Universidad Veracruzana

Inteligencia Artificial

Métodos Probabilísticos

Actividad 6

Leonardo Flores Torres

31 de octubre de 2022

1. Ajuste de distribuciones de probabilidad a datos de entrenamiento y la distribución predictiva para una distribución Gaussiana unidimensional. Defina $N > 10$ puntos aleatorios extraídos de una distribución gaussiana $\mathrm{G}_x(\mu_0, \sigma_0)$, con $\mu_0 = 5$ y $\sigma_0 = 1.5$, $\mathrm{RandomG}_x(\sigma_0, \mu_0)$.

   Solución:

   Una distribución normal univariada $\mathrm{Norm}_x[\sigma, \mu]$ está dada por

   $$\mathrm{Norm}_x[\sigma, \mu] = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right], \tag{1}$$

   donde μ es la media y σ es la desviación estándar. Se generó una distribución normal univariada centrada en $\mu_0 = 5$ con desviación estándar $\sigma_0 = 1.5$, como se muestra en la figura 1, con dominio $D_0 : \{x \mid x \in [\mu_0 - 4\sigma_0,\ \mu_0 + 4\sigma_0]\}$. De esta distribución normal se realizó un muestreo para extraer un número $N = 30$ de datos los cuales son el conjunto de entrenamiento $\{x_i\}_{i=1}^{30}$ usado en el resto de los incisos a responder en esta actividad.

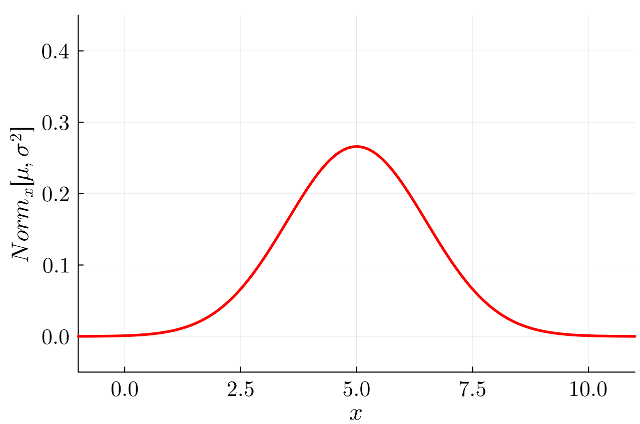


Figura 1: Distribución gaussiana de probabilidad usada para extrar los datos de entrenamiento.

   Solamente para clarificar, los parámetros que maximizan la verosimilitud $\{\hat{\mu}, \hat{\sigma}\}$ se encuentran al computar las derivadas e igualando a cero para encontrar un máximo, respecto a cada uno de los parámetros $\{\mu, \sigma\}$, del producto de las verosimilitudes individuales para cada uno de los puntos del conjunto muestreal $\{x_i\}_{i=1}^{N}$. Como se está usando una distribución normal univariada se tiene que la verosimilitud es igual a

$$\Pr(x_{1\ldots N}|\mu,\sigma^2) = \prod_{i=1}^{N} \mathrm{Norm}_{x_i}\left[\mu,\sigma^2\right] . \tag{2}$$

Después de encontrar las derivadas respecto a cada uno de los parámetros e igualando a cero se obtiene que los parámetros que maximizan la verosimilitud son

$$\hat{\mu} = \frac{1}{N}\sum_{i=1}^{N} x_i \ , \tag{3}$$

$$\hat{\sigma}^2 = \frac{1}{N}\sum_{i=1}^{N} \left(x_i - \hat{\mu}\right)^2 \ . \tag{4}$$

