampling`: For each dimension $i \in \{1, \ldots, D\}$ of the parameter space, create a `jnp.linspace` between $(\boldsymbol{\theta}_{\text{MAP}})_i \pm k \cdot \sqrt{(\Sigma_{\text{Laplace}})_{ii}}$, where k is called `num_stds_per_axis` in code. The number of grid points per axis is called `num_grid_points_per_axis`. The rest of the sampling function is provided for you. Read it carefully and explain what it does in your report. What does `jax.vmap` do? What does `jax.scipy.special.logsumexp` do?

Use the samples to directly compute a Monte Carlo estimate of the posterior predictive. Use the function `posterior_pred_brute_force_mc` to implement this. Include all plots in your report. Discuss the computational complexity of the brute-force approach compared to the Laplace approximation approach.

Task 1.9 [2 points] In `classify_moons_data`, we will repeat these experiments with a non-linear dataset (still $d = 2$). Here, we now use radial basis functions (RBFs) as our feature transformation ϕ, and pick $D = 501$. Again, pick reasonable parameters $\boldsymbol{\mu}$ and Σ for the Gaussian prior $p(\theta)$ (e.g., standard Gaussian prior with mean $\mathbf{0}$ and covariance I).

Run all experiments that you have performed in the previous tasks and include all plots in your report. Can you still use the brute-force approach to sample from the posterior? Why/why not? Discuss the difference between the plot that uses the MAP solution as a point estimate, and the plots that show the posterior predictive.

Task 2 – Simple Monte Carlo Simulation [1.5 Points]

In Assignment 1, we found the expected value of random variables by solving the corresponding integral (or sum). It is possible to work the other way around, that is, solve an integral by expressing it as an expectation and then using simulation to get the approximated value. Consider the integral

$$\int_0^{2\pi} \int_0^{2\pi} \sin(\sqrt{x^2+y^2} + x + y)\,\mathrm{d}x\,\mathrm{d}y$$

This integral is analytically challenging. While it is not an expectation, it can be formulated as the product of a constant, K, times an expectation using a suitable choice of joint density $p(x, y)$.

$$\int_0^{2\pi} \int_0^{2\pi} \sin(\sqrt{x^2+y^2} + x + y)\,\mathrm{d}x\,\mathrm{d}y = K \cdot \mathbb{E}_{(x,y)\sim p(x,y)}[\sin(\sqrt{x^2+y^2} + x + y)]$$

1. Find $p(x, y)$ and K such that the above identity is true.
2. In `monte_carlo.py`, approximate this integral using Monte Carlo simulation (use at least $5 \cdot 10^6$ samples from the joint distribution $p(x, y)$). Report the final value in your report, along with the number of samples you have used.
3. Use `scipy.integrate.nquad`[2] as a different way to compute this integral numerically and compare the output to the Monte Carlo estimate. Report the result, the estimated error, and the number of function evaluations (use `full_output=True` to get this information).

Task 3 – Bayesian Networks [9.5 Points]

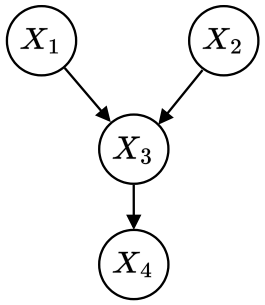

Figure 1: "Extended collider:" A collider $X_1 \to X_3 \leftarrow X_2$ with a descendant X_4.

Task 3.1 [1.5 points] Consider the Bayesian network structure given in Figure 1 (let's call it an "extended collider") and assume that random variables X_1, X_2, X_3 and X_4 are binary, i.e. taking values in $\{0, 1\}$. Provide conditional probability distributions (CPDs) such that in the resulting Bayesian network we have $X_1 \not\perp\!\!\!\perp X_2 \,|\, X_4$. Demonstrate this by showing that the conditional $p(X_1, X_2 \,|\, X_4)$ does *not* factorize as $p(X_1, X_2 \,|\, X_4) = p(X_1 \,|\, X_4)\, p(X_2 \,|\, X_4)$ for your choice of CPDs.

Task 3.2 [1.5 points] Consider the Bayesian network structure in Figure 2. For each of the following statements, state if they are true for all probabilistic models that follow this DAG. For each answer, give a brief explanation why you think that the statement holds in general (or does not hold in general).

1. $X_2 \perp\!\!\!\perp X_8 \,|\, X_5$

[2] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.nquad.html`

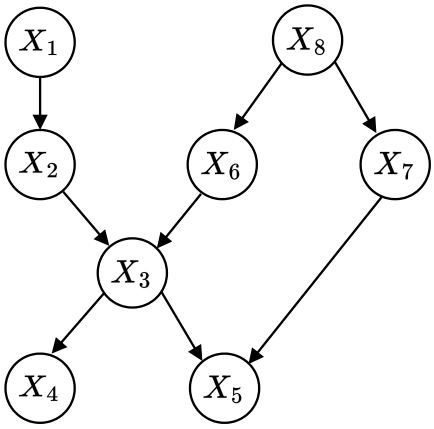

Figure 2: A Bayesian Network over random variables $X_1, \ldots, X_8$.

