Exercise Set 1 | PML Fall 2022

Unless indicated otherwise the exercises refer to "Pattern Regognition and Machine Learning" by CM Bishop (Springer) 2006.

Theory

Basic Probability Theory

1.10)

Let x, y be independent random variables, then the linear property of E is: the expectation is
$E[x+z] = \sum_x \sum_z p(x,z)(x+z) = \sum_x \sum_z p(x)p(z)(x) + \sum_x \sum_z p(x)p(z)(z)$
$= \sum_x p(x)x \sum_z p(z) + \sum_z p(z)z \sum_x p(x) = E[x] \times 1 + E[z] \times 1,$

and variance as $Var[x+z] = E[(x+z)^2] - E[(x+z)]^2$
$= E[x^2 + 2xz + z^2] - (E[x] + E[z])^2$
$= E[x^2] + 2E[x]E[z] + E[z^2] - E[x]^2 - 2E[x]E[z] - E[z]^2$
$= E[x^2] - E[x]^2 + E[z^2] - E[z]^2 = Var[x] + Var[z]$.

2.8)

Consider variables x, y jointly distributed with $p(x, y)$, then:
$E_y[E_x[x|y]] = E_y[\sum_x p(x|y)x] = \sum_y p(y) \sum_x p(x|y)x = \sum_x p(x)x = E[x]$.

For the variance it is the case that:
$E_y[V_x[x|y]] = E_y[E_x[x^2|y] - E_x[x|y]^2] = E_y[E_x[x^2|y]] - E_y[E_x[x|y]^2] = E_x[x^2] - E_y[E_x$
and $V_x[E_x[x|y]] = E_y[E_x[x|y]^2] - E_y[E_x[x|y]]^2$ therefore it follows that
$V[x] = E_x[x^2] - E_y[E_x[x|y]^2] + E_y[E_x[x|y]^2] - E_y[E_x[x|y]]^2 = E_x[x^2] - E_x[x]^2$.

2.20)

A p.d. real valued matrix Σ defined in the quadratic form as $a^T \Sigma a$ is positive for any real value of vector a, because (given symmetry and decomposition) and following matrix notation for (2.45) and (2.48) s.t. $U^T a = y \neq 0$ then: $a^T \Sigma a = a^T (U^T \Lambda U) a = y^T \Lambda y = \sum_i \lambda_i y_i^2 > 0$. Here matrix symmetry and real values allow us to use *spectral decomposition*.

Otherwise using (2.45) : $a^T\Sigma a = \sum_i \lambda_i u_i^T \Sigma \sum_j \lambda_j u_j = \sum_{ij} \lambda_i \lambda_j u_i \Sigma u_j$ and as orthonormal basis we can write $\Sigma u_j = \phi_j u_j$ for eigenvalues ϕ, then $\Rightarrow \sum_{ij} \lambda_i \lambda_j \phi_j u_i u_j = \sum_i \lambda_i^2 \phi_i > 0$.

See Matrix Cookbook (293) section 5.3.1 applicable to real symmetric matrices.

8.3)