Para generar estos datos se definen los parámetros iniciales `μ0` y `σ0`, se define el dominio en `x0`, y el muestreo se computa en `xsample`. También se computan los valores de la gaussiana en el dominio, i.e., $\mathrm{G}_{x_0}(\mu_0,\sigma_0)$. Es importante mencionar que el dominio será el mismo en todas las demás distribuciones que se generen, esto quiere decir $D_i = D_0$ para $i = 1,2,3$ donde i denota el conjunto de condiciones $\{\mu_i,\ \sigma_i\}$ mencionadas en el segundo inciso de la actividad.

```julia
julia> using Revise, NormalDist; nd = NormalDist;
julia> begin
       # Parametros iniciales de la gaussiana
       μ0 = 5
       σ0 = 1.5
       # Dominio de la gaussiana original
       x0 = range(μ0 - 4 * σ0, μ0 + 4 * σ0, 2000)
       # Evaluacion de la gaussiana en el dominio G(x0)
       w0 = nd.normalDist(x0, μ0, σ0)
       # Numero de puntos para muestreo
       npts = 30
       # Muestreo de puntos
       xsample = nd.randomSample(x0, w0, npts)
       # Grafica de la gaussiana
       fig = nd.plotNormal(x0, w0)
       end
```

Nótese que en ningún momento se ha mencionado que es necesario restringir el conjunto de entrenamiento a valores únicos, por lo que está permitido tener puntos repetidos dentro del conjunto aleatorio de entrenamiento $\{x_i\}_i^{30}$.

2. Grafique 3 figuras como la lámina 16 del Prince, diapositiva 4, donde se muestren los N puntos, y los valores del likelihood para 3 Gaussianas considerando

   a) $\mu_1 = 3$, $\sigma_1 = 1$,

   b) $\mu_2 = 6$, $\sigma_2 = 1.6$,

   c) $\mu_3 = 5.1$, $\sigma_3 = 1.4$.

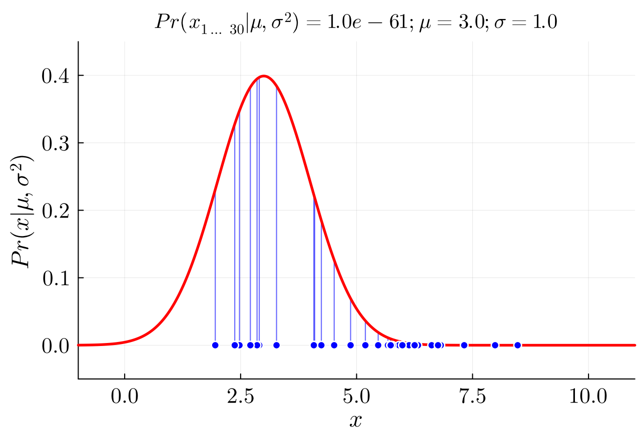
   Grafique las Gaussianas sobre los puntos de entrenamiento y su ordenada para cada Gaussiana, $\mathrm{Gauss}_x(\mu_i,\sigma_i)$.

   Solución:

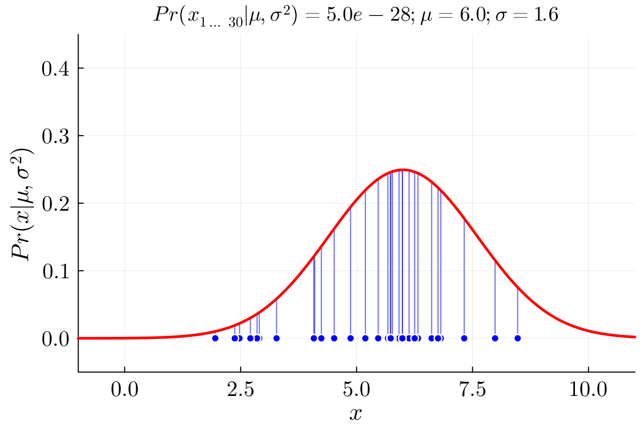
   Un ejemplo de como se computaron todas las distribuciones se muestra a continuación:

```
17 julia> begin
18        # Parametros para el i-esimo caso
19        μi = 6       # parametro μi
20        σi = 1.6     # parametro σi
21        # Evaluacion de la i-esima gaussiana con nuevos parametros sobre el conjunto de datos
          ↪  muestreados
22        wsample = nd.normalDist(xsample, μi, σi)
23        # Computo de la i-esima gaussiana con nuevos parametros
24        wi = nd.normalDist(x0, μi, σi)
25        # Likelihood de los datos xsample respecto a los i-esimos parametros
26        lh = nd.likelihood(xsample, μi, σi)
27        # Grafica de la i-esima gaussiana con los datos muestreados
28        fig = nd.plotGuess(xsample, wsample, x0, wi, lh, μi, σi; maxlh = false)
29        end
```

