In [19]:
```
# check current directory

pwd()
```

```
"/Users/xyliu/Desktop/ISE 616 final/ISE-616-final"
```

In [40]:
```
using JuMP
using Ipopt
using LinearAlgebra
using MathOptInterface
const MOI = MathOptInterface
using Random
```

Modeling

function of one-hot encoding

In [21]:
```
############################
# Encoding information type
############################

"""
    ZGEncodingInfo

Stores the information needed to convert between
original (integer-coded) categorical/group features
and their one-hot encodings.

Fields:
- k_z::Vector{Int}     : number of categories for each categorical compon
- z_start::Vector{Int} : start index (column) of each component in Z_oneh
- num_g::Int           : total number of groups
"""

struct ZGEncodingInfo
    k_z::Vector{Int}
    z_start::Vector{Int}
    num_g::Int
end



#############################################
# Original (Z, group) → Reduced (Z_enc, G_enc)
#############################################

"""
    encode_zg_reduced(Z::AbstractMatrix{<:Integer},
                      group::AbstractVector{<:Integer})

Encode categorical features Z and group indexes into a reduced dummy
representation:

- Each categorical component l with k_l levels is mapped to (k_l - 1)
  dummy variables. The last category k_l is the baseline (all zeros).
```

```
- The group variable with num_g levels is mapped to (num_g − 1)
  dummy variables. The last group num_g is the baseline (all zeros).

Input:
- Z     : N × m matrix, each column l stores an integer in {1, ..., k_l}
- group : length-N vector, each entry in {1, ..., num_g}

Output:
- Z_enc : N × sum_l (k_l − 1) matrix (reduced dummies for Z)
- G_enc : N × (num_g − 1) matrix (reduced dummies for group)
- info  : ZGEncodingInfo, stores the encoding scheme for later decoding
"""
function encode_zg_reduced(Z::AbstractMatrix{<:Integer},
                           group::AbstractVector{<:Integer})
    N, m = size(Z)
    @assert length(group) == N "group must have length N"

    # k_z[l] = number of categories for component l (assumed {1, ..., k_l
    k_z = [maximum(Z[:, l]) for l in 1:m]

    # Column start indices in the reduced dummy matrix Z_enc.
    # Component l uses (k_z[l] − 1) columns (baseline = category k_z[l]).
    z_start = Vector{Int}(undef, m)
    col = 1
    for l in 1:m
        z_start[l] = col
        d_l = max(k_z[l] - 1, 0)  # number of dummies for component l
        col += d_l
    end
    total_z = col - 1   # total number of columns in Z_enc

    # Allocate Z_enc: N × total_z
    Z_enc = zeros(Int8, N, total_z)

    for i in 1:N
        for l in 1:m
            val = Z[i, l]
            @assert 1 ≤ val ≤ k_z[l] "Category out of range in column $l

            k_l = k_z[l]
            d_l = max(k_l - 1, 0)

            # If d_l == 0, there is effectively a single category (no col
            if d_l > 0 && val < k_l
                col0 = z_start[l]             # start index for componen
                Z_enc[i, col0 + val - 1] = 1   # categories 1..(k_l−1)
            end
            # If val == k_l, baseline category -> all zeros for this comp
        end
    end

    # Encode group using (num_g − 1) dummies; last group is baseline.
    num_g = maximum(group)
    d_g   = max(num_g - 1, 0)
    G_enc = zeros(Int8, N, d_g)

    if d_g > 0
        for i in 1:N
            gi = group[i]
            @assert 1 ≤ gi ≤ num_g "Group index out of range at row $i"
```

```
            if gi < num_g
                # Non-baseline group in {1, ..., num_g - 1}
                G_enc[i, gi] = 1
            else
                # Baseline group num_g -> all zeros
            end
        end
    end

    info = ZGEncodingInfo(k_z, z_start, num_g)
    return Z_enc, G_enc, info
end
```

encode_zg_reduced

test one-hot encode

In [22]:
```
# testing
Z = [
    1  1;    # row 1
    2  3;    # row 2
    3  2;    # row 3
    1  3;    # row 4
]
# group: length-N vector, values in {1,2,3}
group = [1, 2, 1, 3]

#######################
# 2. Run encoding
#######################

Z_enc, G_onehot, info = encode_zg_reduced(Z, group)
println("Original Z:")
println(Z)
println()

println("Encoded Z_enc (reduced one-hot):")
println(Z_enc)
println("size(Z_enc) = ", size(Z_enc))
println()

println("Group one-hot G_onehot:")
println(G_onehot)
println("size(G_onehot) = ", size(G_onehot))
println()

println("Encoding info:")
println("  k_z     = ", info.k_z)
println("  z_start = ", info.z_start)
println("  num_g   = ", info.num_g)
```

```
Original Z:
[1 1; 2 3; 3 2; 1 3]

Encoded Z_enc (reduced one-hot):
Int8[1 0 1 0; 0 1 0 0; 0 0 0 1; 1 0 0 0]
size(Z_enc) = (4, 4)

Group one-hot G_onehot:
Int8[1 0; 0 1; 1 0; 0 0]
size(G_onehot) = (4, 2)

Encoding info:
  k_z     = [3, 3]
  z_start = [1, 3]
  num_g   = 3
```

