Probabilistic Machine Learning with Julia

Lecture 3 | Probabilistic Programs

Amirabbas Asadi

amir.asadi78@sharif.edu

2025

About the Course

Probabilistic Machine Learning with Julia

Exploring Julia ecosystem for probabilistic modeling and inference focused on probabilistic programming

Course page : `bayesianrl.github.io/pml2025`

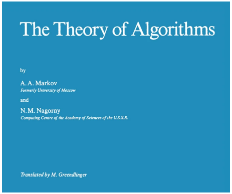
Lecturer: Amirabbas Asadi (`amir.asadi78@sharif.edu`)

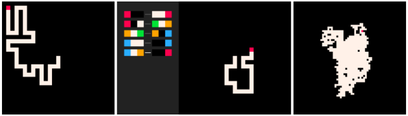
Course Outline

- Intoroduction to Julia
- Introduction to Probabilistic Programming
- Probabilistic Modeling
  - Parametric Probabilistic Models
    - Basic Latent Variable Models [`Turing.jl`]
    - Bayesian Deep Learning [`Turing.jl`, `Lux.jl`]
    - Bayesian Differential Equations [`DifferentialEquations.jl`]
  - Nonparametric Probabilistic Models [`GaussianProcesses.jl`]
- Inference
  - Markov Chain Monte Carlo [`AdvancedMH.jl`, `AdvancedHMC.jl`]
  - Parametric Variational Inference [`AdvancedVI.jl`]
  - Nonparametric Variational Inference [`NonparametricVI.jl`]
  - Reactive Message Passing [`RxInfer.jl`]
- Applications
  - Probabilistic Time Series Modeling
  - Bayesian Optimization
  - Bayesian Reinforcement Learning
  - Active Inference

Probabilistic Programs

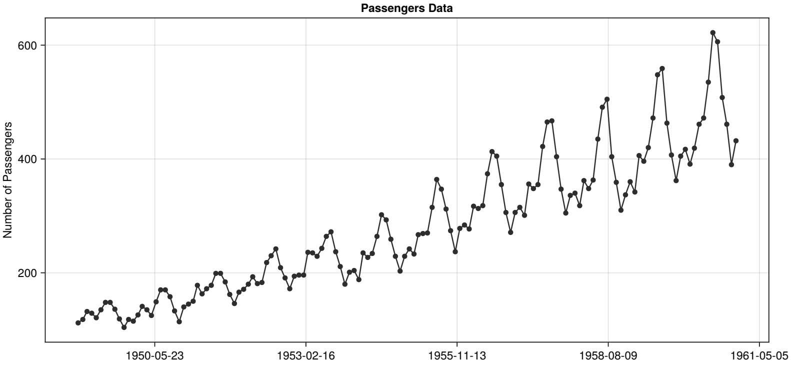
The meeting of grammars and stochastic processes





https://yuu6883.github.io/MarkovJuniorWeb/

A Simple Time Series Example



A Stupid Model

Let's denote the number of passengers at time τ by Y_τ

$$Y_1, Y_2, Y_3, \cdots Y_T$$

Our goal is to design a probabilistic model:

$$p(Y_1 = y_1, Y_2 = y_2, Y_3 = y_3, \cdots, Y_T = y_T)$$

A Stupid Model

$$Y_1, Y_2, Y_3, \cdots, Y_T$$

In our very first example, we assume Y_i are i.i.d !!!

$$p(y_1, y_2, \cdots, y_T) = p(y_1)p(y_2)\cdots p(y_T)$$

We further assume they follow a Poisson distribution:

$$p(Y_\tau = k; \lambda) = \frac{\lambda^k e^{-\lambda}}{k!}$$

A Stupid Model

The two main functions in any probabilistic programming language are:

- **sample** : Generating a random variable from a distribution
- **observe** : Conditioning on observed data

A Stupid Model

The two main functions in any probabilistic programming language are:

- **sample** : Generating a random variable from a distribution
- **observe** : Conditioning on observed data

For example this program only includes sampling and defines a joint density

```
@model function p()
    λ ~ Exponential(300.0)
    y ~ Poisson(λ)
end
```

$$p(\lambda, y)$$

A Stupid Model

$$p(\lambda, y)$$

So far we haven't see anything more than simulation. The main feature of probabilistic programs is conditioning the model on observed data.

