Lab 04

Regression with Bayesian Methods

2023-09-15

Table of contents

# 1. Problem statement

Use linear regression to relate the mean annual runoff of several streams (runoff) with nine other variables:

1. the precipitation falling at the gage (precip),
2. the drainage area of the basin (area),
3. the average slope of the basin (slope)
4. the length of the drainage basin (length)
5. the perimeter of the basin (perim)
6. the diameter of the largest circle which could be inscribed within the drainage basin (diameter)
7. the shape factor of the basin (shape\_factor)
8. the stream frequency – the ratio of the number of streams in the basin to the basin area (stream\_freq)
9. the relief ratio for the basin (relief\_ratio)

|  |
| --- |
| Credit |
| This is a light modification of Example 11.3 from  Helsel, D. R., Hirsch, R. M., Ryberg, K. R., Archfield, S. A., & Gilroy, E. J. (2020). Statistical methods in water resources. Techniques and Methods. U.S. Geological Survey. https://doi.org/10.3133/tm4A3 |

# 2. Setup

First, we load the packages that we will use

using CSV  
using DataFrames  
using Distributions  
using DynamicHMC  
using LaTeXStrings  
using LinearAlgebra: I  
using Plots  
using StatsPlots  
using Turing

## 2.1 Data

Next, we need to load in the data.

runoff\_df = CSV.read("data/runoff.csv", DataFrame)  
display(size(runoff\_df))  
first(runoff\_df, 3)

(13, 10)

# 3. Model explanation and exploration

## 3.1 Model specification

We are going to fit models of the form

We only have 13 observations of 10 variables, so we’ll need to be careful with our data. Before diving into the model, let’s look at the data.

## 3.2 Pairwise correlations

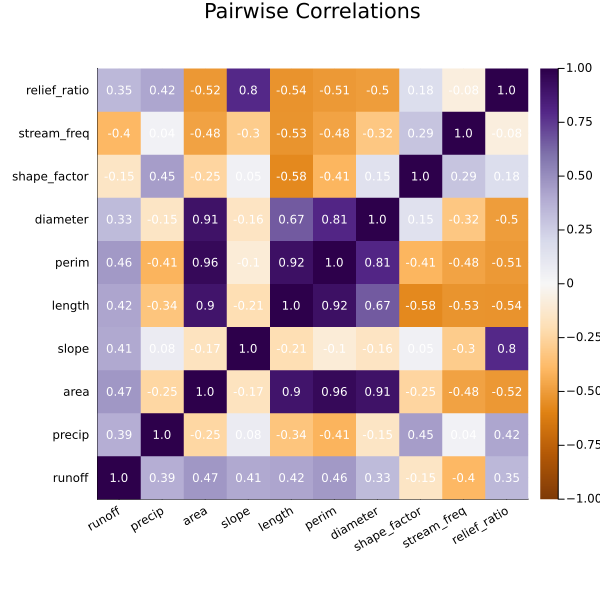
One simple way to look at the data is to look at the pairwise correlations between the variables. The following function plot\_correlation will plot a heatmap of the pairwise correlations.

function plot\_correlation(df::AbstractDataFrame)  
 varnames = names(df)  
 correlations = cor(Matrix(df))  
 P = size(correlations, 1)  
  
 p = heatmap(  
 varnames, # x labels  
 varnames, # y labels  
 correlations; # variable  
 aspect\_ratio=1, # 1:1 aspect ratio  
 xrotation=30, # rotate x ticks  
 size=(600, 600), # specify the size of the plot  
 title="Pairwise Correlations",  
 clims=(-1, 1),  
 cmap=:PuOr, # select a diverging color map  
 )  
  
 # add the exact text  
 ann = [  
 (i - 0.5, j - 0.5, text(round(correlations[i, j]; digits=2), 8, :white, :center))  
 for i in 1:P for j in 1:P  
 ]  
 annotate!(p, ann; linecolor=:white)  
 return p  
end

Line 3

cor expects a Matrix input and will throw an error if we give it a DataFrame. Matrix(df) just extracts numeric values for our DataFrame. This won’t work well if we have non-numeric columns, but we don’t here.

plot\_correlation(runoff\_df)



|  |
| --- |
| Task |
| Describe what you learn from this plot in the specific context of modeling the relationship between runoff and the other variables. What can you not learn from this plot? |

# 4. Linear regression

We can now fit a linear regression model to a single variable. You can pick whichever one you like; describe **why** you picked it.