In [23]:

```
#############################################
# Reduced (Z_enc, G_enc) → Original (Z, group)
#############################################

"""
    decode_zg_reduced(Z_enc::AbstractMatrix{<:Integer},
                      G_enc::AbstractMatrix{<:Integer},
                      info::ZGEncodingInfo)

Decode reduced dummy encoded categorical and group features back to
their original integer-coded form.

For each categorical component l with k_l levels:
- If the corresponding block has all zeros, we recover category k_l
  (the baseline).
- Otherwise, the position of the 1 determines the category in {1, ..., k_

For the group variable with num_g levels:
- If the row in G_enc is all zeros, we recover group num_g (baseline).
- Otherwise, the position of the 1 determines the group in {1, ..., num_g

Input:
- Z_enc : N × sum_l (k_l - 1) matrix (reduced dummies for Z)
- G_enc : N × (num_g - 1) matrix (reduced dummies for group)
- info  : ZGEncodingInfo produced earlier by `encode_zg_reduced`

Output:
- Z     : N × m matrix, each entry in {1, ..., k_l}
- group : length-N vector, each entry in {1, ..., num_g}
"""
function decode_zg_reduced(Z_enc::AbstractMatrix{<:Integer},
                           G_enc::AbstractMatrix{<:Integer},
                           info::ZGEncodingInfo)
    N, total_z = size(Z_enc)
    m = length(info.k_z)

    # Check group encoding consistency
    num_g = info.num_g
    d_g   = max(num_g - 1, 0)
    @assert size(G_enc, 1) == N "Z_enc and G_enc must have same number of
    @assert size(G_enc, 2) == d_g "G_enc columns must be num_g - 1"

    # Recover categorical matrix Z (N × m)
    Z = zeros(Int, N, m)
```

```
    for l in 1:m
        k_l = info.k_z[l]
        d_l = max(k_l - 1, 0)
        s   = info.z_start[l]
        e   = s + d_l - 1

        if d_l == 0
            # Only one category exists for this component; always categor
            for i in 1:N
                Z[i, l] = 1
            end
            continue
        end

        @assert e ≤ total_z "Z_enc has too few columns for component $l"

        # View the reduced dummy block for component l
        sub = @view Z_enc[:, s:e]  # N × d_l
        for i in 1:N
            idx = findfirst(==(1), sub[i, :])
            if idx === nothing
                # All zeros: baseline category k_l
                Z[i, l] = k_l
            else
                # Non-baseline category in {1, ..., k_l - 1}
                Z[i, l] = idx
            end
        end
    end

    # Recover group vector (length N) from reduced dummies
    group = Vector{Int}(undef, N)
    if d_g == 0
        # Only one group; always group 1
        for i in 1:N
            group[i] = 1
        end
    else
        for i in 1:N
            idx = findfirst(==(1), G_enc[i, :])
            if idx === nothing
                # Baseline group num_g
                group[i] = num_g
            else
                # Non-baseline group in {1, ..., num_g - 1}
                group[i] = idx
            end
        end
    end

    return Z, group
end
```

```
decode_zg_reduced
```

test one-hot decode

In [24]:
```
Z, group = decode_zg_reduced(Z_enc, G_onehot, info)
Z, group
```

```
([1 1; 2 3; 3 2; 1 3], [1, 2, 1, 3])
```