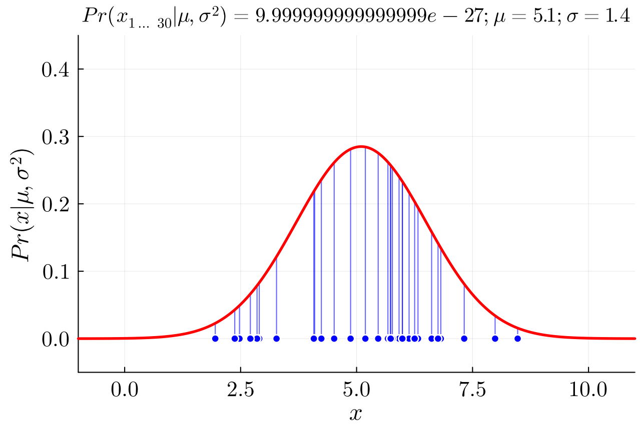
Las gaussianas obtenidas con sus respectivos parámetros se muestran en la figura 2, donde también se incluyó la gráfica para la gaussiana con los parámetros que maximizan la verosimilitud $\{\hat{\mu},\ \hat{\sigma}\}$. Además, cada gráfica incluye su verosimilitud dependiendo de sus parámetros $Pr(x_{1\ldots30}|\ \mu_i,\ \sigma_i^2)$. Para el caso de $\{\hat{\mu},\ \hat{\sigma}\}$ sólamente se realizaron las siguientes asignaciones `μi = nd.maxμ(xsample)` y `σi = nd.maxσ(xsample)` en las lineas 19 y 20, y se cambia el parámetro `maxlh = true` de la función donde se genera la gráfica para decir al programa que cambie el título de la gráfica.



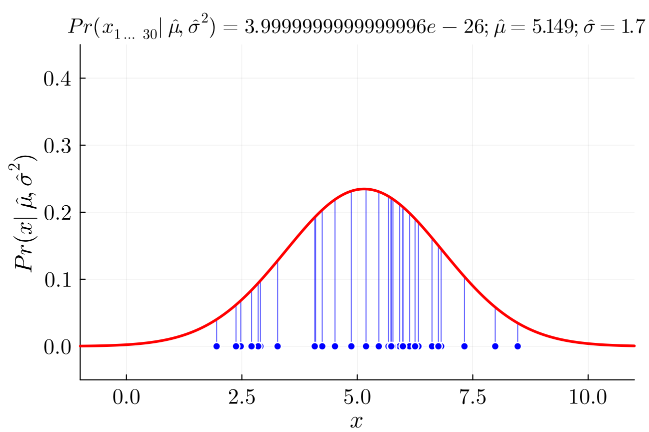
(a) Distribución $G_x(\mu_1, \sigma_1)$ para los parámetros $\mu_1 = 3$, $\sigma_1 = 1$.



(b) Distribución $G_x(\mu_2, \sigma_2)$ para los parámetros $\mu_2 = 6$, $\sigma_2 = 1.6$.



(c) Distribución $G_x(\mu_3, \sigma_3)$ para los parámetros $\mu_3 = 5.1$, $\sigma_3 = 1.4$.



(d) Distribución $G_x(\hat{\mu}, \hat{\sigma})$ para los parámetros que maximizan el likelihood, $\hat{\mu}$, $\hat{\sigma}$.

Figura 2: Gráficas de las gaussianas requeridas para el segundo punto de la actividad con sus respectivos parámetros.

Nótese que para el conjunto de puntos muestreados $\{x_i\}_{i=1}^{30}$ se obtiene un buen valor para la verosimilitud, con parámetros $\{\hat{\mu} = 5.149,\ \hat{\sigma} = 1.7\}$ aunque estos valores dependen en gran medida de lo puntos dentro del conjunto muestreal.