We need priors on the intercept, slope, and standard deviation. Making some plausible choices, our full model is:

where is the half-normal distribution, i.e. a Normal distribution truncated at zero.

|  |
| --- |
| Task |
| Update the above model to reflect the priors you end up choosing. You are welcome to use other distributions or parameter values. You will need to modify the LaTeX math, but only very minor edits are required. |

## 4.1 Define model

We can implement this in Turing as follows:

#| output: false  
@model function linear\_reg(X::AbstractVector, y::AbstractVector)  
  
 # First, define all parameters  
 # they need priors so we use a ~  
 α # the intercept  
 β # the slope  
 σ # the standard deviation  
  
 # We can calculate intermediate quantities in our function  
 # Turing knows this isn't a parameter because we use = not ~  
 μ = α .+ β .\* X  
  
 # likelihood model  
 y ...  
end

::: {.callout-important} ## Task

Implement the model above so that it is consistent with the mathematical model written above. You will iterate between this and the next step a few times. You only need to keep your final model, but briefly describe what you did **and why**. :::## Prior predictive check

Before we fit the model, we run a prior predictive check. This lets us check whether the prior assumptions we have encoded into the model are consisten with our beliefs. First we sample from the prior:

y = vec(runoff\_df[!, :runoff])  
X = vec(runoff\_df[!, :YOUR\_VAR\_HERE]) # which variable did you choose?  
ppc\_linear = let  
 model = linear\_reg(X, y) # call the model  
 sampler = Prior() # we want to sample from the prior  
 nsamples = 10\_000  
 sample(model, sampler, nsamples; drop\_warmup=true)  
end

We can use the lm\_predict function to generate predictions from the prior.

function lm\_predict(chn::Turing.MCMCChains.Chains, X::AbstractVecOrMat, N::Int)  
 p = get\_params(chn)  
 α = vec(p[:α])  
 β = if isa(p[:β], Tuple)  
 β\_matrix = hcat(vec.(p[:β])...)  
 [β\_matrix[i, :] for i in 1:size(β\_matrix, 1)]  
 else  
 vec(p[:β])  
 end  
 idx = rand(1:length(α), N)  
 ŷ = [  
 α[i] .+ X \* β[i] for i in idx  
 ]  
 if N == 1  
 ŷ = ŷ[1]  
 end  
 return ŷ  
end

Line 2

This function extracts the parameters from the chain

Line 3

If we use multiple chains, then p[:α] is a Matrix. We convert it to a vector so that it’s “flat”.

Line 4

If we have multiple predictors (multivariate linear regression) then p[:β] is a tuple of vectors. We convert it into a vector of vectors so that β[i] is a vector of coefficients for the th sample.

Line 10

We randomly select N samples from the posterior.

Line 11

We calculate the predictions for each of the N samples.

Line 14

If N is 1, then we return a vector, not a vector of vectors.

We leverage this function to generate predictions from the prior and plot them.

X\_pred = ... # specify the range of values for  
p1 = plot(xlabel="Precip", ylabel="Runoff", title="Prior predictive check")  
for i in 1:250  
 y\_pred = lm\_predict(ppc\_linear, X\_pred, 1)  
 plot!(p1, X\_pred, y\_pred; color=:black, alpha=0.1, label=false)  
end  
p1

|  |
| --- |
| Task |
| Fill in the missing lines in the above blocks and run. |

## 4.2 Inference

Once we have a prior that we’re happy with, we can use our model to generate samples from the posterior distribution. We draw 4 chains of 5000 samples. If you get a weird error here, it may be because of threading. Try the mapreduce based approach [here](https://turinglang.org/v0.29/docs/using-turing/guide/#sampling-multiple-chains) rather than waste time debugging.

chn\_linear = let  
 model = ... # FILL IN  
 sampler = externalsampler(DynamicHMC.NUTS())  
 n\_per\_chain = 5000  
 nchains = 4  
 sample(model, sampler, MCMCThreads(), n\_per\_chain, nchains; drop\_warmup=true)  
end  
summarystats(chn\_linear)