DAG builder

In [25]:
```
################################
# Graph structure for sample-level DAGs
# compatible with Theorem 2 (graph-based formulation)
################################

# -----------------------------
# Arc kind (structure only)
# -----------------------------

"""
    ArcKind

Abstract type for different kinds of arcs in the DAG.
We distinguish between:
  - `CatArc` : categorical feature transitions
  - `TermArc`: terminal transitions to (m+1, 0) with a chosen group g
"""
abstract type ArcKind end

"""
    CatArc

Categorical arc corresponding to choosing category `c` for component `k`.

Fields:
- k      : which categorical component (1..m)
- c      : chosen category in {1, ..., k_z[k]}
- d_prev : previous accumulated categorical distance at state (k-1, d_pre
- d      : new accumulated distance at state (k, d)
"""

struct CatArc <: ArcKind
    k::Int
    c::Int
    d_prev::Float64
    d::Float64
end

"""
    TermArc

Terminal arc from (m, d) to (m+1, 0) with a chosen destination group g.

Fields:
- d : accumulated categorical distance at state (m, d)
- g : destination group in {1, ..., num_g}
"""
struct TermArc <: ArcKind
    d::Float64
    g::Int
end

# -----------------------------
# Arc and per-sample DAG
# -----------------------------
```

```
"""
    Arc

Directed edge in the DAG for a single sample i.

Fields:
- src  : index of the source node in `nodes`
- dst  : index of the destination node in `nodes`
- kind : arc type (CatArc or TermArc), which encodes all information
         needed to build w^i(e; β, λ, r_i, ...) later
"""
struct Arc
    src::Int
    dst::Int
    kind::ArcKind
end




"""
    SampleDAG

Graph structure associated with a single sample i, compatible with
Theorem 2 (graph-based formulation).

Fields:
- sample_index : index i of the sample this DAG corresponds to
- nodes        : vector of DP states (k, d), plus the terminal node (m+1,
                 each node is a Tuple{Int,Float64}
- arcs         : list of directed edges with their arc kind (no numeric w
- source       : index of the source node (corresponds to state (0, 0.0))
- sink         : index of the terminal node (corresponds to state (m+1, 0
"""
struct SampleDAG
    sample_index::Int
    nodes::Vector{Tuple{Int,Float64}}
    arcs::Vector{Arc}
    source::Int
    sink::Int
end




# ------------------------------
# Categorical encoding info
# ------------------------------
"""
    CatEncodingInfo

Encoding information for categorical features (structure level only).

Fields:
- k_z::Vector{Int} : number of categories per component (length m).
                     For component k, categories are {1, ..., k_z[k]}.
- num_g::Int       : total number of groups (used for terminal arcs).
```

```
"""
struct CatEncodingInfo
    k_z::Vector{Int}
    num_g::Int
end


# ------------------------------
# Build DAG structure for one sample i
# ------------------------------
"""
    build_sample_dag_structure(i, info, delta, z_i) -> SampleDAG

Build the DAG structure G^i = (V^i, A^i) for a fixed sample i,
compatible with Theorem 2 (graph-based formulation), using our setup.

This function ONLY builds:
  - the nodes (states (k, d) plus the terminal (m+1, 0)),
  - the arcs with their structural information (CatArc or TermArc).

It does NOT compute numeric weights w^i(e). Those should be constructed
later as expressions of (β, λ, r_i, y_i, g_i, B_{g_i}, C_{g_i}, ...).

Arguments:
- i      : sample index
- info   : CatEncodingInfo (k_z, num_g)
- delta  : length-m vector of δ_k in
           d(z, zᴵ) = Σ_k δ_k * 1[z_k ≠ z_kᴵ]
- z_i    : length-m vector of original categories for sample i,
           each z_i[k] ∈ {1, ..., k_z[k]}

Output:
- SampleDAG describing the structure of G^i
"""
function build_sample_dag_structure(
    i::Int,
    info::CatEncodingInfo,
    delta::AbstractVector{<:Real},
    z_i::AbstractVector{<:Integer}
)::SampleDAG
    k_z   = info.k_z
    num_g = info.num_g
    m     = length(k_z)

    # -------------------------
    # 0. Sanity checks
    # -------------------------
    @assert length(delta) == m "delta must have length m"
    @assert length(z_i) == m "z_i must have length m"

    # -------------------------
    # 1. Enumerate states (k, d) with per-layer dedup
    # -------------------------
    nodes = Vector{Tuple{Int,Float64}}()
    node_index = Dict{Tuple{Int,Float64},Int}()

    # Source state (0, 0.0)
    push!(nodes, (0, 0.0))
    node_index[(0, 0.0)] = 1
    source_idx = 1
```

```
arcs = Vector{Arc}()

# current_layer holds all unique states (k-1, d_prev)
current_layer = [(0, 0.0)]

for k in 1:m
    next_layer = Tuple{Int,Float64}[]
    seen_next = Set{Tuple{Int,Float64}}()

    k_l   = k_z[k]
    δ_k   = float(delta[k])
    z_i_k = z_i[k]

    for (k_prev, d_prev) in current_layer
        @assert k_prev == k - 1

        # Enumerate all categories c ∈ {1, ..., k_l}
        for c in 1:k_l
            mismatch = (c != z_i_k)
            d = d_prev + (mismatch ? δ_k : 0.0)
            state = (k, d)

            # Add state to global node list if new
            if !haskey(node_index, state)
                push!(nodes, state)
                node_index[state] = length(nodes)
            end

            # Ensure each (k, d) appears at most once in next_layer
            if !(state in seen_next)
                push!(next_layer, state)
                push!(seen_next, state)
            end

            # Add categorical arc from (k-1, d_prev) to (k, d)
            src = node_index[(k-1, d_prev)]
            dst = node_index[state]
            kind = CatArc(k, c, d_prev, d)
            push!(arcs, Arc(src, dst, kind))
        end
    end

    # Move to next layer (already deduplicated)
    current_layer = next_layer
end

# States with k = m are the final DP layer S_1^i
# current_layer is already deduplicated, but we keep the name for cla
S1_states = current_layer

# -------------------------
# 2. Add terminal node (m+1, 0.0) and terminal arcs
# -------------------------
terminal_state = (m+1, 0.0)
push!(nodes, terminal_state)
node_index[terminal_state] = length(nodes)
sink_idx = node_index[terminal_state]

# For each (m, d) in S_1^i, and for each group g, add a TermArc
```

```
    for (k_state, d) in S1_states
        @assert k_state == m
        src = node_index[(m, d)]
        for g in 1:num_g
            kind = TermArc(d, g)
            push!(arcs, Arc(src, sink_idx, kind))
        end
    end

    # --------------------------
    # 3. Return SampleDAG
    # --------------------------
    return SampleDAG(i, nodes, arcs, source_idx, sink_idx)
end
```

```
build_sample_dag_structure
```