2. $X_6 \perp\!\!\!\perp X_7 \,|\, X_8$
3. $X_1 \perp\!\!\!\perp X_8 \,|\, X_6$
4. Can we always write $p(X_6 \,|\, X_7, X_8) = p(X_6 \,|\, X_8)$?
5. Can we always write $p(X_1, X_8 \,|\, X_6) = p(X_1 \,|\, X_6)p(X_8 \,|\, X_6)$?
6. $X_1 \perp\!\!\!\perp X_8 \,|\, X_6, X_5$

Task 3.3 [1 point] Again, consider the Bayesian network structure in Figure 2. For each statement, find a *minimal* set of random variables $\mathcal{X} \subseteq \{X_1, \ldots, X_8\}$ such that the statement is true in all probabilistic models that follow this DAG.

1. $X_7 \not\perp\!\!\!\perp X_1 \,|\, \mathcal{X}$
2. $X_7 \perp\!\!\!\perp X_6 \,|\, X_5, \mathcal{X}$
3. $X_1 \perp\!\!\!\perp X_8 \,|\, X_5, \mathcal{X}$

Task 3.4 [3 points] Consider the Bayesian network structure in Figure 2 and assume that all random variables X_i are binary. This network is defined in the `BayesianNetwork` class in `bayesian_nets.py`.

- Implement the `sample_joint_distribution` method to sample N times from the joint distribution using ancestral sampling.
- Then, implement `maximum_likelihood_estimation`, which computes a maximum likelihood estimator (MLE) of the Bayesian Network parameters (i.e., the CPDs), given the N samples you have drawn.
- Compare the estimated parameters to the true parameters by plotting a bar chart using `plot_params` for all $N \in \{10, 100, 10^4, 10^6\}$. Include the plots into your report and discuss them. Is the convergence equally fast for all estimated Bernoulli parameters? Write down the expectation and variance of the MLEs as a function of the true network parameters and the number of samples used to estimate them.
- What happens when some true Bernoulli parameter of the network is exactly 0 (or 1)? How would you change your implementation to make it robust against this? What would happen to the variance of your MLE in this case?

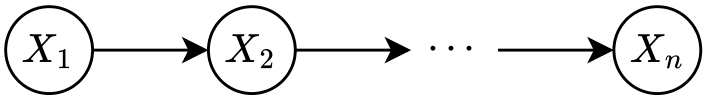

Figure 3: A Markov chain over random variables $X_1, \ldots, X_n$.

Task 3.5 [2.5 points] Consider the Bayesian network in Figure 3: This network is called a Markov chain (of length n). This structure is defined in the `MarkovChainBayesianNetwork` class in `bayesian_nets.py`. Again, assume that all random variables X_i are binary.

- Implement the `compute_joint_distribution` method to compute the full joint distribution of all random variables. This method should return an array of floats, where the first element corresponds to $p(X_1 = 0, X_2 = 0, \ldots, X_n = 0)$, the second element to $p(X_1 = 1, X_2 = 0, \ldots, X_n = 0)$, ..., and the last element to $p(X_1 = 1, X_2 = 1, \ldots, X_n = 1)$.
- Compute the marginal of the last random variable $p(X_n)$ using an explicit sum operation on the joint distribution you have just computed. Briefly discuss the computational complexity of this approach as a function of n.
- Then, implement `compute_marginal_variable_elimination`, which should compute the marginal distribution $p(X_i)$ using the *variable elimination* (VE) algorithm (the index $1 \leq i \leq n$ is supplied to the method). You do not have to implement VE in the most general case—it is sufficient to implement it for the special case of computing the marginal of a single variable in a Markov chain. Discuss the computational complexity of this algorithm as a function of n.
- Set $n = 22$. Use python's `time` module to measure the time it takes to compute the marginal $p(X_n)$ by (1) computing the full joint and explicitly summing over all other variables, and (2) using the `compute_marginal_variable_elimination` method. Report the results and discuss them briefly.

Task 4 – Peer Review [0 Points]

Each group member must write a single paragraph outlining their opinion on the work distribution within the group. Did every group member contribute equally? Did you split up tasks in a fair manner, or jointly worked through the exercises? Do you think that some members of your group deserve a different grade from others?

References

[1] David John Cameron Mackay. *Bayesian methods for adaptive models.* California Institute of Technology, 1992.