Based on values close to 1, we have some evidence that our chains have converged. We can also check the trace plots:

... # Fill in to generate the trace plot

|  |
| --- |
| Task |
| Fill in the missing lines in the above blocks. Describe what you learn from the trace plot. |

## 4.3 Posterior predictive checks

We can again plot our predictions, this time using the posterior samples. We add the data to the plot as well.

p2 = plot(xlabel="Precip", ylabel="Runoff", title="Posterior") # blank plot  
pred\_linear = lm\_predict(chn\_linear, X\_pred, 500) # generate predictions  
for y\_pred in pred\_linear  
 plot!(p2, X\_pred, y\_pred; color=:black, alpha=0.1, label=false) # add each prediction to a plot  
end  
# add the observations with scatter!  
p2

|  |
| --- |
| Task |
| Add the observations to the plot above with scatter! |

# 5. Multiple predictors

By using a model with only one predictor, we may be throwing away usful information from other variables.

#| output: false  
@model function linear\_reg(X::Matrix, y::AbstractVector)  
  
 N, P = size(X) # number of observations, number of predictors  
  
 # priors  
 α ~ Normal(0, 5) # intercept  
 β ~ MvNormal(zeros(P), 5.0 \* I)  
 σ ~ LogNormal(2, 0.5)  
  
 # intermediate quantity  
 # deterministic transformation  
 μ = α .+ X \* β  
  
 # likelihood  
 y .~ Normal.(μ, σ)  
  
 return μ  
end

Line 8

β is a vector, so we use a multivariate normal distribution. Here, we assume that the predictors are uncorrelated, so we use a diagonal covariance matrix.

Line 9

We use a log-normal distribution for because it is a positive quantity. This is an alternative to a truncated Normal.

|  |
| --- |
| Task |
| Uncomment the above model and then modify it to use better priors, iterating with the next step |

## 5.1 Prior predictive check

X = Matrix(runoff\_df[:, Not(:runoff)]) # all columns except runoff  
y = vec(runoff\_df[!, :runoff]) # runoff   
  
ppc\_multi = let  
 ... # FILL IN  
end

How should we visualize the prior predictive check? A simple way is to visualize the implied distribution of given observed :

histogram(  
 vcat(lm\_predict(ppc\_multi, X, 500)...),  
 xlabel=L"$y$",  
 ylabel=L"Implied $p(y)$",  
 label=false,  
 normalize=:pdf,  
 yticks=false,  
 title="Prior predictive check",  
)

|  |
| --- |
| Task |
| Modify the above code to draw samples from the prior and save as ppc\_multi. |

## 5.2 Inference

Now that our priors seem plausible (we could a;ways do better, but that’s not the point of this lab), we can fit the model.julia chn\_multi = let ... end # summary stats

|  |
| --- |
| Task |
| Draw the samples from the posterior and save as chn\_multi. Display the summary statistics. What do you learn? |

## 5.3 Posterior predictive check

How should we visualize these inferences? It’s hard to plot a single vs when we have multiple . There are many possible plots, but a good one is to plot the predicted vs actual values of . To make the plot more readable, we’ll add some jitter to the data. Here’s a function to do that:

function jitter(x, y; ϵx=0.1, ϵy=0.1)  
 x\_jit = x .+ rand(Normal(0, ϵx), size(x))  
 y\_jit = y .+ rand(Normal(0, ϵy), size(y))  
 return x\_jit, y\_jit  
end

Now we can create our scatter plot.

p4 = plot(; xlabel="Predicted Runoff", ylabel="Actual Runoff", aspect\_ratio=1) # blank plot  
for i in 1:500 # loop through  
 pred = lm\_predict(chn\_multi, X, 1)  
 xplt, yplt = jitter(pred, y; ϵx=0.1, ϵy=0.1)  
 scatter!(xplt, yplt; color=:black, alpha=0.5, label=false, markersize=0.5)  
end  
Plots.abline!(p4, 1, 0; color=:red, label="1:1 line")  
p4

|  |
| --- |
| Task |
| The above block shouldn’t need to be modified. Add the curly brackets when you are ready. What do you learn from this plot? |

# 6. Submission

As you did for previous labs, you will submit your lab by pushing it to GitHub. In addition, submit the DOCX or PDF file (as for lab 2) to Canvas.