A Stupid Model

$$p(\lambda, y)$$

So far we haven't see anything more than simulation. The main feature of probabilistic programs is conditioning the model on observed data.

$$\overbrace{p(\lambda|y)}^{\text{posterior}} = \frac{\overbrace{p(y|\lambda)}^{\text{likelihood}}\ \overbrace{p(\lambda)}^{\text{prior}}}{\underbrace{p(y)}_{\text{evidence}}}$$

A Stupid Model

$$p(\lambda, y)$$

So far we haven't see anything more than simulation. The main feature of probabilistic programs is conditioning the model on observed data.

$$\overbrace{p(\lambda|y)}^{\text{posterior}} = \frac{\overbrace{p(y|\lambda)}^{\text{likelihood}}\ \overbrace{p(\lambda)}^{\text{prior}}}{\underbrace{p(y)}_{\text{evidence}}}$$

$$p(\lambda|y) \propto p(y|\lambda)p(\lambda)$$

A Stupid Model

Observing a variable simply means conditioning. We usually pass observations as arguments to the model:

```
@model function p(y_obs)
    λ ~ Exponential(300.0)
    y_obs ~ Poisson(λ)
end
```

$$p(\lambda|y_{\text{obs}})$$

Unobserved variables like λ here are called latent variable. We need to specify a prior for each latent variable like `Exponential` in the above example.

A Stupid Model

We use `Turing.jl` to define a probabilistic program:

```julia
using Turing
```

To create a probabilistic program with Turing wrap a function in `@model` macro

```julia
@model function naive_model(y)




end
```

Here `y` is a vector of observed data.

A Stupid Model

We use `Turing.jl` to define a probabilistic program:

```julia
using Turing
```

We use operator ~ to sample a variable:

```julia
@model function naive_model(y)
    λ ~ Exponential(300.0)




end
```

`Exponential` is defined in `Distributions.jl` package.

A Stupid Model

We use `Turing.jl` to define a probabilistic program:

```julia
using Turing
```

We can use usual control flows (`while`, `for`, `if`, ...) inside a probabilistic program:

```julia
@model function naive_model(y)
    λ ~ Exponential(300.0)

    for τ ∈ eachindex(y)
        y[τ] ~ Poisson(λ)
    end
end
```

Note that we have used ~ for both sampling latent variables and observing data.

A Stupid Model

```julia
@model function naive_model(y)
    λ ~ Exponential(300.0)

    for τ ∈ eachindex(y)
        y[τ] ~ Poisson(λ)
    end
end
```

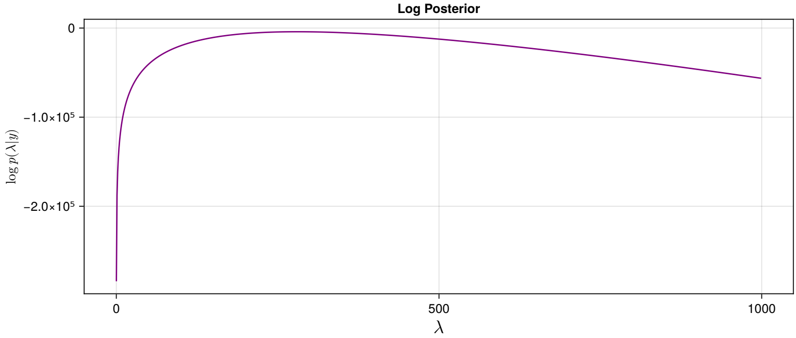
Once the model is defined, we can generate samples from prior:

```julia
rand(naive_model(y))
# output
(λ = 205.46898514251205,)
```

A Stupid Model

We can also evaluate the unnormalized posterior:

```
logjoint(naive_model(y), (λ=λ₀))
```



A Stupid Model

We can compute **Maximum a Posterior** estimation using `Optim.jl` package.