Por completez se agrega el ejemplo de como computar la distribución con los parámetros mostrados en la figura 2d:

```
julia> begin
       # Parametros para el caso de maximizacion
       μi = nd.maxμ(xsample)
       σi = nd.maxσ(xsample)
       # probabilidades asociadas al conjunto muestreal
       wsample = nd.normalDist(xsample, μi, σi)
       # Computo de la gaussiana con parametros maximizados
       wi = nd.normalDist(x0, μi, σi)
       # Likelihood de los datos xsample respecto a los parametros
       lh = nd.likelihood(xsample, μi, σi)
       # Grafica de la gaussiana
       fig = nd.plotGuess(xsample, wsample, x0, wi, lh, μi, σi; maxlh = true)
       end
```

3. Genere el mapa térmico de las probabilidades considerando $\mu \in [2.5, 6.5]$, y $\sigma \in [0, 2]$, dividiendo ambos intervalos en 10 partes. Coloque un marcador en el punto de máxima verosimilitud.
4. Calcule, usando el método de Maximum Likelihood, los parámetros $\hat{\mu}$ y $\hat{\sigma}^2$ que tanto coincide con el mapa térmico.

Solución:

Los incisos 3 y 4 se resolvieron de manera conjunta ya que al añadir un marcador en el mapa de calor que indique donde se encuentra el punto de máxima verosimilitud este se tuvo que haber computado de antemano. Me tomé la libertad de cambiar el número de divisiones a 150 en que se subdividen los intervalos para μ y σ. La instrucción escrita en `julia` que genera el mapa de calor se muestra a continuación:

```
julia> begin
       # Numero de particiones para los intervalos de μ y σ
       partitions = 150
       # Intervalos de μ y σ
       μint = (2.5, 6.5)
       σint = (0, 2)
       # Deltas de μ y σ
       dμ = (μint[2] - μint[1]) / partitions
       dσ = (σint[2] - σint[1]) / partitions
       # Dominios de μ y σ
       μrange = range(μint[1] + dμ/2, μint[2] - dμ/2, partitions)
       σrange = range(σint[1] + dσ/2, σint[2] - dσ/2, partitions)
       # Mallado para obtener un espacio bidimensional
       (μmesh, σmesh) = nd.ndgrid(μrange, σrange)
       # Valores del likelihood en el mallado
       lhmesh = nd.likelihood.(Ref(xsample), μmesh, σmesh)
       # Parametros que maximizan el likelihood y el likelihood maximo
       μmax = nd.maxμ(xsample)
       σmax = nd.maxσ(xsample)
```

```julia
    lhmax = nd.likelihood(xsample, μmax, σmax)
    # Grafica del mallado
    fig = nd.plotHeat([3, 6, 5.1], [1, 1.6, 1.4], μmax, σmax, μrange, σrange, lhmesh, npts)
    end
```

El mapa de calor se muestra en la figura 3. El punto de máxima verosimilitud con parámetros $\{\hat{\mu}, \hat{\sigma}\}$ está marcado con una estrella de color cyan, mientras que los demás puntos marcados con círculos color magenta corresponden a las verosimilitudes con sus conjuntos respectivos de parámetros $\{\mu_i, \sigma_i\}$. Los valores para las verosimilitudes de cada uno de los casos se puede observar en el título

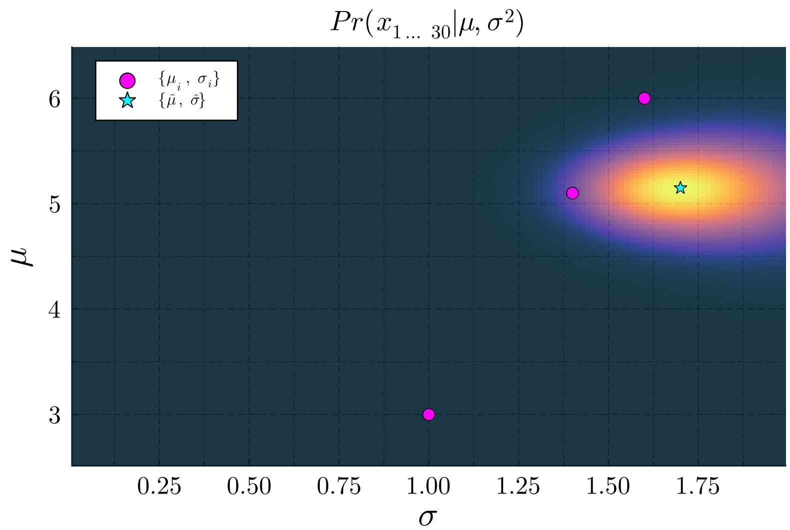


Figura 3: Mapa de calor incluyendo las verosimilitudes de sus respectivos parámetros.