test DAG builder

In [26]:

```
i = 2001

k_z   = [2, 2]       # each component 2 cats
delta = [1.0, 1.0]   # δ₁ = δ₂ = 1
z_i   = [1, 2]       # sample i: z₁ = 1, z₂ = 2
num_g = 2

info = CatEncodingInfo(k_z, num_g)

dag_i = build_sample_dag_structure(i, info, delta, z_i)
dag_i
```

```
SampleDAG(2001, [(0, 0.0), (1, 0.0), (1, 1.0), (2, 1.0), (2, 0.0), (2, 2.
0), (3, 0.0)], Arc[Arc(1, 2, CatArc(1, 1, 0.0, 0.0)), Arc(1, 3, CatArc(1,
2, 0.0, 1.0)), Arc(2, 4, CatArc(2, 1, 0.0, 1.0)), Arc(2, 5, CatArc(2, 2,
0.0, 0.0)), Arc(3, 6, CatArc(2, 1, 1.0, 2.0)), Arc(3, 4, CatArc(2, 2, 1.0,
1.0)), Arc(4, 7, TermArc(1.0, 1)), Arc(4, 7, TermArc(1.0, 2)), Arc(5, 7, T
ermArc(0.0, 1)), Arc(5, 7, TermArc(0.0, 2)), Arc(6, 7, TermArc(2.0, 1)), A
rc(6, 7, TermArc(2.0, 2))], 1, 7)
```