Evalutating the joint distribution over three binary variables

```
true
```

```
begin
    using DataFrames
    table = DataFrame(
                    a=[0,0,0,0,1,1,1,1],
                    b=[0,0,1,1,0,0,1,1],
                    c=[0,1,0,1,0,1,0,1],
                    p_abc=[0.192,0.144,0.048,0.216,0.192,0.064,0.048,0.096])
    @show table
    println()

    function test_conditional(a, b, c, table::DataFrame=table)
        p_c = sum(table[table.c .== c, :p_abc])
        p_a_c = sum(filter(row -> row.a == a && row.c == c, table).p_abc)/p_c
        p_b_c = sum(filter(row -> row.b == b && row.c == c, table).p_abc)/p_c
        p_a_b_c = sum(filter(row -> row.a == a && row.b==b && row.c==c,
        table).p_abc)/p_c
        println("p(a=$a,b=$b|c=$c)=$p_a_b_c =
        $p_a_c*$p_b_c=p(a=$a|c=$c)p(b=$b|c=$c)")
        #@assert isequal(p_a_b_c, p_a_c * p_b_c)
        return isequal(p_a_b_c, p_a_c * p_b_c)
    end

    p_a = sum(table[table.a .== 1, :p_abc])
    p_b = sum(table[table.b .== 1, :p_abc])
    # a and b are marginally dependent:
    @assert p_a * p_b != p_a + p_b
    # test all for conditional independence
    map(test_conditional, table.a, table.b, table.c) |> all
end
```

```
table = 8×4 DataFrame
 Row │ a      b      c      p_abc
     │ Int64  Int64  Int64  Float64
─────┼──────────────────────────────
   1 │     0      0      0    0.192
   2 │     0      0      1    0.144
   3 │     0      1      0    0.048
   4 │     0      1      1    0.216
   5 │     1      0      0    0.192
   6 │     1      0      1    0.064
   7 │     1      1      0    0.048
   8 │     1      1      1    0.096

p(a=0,b=0|c=0)=0.4 = 0.5*0.8=p(a=0|c=0)p(b=0|c=0)
p(a=0,b=0|c=1)=0.2769230769230769 = 0.6923076923076923*0.39999999999999997
=p(a=0|c=1)p(b=0|c=1)
p(a=0,b=1|c=0)=0.1 = 0.5*0.2=p(a=0|c=0)p(b=1|c=0)
p(a=0,b=1|c=1)=0.41538461538461535 = 0.6923076923076923*0.6=p(a=0|c=1)p(b=
1|c=1)
p(a=1,b=0|c=0)=0.4 = 0.5*0.8=p(a=1|c=0)p(b=0|c=0)
p(a=1,b=0|c=1)=0.12307692307692307 = 0.3076923076923077*0.3999999999999999
7=p(a=1|c=1)p(b=0|c=1)
p(a=1,b=1|c=0)=0.1 = 0.5*0.2=p(a=1|c=0)p(b=1|c=0)
p(a=1,b=1|c=1)=0.18461538461538463 = 0.3076923076923077*0.6=p(a=1|c=1)p(b=
1|c=1)
```

8.4)

Show $p(a,b,c) = p(a)p(c|a)p(b|c)$.

Remark: $p(a,b,c) = p(a)\frac{p(c,a)}{p(a)}\frac{p(b,c)}{p(c)} = p(c,a)\frac{p(b,c)}{p(c)}$.

```
true
```

```julia
begin
    function test_joint(a,b,c,table::DataFrame=table)
        p_c = sum(table[table.c .== c, :p_abc])
        p_c_a = sum(filter(row -> row.c == c && row.a == a, table).p_abc)
        p_b_c = sum(filter(row -> row.b == b && row.c == c, table).p_abc)/p_c
        p_a_b_c = sum(filter(row -> row.a == a && row.b==b && row.c==c,
        table).p_abc)
        joint = p_c_a*p_b_c
        println("p(a=$a,b=$b,c=$c)=$p_a_b_c = $joint=p(c,a)*p(b|c)/p(c)")
        # @show p_c
        # @show p_c_a
        # @show p_b_c
        # @show p_a_b_c
        return isequal(p_c_a*p_b_c, p_a_b_c)
    end
    map(test_joint, table.a, table.b, table.c) |> all
end
```

```
p(a=0,b=0,c=0)=0.192 = 0.192=p(c,a)*p(b|c)/p(c)
p(a=0,b=0,c=1)=0.144 = 0.144=p(c,a)*p(b|c)/p(c)
p(a=0,b=1,c=0)=0.048 = 0.048=p(c,a)*p(b|c)/p(c)
p(a=0,b=1,c=1)=0.216 = 0.216=p(c,a)*p(b|c)/p(c)
p(a=1,b=0,c=0)=0.192 = 0.192=p(c,a)*p(b|c)/p(c)
p(a=1,b=0,c=1)=0.064 = 0.064=p(c,a)*p(b|c)/p(c)
p(a=1,b=1,c=0)=0.048 = 0.048=p(c,a)*p(b|c)/p(c)
p(a=1,b=1,c=1)=0.096 = 0.096=p(c,a)*p(b|c)/p(c)
```

8.9)

D-separation, we recall:

i) [A] —> [X] —> [B] ; [A] <— [X] —> [B] then $A \perp\!\!\!\perp B|X$, if X is in the conditioning set.

ii) [A]—>[X]<—[B] $A \perp\!\!\!\perp B$ if X not in conditioning set.

For the Markov Blanket see the definition from Bishop (Fig.8.26) and Lecture notes. The answer follows directly from the definition in these materials.