de las figuras 2a a 2d con sus respectivos parámetros. El muestreo no es perfecto, mientras más puntos se obtengan para el muestreo mejor será la aproximación para la máxima verosimilitud. Aún así, al computar las distribuciones gaussianas $G_x(\mu_i, \sigma_i)$ surgia frecuentemente el caso en que la máxima verosimilitud era menor que la verosimilitud para el caso $\{\mu_3 = 5.1, \sigma_3 = 1.4\}$. Atribuyo este comportamiento como resultado del muestreo, pero la máxima verosimilitud siempre parecía encontrarse en el centro del elipsoide del mapa de calor.

Incluso se puede encontrar el valor del máximo likelihood correspondiente al mallado, se usa la función `argmax` sobre la variable donde se guardó el mallado la cual regresa índices y éstos se usan para encontrar los valores correspondientes de μ y σ en ese punto de la malla.

```julia
julia> μmeshi, σmeshi = argmax(lhmesh) |> x -> (μrange[x[1]], σrange[x[2]])
(5.153333333333333, 1.7)

julia> μmax, σmax
(5.149274637318659, 1.700282387563139)

julia> abs(μmeshi - μmax)
0.0040586960146740125

julia> abs(σmeshi - σmax)
0.00028238756313903046
```

Se puede observar que la diferencia para μ es de 0.4 porciento, mientras que la diferencia de σ es del 0.02 porciento. Para concluir con el trabajo solamente se quisiera recordar al lector que las instrucciones mencionadas sobre como se calcularon los distintos pasos de la actividad se hicieron dentro del `REPL` de `julia`, también se incluye el módulo desarrollado para esta actividad en el Apéndice.