$$\lambda_{\text{MAP}} = \text{argmax}_{\lambda}\, p(\lambda|y)$$

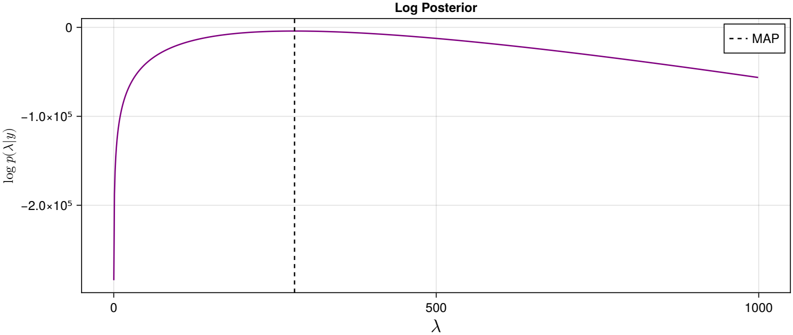
```
using Optim
optimize(naive_model(y), MAP(), LBFGS())
```

A Stupid Model

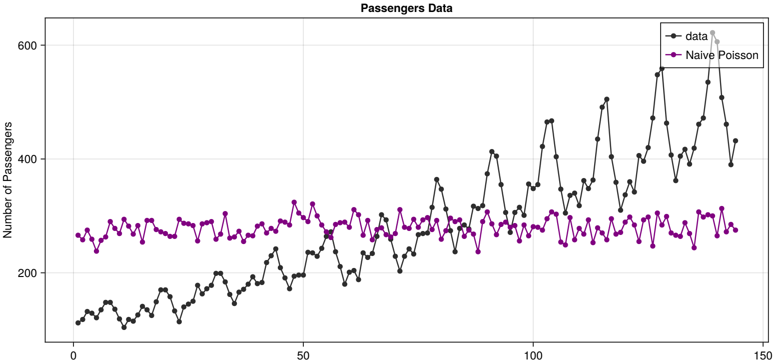
We can compute **Maximum a Posterior** estimation using `Optim.jl` package.

$$\lambda_{\text{MAP}} = \operatorname{argmax}_{\lambda} p(\lambda|y)$$

```
using Optim
optimize(naive_model(y), MAP(), LBFGS())
```



A Stupid Model



Poisson Regression

To make the model realistic, we can assume λ changes over time.

$$Y_\tau \sim \text{Poisson}(\lambda_\tau)$$

This model is known as **Poisson Regression** where $\log \lambda_\tau$ changes linearly over time.

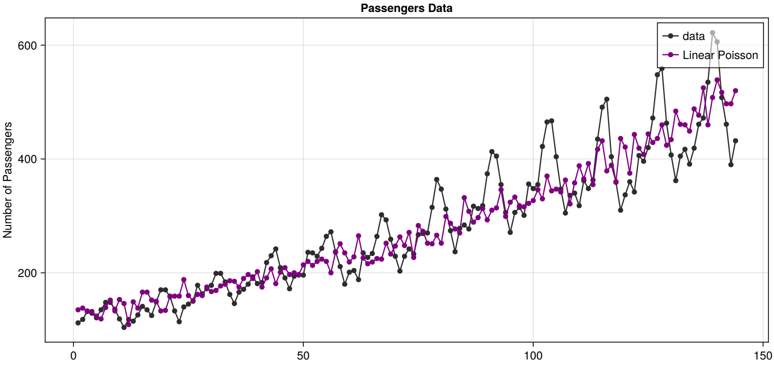
$$\log \lambda_\tau = \alpha\tau + \beta$$

Poisson Regression

```julia
@model function poisson_linear_model(y)
    α ~ Normal(0.0, 2.0)
    β ~ Normal(0.0, 2.0)

    for τ in 1:length(y)
        log_λ = α*τ + β
        y[τ] ~ Poisson(exp(log_λ))
    end
end
```