8.11)

[B]—>[G]<—[F]

```
     |
     v
    [D]
```

```
0.10962566844919786
```

```julia
begin
    p_b1=0.9
    p_b0=1-p_b1
    p_f1=0.9
    p_f0=0.1
    p_g0=0.315
    p_g0_f0=0.81
    p_f0_g0=0.257
    p_g1_b1_f1=0.8
    p_g0_b1_f1=1-p_g1_b1_f1
    p_g1_b1_f0=0.2
    p_g0_b1_f0=1-p_g1_b1_f0
    p_g1_b0_f1=0.2
    p_g0_b0_f1=1-p_g1_b0_f1
    p_g1_b0_f0=0.1
    p_g0_b0_f0=1-p_g1_b0_f0
    p_d1_g1 = 0.9
    p_d0_g1 = 1-p_d1_g1
    p_d0_g0 = 0.9
    p_d1_g0 = 1-p_d0_g0
    ## i)
    # step 1: compute likelihood
    p_d0_f0 = (p_d0_g0*p_g0_b0_f0*p_b0+p_d0_g1*p_g1_b0_f0*p_b0) +
    (p_d0_g0*p_g0_b1_f0*p_b1+p_d0_g1*p_g1_b1_f0*p_b1)
    # step 2: compute marginal
    p_d0 =
    (p_d0_g0*p_g0_b0_f0*p_b0*p_f0+p_d0_g1*p_g1_b0_f0*p_b0*p_f0+p_d0_g0*p_g0_b1_f
    0*p_b1*p_f0+p_d0_g1*p_g1_b1_f0*p_b1*p_f0)+
    (p_d0_g0*p_g0_b0_f1*p_b0*p_f1+p_d0_g1*p_g1_b0_f1*p_b0*p_f1+p_d0_g0*p_g0_b1_f
    1*p_b1*p_f1+p_d0_g1*p_g1_b1_f1*p_b1*p_f1) # (F=0) + (F=1)
    # step 3: Bayes Rule
    @show p_f0_d0 = (p_d0_f0*p_f0)/p_d0
    ## ii)
    # step 1: compute likelihood
    p_d0_b0_f0 = (p_d0_g0*p_g0_b0_f0*p_b0) + (p_d0_g1*p_g1_b0_f0*p_b0)
    # step 2: compute marginal
    p_d0_b0 = p_d0_g0*p_g0_b0_f0*p_b0*p_f0 + p_d0_g1*p_g1_b0_f0*p_b0*p_f0 +
    p_d0_g0*p_g0_b0_f1*p_b0*p_f1 + p_d0_g1*p_g1_b0_f1*p_b0*p_f1
    # step 3: Bayes Rule
    @show p_f0_d0_b0 = (p_d0_b0_f0*p_f0)/p_d0_b0
end
```

```
p_f0_d0 = (p_d0_f0 * p_f0) / p_d0 = 0.21250000000000008
p_f0_d0_b0 = (p_d0_b0_f0 * p_f0) / p_d0_b0 = 0.10962566844919786
```

Programming Exercise

```
TaskLocalRNG()
```

```julia
begin
    using Random
    using Distributions
    using LinearAlgebra
    using PlutoUI
    Random.seed!(42)
end
```

Solve for D: ☐

```
(2×2 Matrix{Float64}:, 0.1)
  1.0  0.0
  0.0  1.0
```

```julia
begin
    N=2
    Σ_x = convert(Array{Float64}, collect(I(N)))
    σ = 0.1 # Option: A,B,C
    if option
        Σ_x[1,1] = 0.1 # Option for D)
        # σ = 0.01 # Even more options here...
    end
    Σ_x, σ
end
```

A)

Create dataset for regression.:

```
(20×2 Matrix{Float64}: , [0.3581, 0.705565, -0.244711, 1.2944, 0.472981, 2.10565, 0.
  -0.954618  -0.61694
   0.888373   1.70221
  -0.307589  -0.595113
   0.240871   1.34142
  -0.312024   0.220427
   0.199745   2.21349
  -1.41501   -0.83954
   ⋮
  -0.120277  -0.289603
  -1.14098    0.220411
   0.520437   0.765463
   1.37368    1.73886
   0.576634  -0.0593636
  -1.16674   -0.973451
```

```
begin
    sample_size=20
    dim = 2
    μ = [0., 0.]
    N_x = MvNormal(μ, Σ_x)
    X = reshape(rand(N_x, sample_size), (sample_size, dim))
    θ = [-1,1]
    y = (X*θ) .+ rand(Normal(0, σ), sample_size)
    X, y
end
```