Apéndice

```julia
module NormalDist

using StatsBase
using Plots
using LaTeXStrings
using LazyGrids # To generate lazy grids with low memory allocation


"""
    normalDist(x::Real, μ, σ)

Computes the corresponding probability of a single point `x` using a
normal distribution with a given mean `μ` and the squared root of the
variance `σ`.
"""
function normalDist(x::Real, μ, σ)
    exparg = - (x - μ)^2 / (2 * σ^2)
    den = sqrt(2 * π * σ^2)
    return exp(exparg) / den
end


"""
    normalDist(xpts::Array, μ, σ)

Computes the corresponding probability from a list of points `xpts` using a
normal distribution with a given mean `μ` and the squared root of the
variance `σ`.
"""
function normalDist(xpts::Union{Array, StepRangeLen}, μ, σ)
    return normalDist.(xpts, μ, σ)
end


"""
    likelihood(xpts, μ, σ)

Computes the likelihood of a given array of points `xpts` with respect of
a normal distribution with parameters `μ` and `σ`.
"""
function likelihood(xpts, μ, σ)
    npts = length(xpts)
    exparg = - sum((xpts .- μ).^2) / (2 * σ^2)
    den = (2 * π * σ^2)^(npts / 2)
```

```julia
    return exp(exparg) / den
end


"""
    maxμ(xpts)

Computes the mean value `μ` that maximizes the likelihood.
"""
function maxμ(xpts)
    npts = length(xpts)
    return sum(xpts) / npts
end


"""
    maxσ(xpts)

Computes the squared-root of the variance `σ` that maximizes
the likelihood.
"""
function maxσ(xpts)
    npts = length(xpts)
    return (sum((xpts .- maxμ(xpts)).^2) / npts) |> sqrt
end


"""
    randomSample(xpts, weights, npts)

Returns the desired number of points `npts` from a list of points `xpts`,
taking into account a list of their weights `weights` from any given
distribution.
"""
function randomSample(xpts, weights, npts)
    return sample(xpts, Weights(weights), npts)
end

function plotNormal(x, w)
    fig = plot(
        tickfont=(12, "Computer Modern"),
        xlim = (x[begin], x[end]),
        ylim = (-0.05, 0.45),
        dpi = 400,
        size = (600, 400),
    )

    plot!(x, w,
        linecolor = :red,
        lw = 2.5,
        label = "",
        xlabel = L"x",
        ylabel = L"Norm_{x}[\mu, \sigma^2]",
        xtickfontsize = 13,
```

```
        ytickfontsize = 13,
        xguidefontsize = 15,
        yguidefontsize = 15,
        titlefontsize = 13,
    )

    return fig
end

function plotGuess(xsample, wsample, x, w, lh, μi, σi; maxlh = false)
    npts = length(xsample)

    roundμ = round(μi, sigdigits=4)
    roundσ = round(σi, sigdigits=4)
    roundlh = round(lh, sigdigits=1)

    # theme(:ggplot2)
    fig = plot(
        tickfont=(12, "Computer Modern"),
        xlim = (x[begin], x[end]),
        ylim = (-0.05, 0.45),
        dpi = 400,
        size = (600, 400),
    )

    for i in 1:npts
        plot!(
            [xsample[i], xsample[i]],
            [0, wsample[i]],
            label = "",
            linecolor = :blue,
            linealpha = 0.6,
        )
    end

    # Just some formatting depending on if the maximum likelihood is
    # ploted or not
    μstr = maxlh == false ? "μ" : raw"\hat{μ}"
    σstr = maxlh == false ? "σ" : raw"\hat{σ}"
    title = raw"Pr(x_{1 \ldots" * "$(npts)" * raw"} |" * μstr * raw", " *
        σstr * raw"^2) = " * "$(roundlh)" * raw"; " * μstr * " = " *
        "$(roundμ); " * σstr * raw" = " * "$(roundσ)"
    ylabel = raw"Pr(x | " * "$(μstr)" * ", " * "$(σstr)" * raw"^2)"
    plot!(x, w,
        linecolor = :red,
        lw = 2.5,
        label = "",
        xlabel = L"x",
        ylabel = latexstring(ylabel),
        title = latexstring(title),
        xtickfontsize = 13,
        ytickfontsize = 13,
        xguidefontsize = 15,
        yguidefontsize = 15,
```

```
        titlefontsize = 13,
    )

    scatter!(xsample, zeros(npts),
        label = "",
        mc = :blue,     # marker color
        msc = :white,   # marker stroke color
        msw = 2,        # marker stroke width
    )

    return fig
end


function plotHeat(μi, σi, μmax, σmax, μrange, σrange, lhmesh, npts)
    title = raw"Pr(x_{1\ldots" * "$(npts)" * raw"} | μ, σ^2)"
    fig = plot(
        xlim = (σrange[begin], σrange[end]),
        ylim = (μrange[begin], μrange[end]),
        dpi = 400,
        size = (600, 400),
        xlabel = L"\sigma",
        ylabel = L"\mu",
        title = latexstring(title),
        xguidefontsize = 17,
        yguidefontsize = 17,
        tickfont = (12, "Computer Modern"),
        grid = :dash,
        gridlinewidth = 0.6,
        gridalpha = 0.95,
        minorgrid = :dash,
        minorgridlinewidth = 0.4,
        minorticks = 2,
        minorgridalpha = 0.95,
        legend = :topleft,
    )

    heatmap!(σrange, μrange, lhmesh,
        c = cgrad(:thermal, rev = false, alpha=0.9), # colormap
        colorbar = :none,
    )

    scatter!(σi, μi,
        label = L"\{μ_{i}\ ,\ σ_{i}\}",
        mc = :magenta,
        ms = 5,
        msw = 1,
        msc = :black,
    )

    scatter!([σmax], [μmax],
        label = L"\{\hat{μ}\ ,\ \hat{σ}\}",
        mc = :cyan,
        ms = 5,
```

```
        shape = :star5,
        msw = 1,
        msc = :black
    )

    return fig
end


end # module NormalDist
```

Referencias

[1] Simon JD Prince. *Computer vision: models, learning, and inference*. Cambridge University Press, 2012.

[2] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 9 2017.