Start: 15.4.2024
Return: 29.4.2024
https://classroom.github.com

Assignment 1
Introduction to Probabilistic Machine Learning

1 Introduction

1.1 GitHub Classroom and the GitHub Repo

In this assignment, you will learn how to represent distributions and the related operations in Julia. In Part 3, you will study Gaussian distributions from a theoretical perspective.

In all practical tasks, please fill out all parts marked with #TODO#. This document, the annotations in the code and the test sets provided with the code aim to assist you on your path.

In all theoretical tasks, please provide your solutions as PDF with the respective name. You can use for example Word or Tex to generate this PDF.

Part 1: Type-Safe Discrete Distribution and 1D Gaussian Distribution Julia Module (2 Points)

- Relevant source files: `discrete.jl`, `gaussian.jl`
- Test files: `test*_discrete.jl` and `test*_gaussian.jl`

For the following lessons and assignments, we need Julia implementations of common probability distributions, that incorporate the ability to multiply and divide those distributions with each other During this assignment, we will focus on the following two distributions:

- A **discrete distribution** over the values $1, \ldots, n$ characterized by the log-probabilities of $1, \ldots, n$.
- A **1D Gaussian distribution** over $\mathbb{R}$ characterized by the precision mean τ and precision ρ.

The file `discrete.jl` should implement a discrete distribution as a parametric Julia type (with the total number of outcomes as the type variable) with the following interfaces:

- An external constructor with one parameter `N` of type `Int64` that returns a `DiscreteN` distribution with uniform probabilities.
- An external constructor with one parameter `logP` of type `VectorFloat64` that takes the log-probabilities as input and returns a `Discretelength(logP)` distribution with those log-probabilities.
- $\mathbb{P}$`(p::Discrete)` that return the actual probabilities of the stored log-probabilities.
- `*`: Should use operator overloading of `Base.:*` (which is the default multiplication operator) to multiply two discrete distributions. Note that the multiplication of two discrete distributions is equivalent to the addition of their log probabilities. Also, use the parametric type system to ensure that only discrete distributions over the same outcomes can be multiplied.
- `/`: Should use operator overloading of `Base.:/` to represent the division of two discrete distributions. Note that the division of two discrete distributions is equivalent to the subtraction of their log probabilities. Also, use the parametric type system to ensure that only discrete distributions over the same outcomes can be multiplied.

Further examples of how to use the corresponding functions are given in the test files as well as the comments within the code.

The file `gaussian.jl` should implement most of the operations in the same way as they have been done for the discrete distribution.

- `*`: Should use operator overloading of `Base.:*` (which is the default multiplication operator) to multiply two 1D-Gaussian distributions.
- `/`: Should use operator overloading of `Base.:/` to represent the division of two 1D-Gaussian distributions.
- Inner Constructor that returns a Gaussian defined by precision mean τ and precision ρ(the natural parameterization)
- `Gaussian1DFromMeanVariance(m, s2)` that returns a Gaussian defined by mean `m` and variance `s2`.
- `mean(g)` that returns the mean of the 1D-Gaussian `g`
- `variance(g)` that returns the variance of the 1D-Gaussian `g`
- `absdiff(g1,g2)` that returns the absolute difference of two 1D-Gaussians `g1` and `g2` in terms of the maximum of $|\tau_1 - \tau_2|$ and $\sqrt{|\rho_1 - \rho_2|}$
- `logNormProduct(g1,g2)` that returns the logarithm of the normalization constant of multiplying `g1` and `g2`
- `logNormRatio(g1,g2)` that returns the logarithm of the normalization constant of dividing `g1` by `g2`

Part 2: Type-Safe and Generic Distribution Bag Julia Module (2 Points)

- Relevant source file: `distributionbag.jl`
- Test files: `test*_distributionbag.jl`

Later in the lecture, we will need to handle collections of distributions. To simplify this, we would like to implement a parametric type `DistributionBag{T}` which is a subtype of `AbstractArray{T, 1}` that can store an arbitrary length array of distributions of type `T`. The implementation already includes this parameterization in some methods.

It should provide those helper functions:

- Internal Constructor: Initializes the array with an empty array and stores the uniform distribution. Is parametrized by `T` which defines the datatype of the array. Please note, that the internal constructor (defined in the struct itself) uses {T} as an explicit type parameter, while the outer constructor receives `T` from its parameters using the `where`-clause.
- Outer Constructor: Accepts a distribution of type `T` as a parameter which serves as a template for the uniform distribution. Initializes the two variables with the uniform distribution being set and the actual bag being initialized as an Array of Type `T`. Passes `T` to the internal constructor.
- `add!`: To add a new distribution of the same type `T` to the bag. The new distribution should be appended to the internal array and the new index should be returned.
- `reset!`: Should replace all stored distributions with the uniform distribution stored in the bag. This means every position of the array is assigned with the uniform distribution stored in uniform.