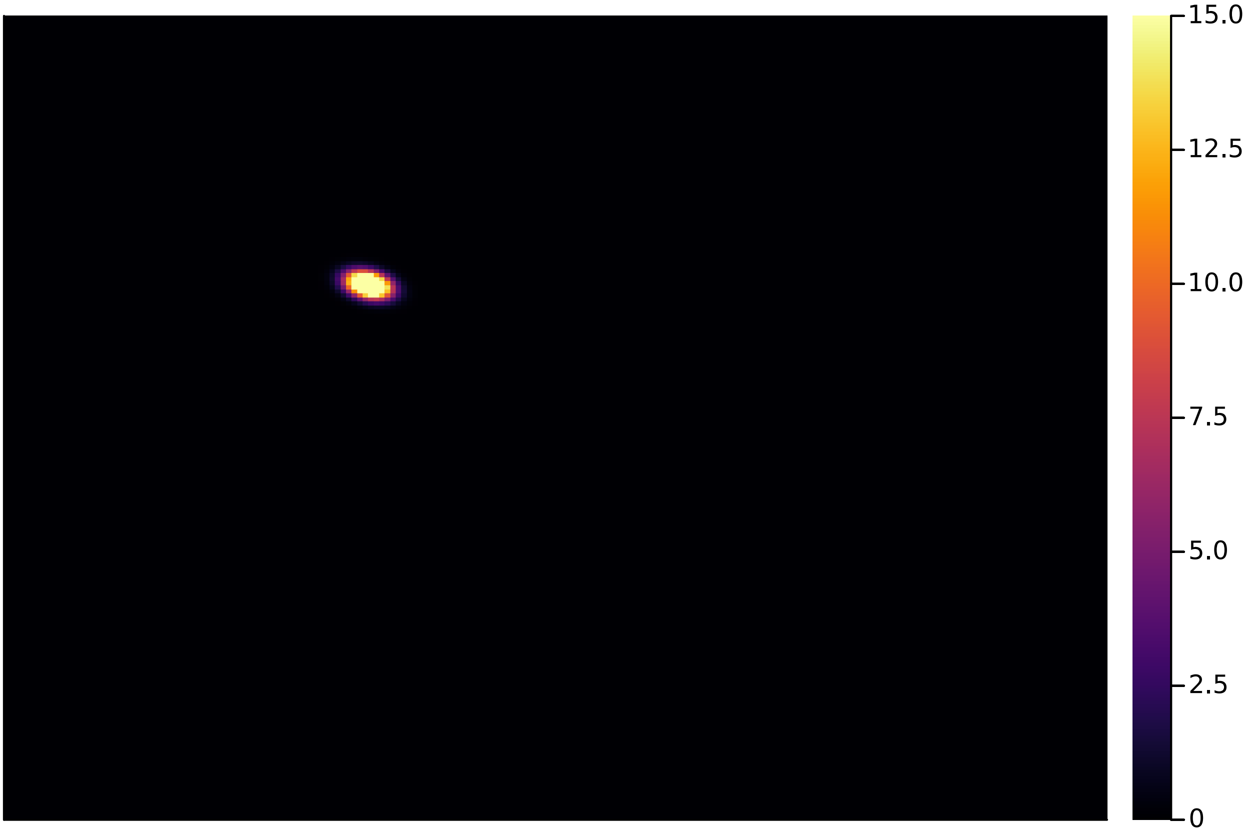
B)

Compute posterior mean and variance - see script Eq. (55),(56):

```
2×2 Matrix{Float64}:
  0.00583072  -0.00132868
 -0.00132868   0.00395157
```

```
begin
    μ_θ = X'*inv(σ.*I + X*X') * y # Eq.(55)
    Σ_θ = I - X'*inv(σ.*I + X*X') * X # Eq.(56)
    @show μ_θ
    @show Σ_θ
end
```

```
μ_θ = [-1.0226511074024365, 0.9979297993773675]
Σ_θ = [0.005830718405949509 -0.001328680661916788; -0.0013286806619201591
0.003951568963384555]
```




```
begin
# plot PDF
    using Base.Iterators
    using Plots
    θ_range = range(-3, stop=3, length=200)
    xx = collect(product(θ_range, θ_range))
    XX = [x[i] for x in [xx...], i ∈ 1:2]
    N_θ = MvNormal(μ_θ, Hermitian(Σ_θ)) # force it to assume p.d. cov.
    values = reshape(pdf(N_θ, XX'), (size(θ_range)[1], size(θ_range)[1]))
    heatmap(values', clims=(0, 15), xaxis=nothing, yaxis=nothing)
end
```

C)

Compute Variance of posterior predictive


```
begin
    variance = sum((XX*Σ_θ).*XX, dims=2) .+ σ
    variance = reshape(variance, (size(θ_range)[1], size(θ_range)[1]))
    heatmap(variance', clims=(0., 0.4), xaxis=nothing, yaxis=nothing)
end
```