Poisson Regression



Poisson Regression

$$\log \lambda_\tau = \alpha\tau + \beta$$

One can add a nonlinear component to handle seasonality

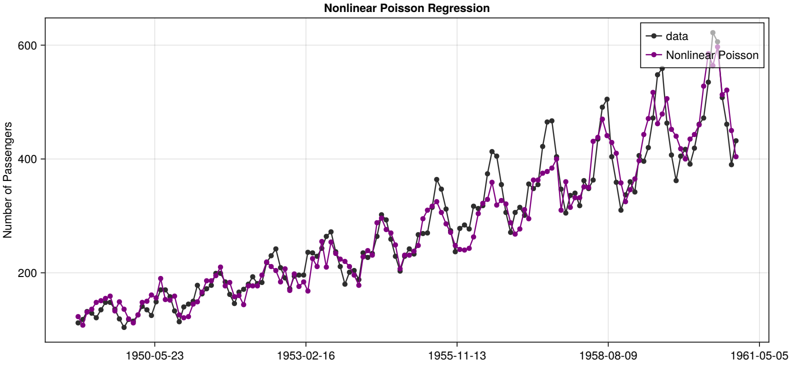
$$\log \lambda_\tau = \alpha\tau + \nu \cos(\pi\omega(\tau - \phi)) + \beta$$

Poisson Regression

```julia
@model function poisson_nonlinear_model(y)
    α ~ Normal(0.0, 2.0)
    β ~ Normal(0.0, 2.0)
    ν ~ Normal(0.0, 2.0)
    ϕ ~ Normal(0.0, 2.0)
    ω ~ Normal(0.0, 2.0)

    for τ in 1:length(y)
        log_λ = α*τ + ν*cos(π*ω*(τ-ϕ)) + β
        y[τ] ~ Poisson(exp(log_λ))
    end
end
```

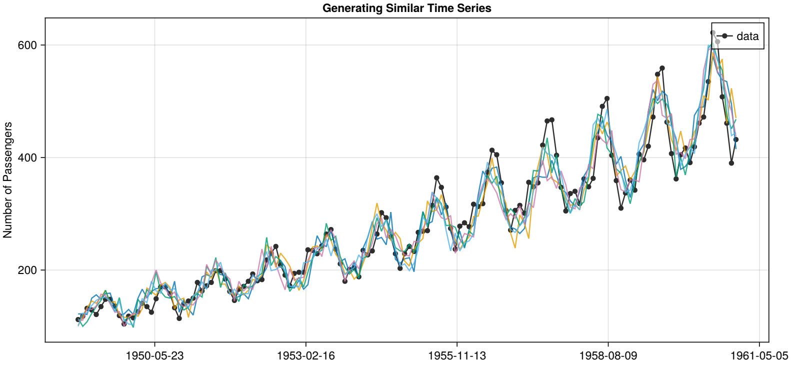
Poisson Regression



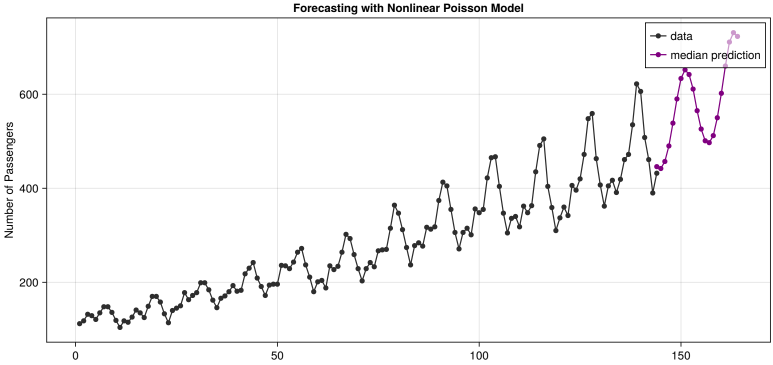
Examples

What else can we do with this probabilistic model?