It should also implement all functions necessary to behave like an array and for the bag to be used in for-loops, which means the following functions:

- `getindex`: Takes an integer `n` and returns the distribution at index `n`.
- `setindex!`: Receives an integer `n` and an instance of `T` and sets the distribution at index `n` to this new value.
- `firstindex`: Returns the integer value of the first valid index. This value is static, given it is the first valid position in the internal array.
- `lastindex`: Returns the value of the last valid index. This value has to be determined based on the size of the internal array.
- `size`: The number of items in the bag. Highly related to `lastindex`, and has to return the number as an integer stored in a tuple `(n,)`.
- `iterate`: Should implement the iterator as described in the second lecture.
- `eltype`: Should return the type `T` of the distributions in the bag (with `T` being a parameter available through the where clause in the method body).

Part 3: Equivalence of Scale-Location and Natural Parameterization of 1D Gaussians (1 Point)

- Relevant files: `gaussians.pdf`

Consider the following two representations of the density of the Normal Distribution, see Unit 2 (slide 10):

(1) Scale-Location Parameters, $x \in \mathbb{R}$, $\mu \in \mathbb{R}$, $\sigma^2 > 0$,

$$N(x; \mu, \sigma^2) = p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot \exp\left(-\frac{1}{2} \cdot \frac{(x-\mu)^2}{\sigma^2}\right)$$

(2) Natural Parameters, $x \in \mathbb{R}$, $\tau \in \mathbb{R}$, $\rho > 0$,

$$G(x; \tau, \rho) = p(x) = \sqrt{\frac{\rho}{2\pi}} \cdot \exp\left(-\frac{\tau^2}{2\rho}\right) \cdot \exp\left(\tau \cdot x - \rho \cdot \frac{x^2}{2}\right)$$

We want to understand the relation between both representations and how they can be transformed into each other. Further, we want to prove the important Multiplication Theorem, see Unit 2 (slide 11), which allows to multiply two Gaussians via simple additions of their Natural Parameters. Solve the following tasks:

(a) Show that $N(x; \mu, \sigma^2) = G\left(x; \frac{\mu}{\sigma^2}, \frac{1}{\sigma^2}\right)$. Start with $G\left(x; \frac{\mu}{\sigma^2}, \frac{1}{\sigma^2}\right)$ and use $\tau = \frac{\mu}{\sigma^2}, \rho = \frac{1}{\sigma^2}$.

(b) Show that $G(x; \tau, \rho) = N\left(x; \frac{\tau}{\rho}, \frac{1}{\rho}\right)$. Start with $N\left(x; \frac{\tau}{\rho}, \frac{1}{\rho}\right)$ and use $\mu = \frac{\tau}{\rho}, \sigma^2 = \frac{1}{\rho}$.

(c) Write down the formulas for the densities (i) $G(x; \tau_1, \rho_1)$, (ii) $G(x; \tau_2, \rho_2)$, (iii) $G(x; \tau_1 + \tau_2, \rho_1 + \rho_2)$, and the constant (iv) $N\left(\underbrace{\frac{\tau_1}{\rho_1}}_{\mu_1}; \underbrace{\frac{\tau_2}{\rho_2}}_{\mu_2}, \underbrace{\frac{1}{\rho_1}}_{\sigma_1^2} + \underbrace{\frac{1}{\rho_2}}_{\sigma_2^2}\right)$, which is based on the Normal density but does not depend on x.

(d) Start with the assertion $G(x; \tau_1, \rho_1) \cdot G(x; \tau_2, \rho_2) = G(x; \tau_1 + \tau_2, \rho_1 + \rho_2) \cdot N\left(\frac{\tau_1}{\rho_1}; \frac{\tau_2}{\rho_2}, \frac{1}{\rho_1} + \frac{1}{\rho_2}\right)$, fill in the formulas prepared in (c), and try to subsequently simplify the equation until the left-hand side and the right-hand side are visibly equal and the proof is complete. It is a good chance to practice various calculation rules and to appreciate that this theorem – thanks to your verification – will save us a lot of time in the next weeks. Good luck!