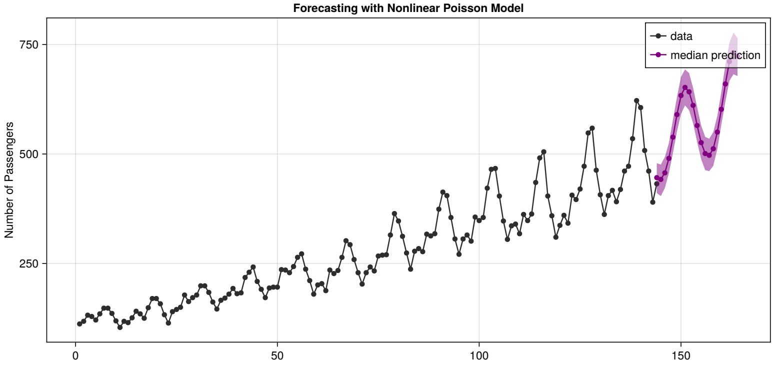
Example: Data Generation



Example: Forecasting

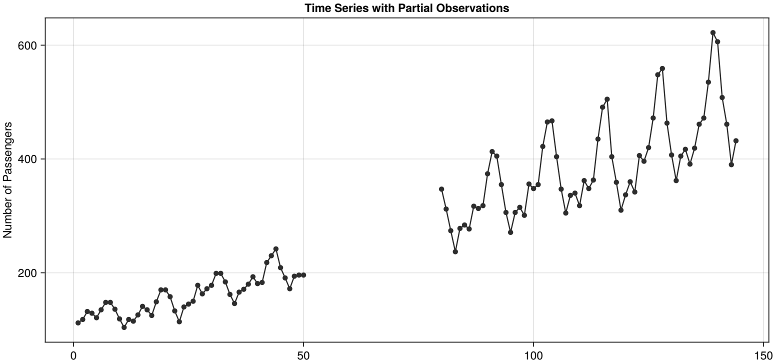


Example: Probabilistic Forecasting

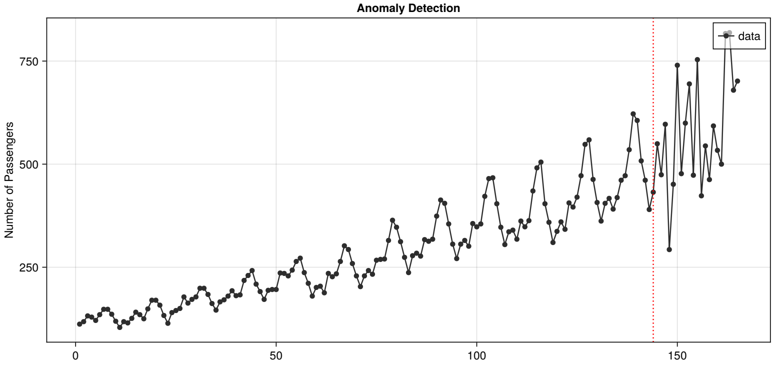


Note that the predicted interval does not look great, Why?

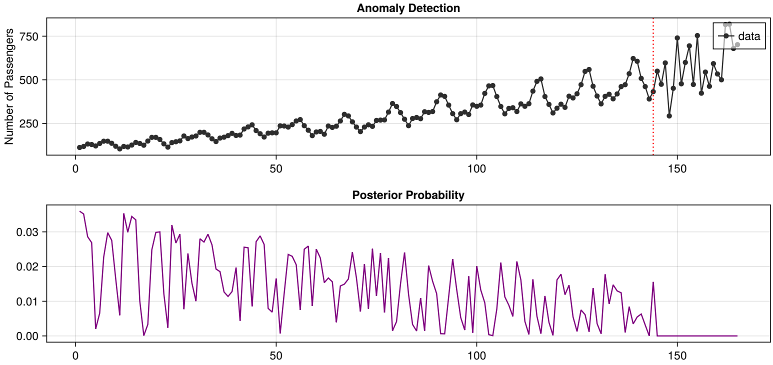
Example: Interpolation



Example: Anomaly Detection



Example: Anomaly Detection



Classification of Probabilistic Programming Languages

All of the probabilistic programs which we have seen so far are called **First-Order Probabilistic Programs** because no **stochastic branching** happens and runtime flow is deterministic.

Classification of Probabilistic Programming Languages

All of the probabilistic programs which we have seen so far are called **First-Order Probabilistic Programs** because no **stochastic branching** happens and runtime flow is deterministic.

```
@model function program_with_stochastic_branch()
    N ~ Poisson(10)
    S = 0
    for i in 1:N
        S += 1
    end
end
```

The representation power of First-Order Probabilistic Programming Languages (FOPLs) is equal to Probabilistic Graphical Models'!

Next Session

In the next session, we will discuss alternative models for our time series problem:

- Bayesian Differential Equation
- State-Space Models
- Bayesian Neural Networks
- Nonparametric Models