modeling

In [27]:

```
using JuMP

"""
    build_group_dro_graph_model(
        X, Z, group, y,
        encinfo,
        delta,
        A_group,
        B_group, C_group,
        gamma_x,
        ε,
        optimizer
    ) -> (model, meta)

Build our group-dependent, graph-based DRO logistic regression model,
using the same reduced encoding convention as `ZGEncodingInfo` /
`encode_zg_reduced`.
```

```
Arguments
---------
- X        :: N × n_x matrix of continuous features.
- Z        :: N × m   matrix of original categorical features.
              Entry Z[i, k] ∈ {1, ..., k_z[k]} (no one-hot).

- group    :: length-N vector of group indices g_i ∈ {1,...,num_g}.
- y        :: length-N vector of labels in {-1, +1}.

- encinfo  :: ZGEncodingInfo
              (k_z, z_start, num_g) describing reduced encoding blocks
              for the categorical features and groups.

- delta    :: length-m vector δ_k used in
              d_cat(z, z^i) = Σ_k δ_k * 1[z_k ≠ z_k^i].

- A_group  :: length-num_g vector A_g
              continuous-part metric weights in
              A_g Σ_j γ_j |x_j - x_j^i|.

- B_group  :: length-num_g vector B_g
              categorical-part metric weight.

- C_group  :: length-num_g vector C_g
              group-change penalty.

- gamma_x  :: length n_x vector γ_j for continuous features.
              Continuous dual constraint will be
                  |β_{xj}| ≤ λ * A_min * γ_j,
              where A_min = minimum(A_group).

- ε        :: Wasserstein radius ε.

- optimizer: optimizer constructor for JuMP, e.g.
              optimizer_with_attributes(Mosek.Optimizer, "QUIET" => true)

Returns
-------
- model :: JuMP.Model
- meta  :: NamedTuple: (encinfo=encinfo, dags=dags, n_nodes=n_nodes)
"""
function build_group_dro_graph_model(
    X::AbstractMatrix{<:Real},
    Z::AbstractMatrix{<:Integer},
    group::AbstractVector{<:Integer},
    y::AbstractVector{<:Integer},
    encinfo::ZGEncodingInfo,
    delta::AbstractVector{<:Real},
    A_group::AbstractVector{<:Real},
    B_group::AbstractVector{<:Real},
    C_group::AbstractVector{<:Real},
    gamma_x::AbstractVector{<:Real},
    ε::Real,
    optimizer,
)
    # -------------------------
    # 0. Dimensions & sanity checks
    # -------------------------
    N, n_x = size(X)
```

```
    N_Z, m = size(Z)
    @assert N_Z == N "X and Z must have the same number of rows (samples)
    @assert length(group) == N "group must have length N."
    @assert length(y) == N "y must have length N."
    @assert length(delta) == m "delta must have length m."
    @assert length(gamma_x) == n_x "gamma_x must have length n_x."

    k_z     = encinfo.k_z
    z_start = encinfo.z_start
    num_g   = encinfo.num_g

    @assert length(k_z) == m "encinfo.k_z must have length m."
    @assert length(z_start) == m "encinfo.z_start must have length m."
    @assert length(A_group) == num_g "A_group length must equal num_g."
    @assert length(B_group) == num_g "B_group length must equal num_g."
    @assert length(C_group) == num_g "C_group length must equal num_g."

    # Total length of β_z under reduced encoding:
    # last block starts at z_start[m], length (k_z[m] - 1)
    # so p_z = z_start[m] + (k_z[m] - 1) - 1
    p_z = z_start[end] + (k_z[end] - 1) - 1

    # Small positive constant for log(exp(inner) - 1) domain
    # We will enforce: r[i] + λ * inner_coeff ≥ η
    η = 1e-6

    # -------------------------
    # 1. Build per-sample DAGs (discrete part only)
    # -------------------------
    info = CatEncodingInfo(k_z, num_g)

    dags      = Vector{SampleDAG}(undef, N)
    n_nodes   = Vector{Int}(undef, N)
    max_nodes = 0

    for i in 1:N
        # Z[i, :] is an AbstractVector{<:Integer}
        dags[i] = build_sample_dag_structure(i, info, delta, Z[i, :])
        n_nodes[i] = length(dags[i].nodes)
        max_nodes = max(max_nodes, n_nodes[i])
    end

    # -------------------------
    # 2. Create JuMP model & decision variables
    # -------------------------
    model = Model(optimizer)

    # Wasserstein dual variable λ ≥ 0
    @variable(model, λ >= 0.0)

    # Per-sample slack variables r_i ∈ ℝ (free),
    # domain of log(exp(...)-1) will be enforced via extra constraints
    @variable(model, r[1:N])

    # Logistic regression parameters
    @variable(model, β0)                # intercept
    @variable(model, β_x[1:n_x])        # continuous coefficients
    @variable(model, β_z[1:p_z])        # categorical coefficients (reduc
    @variable(model, β_grp[1:(num_g-1)])# group coefficients (reduced)
```

```
    # Graph dual variables μ_i_v for each sample i and each node v
    @variable(model, μ[1:N, 1:max_nodes])

    # -------------------------
    # 3. Objective: λ ε + (1/N) Σ r_i
    # -------------------------
    @objective(model, Min, λ * ε + (1.0 / N) * sum(r[i] for i in 1:N))

    # -------------------------
    # 4. Continuous part dual constraints with A_group
    #
    # Metric for x uses A_g:
    #   d_x(x^i, x; g) = A_g Σ_j γ_j |x_j - x_j^i|
    # Dual boundedness ⇒ for each j:
    #   |β_{xj}| ≤ λ * γ_j * min_g A_g
    # We enforce:
    #   -λ * A_min * γ_j ≤ β_x[j] ≤ λ * A_min * γ_j
    # -------------------------
    A_min = minimum(A_group)

    for j in 1:n_x
        @constraint(model,  β_x[j] <=  λ * A_min * gamma_x[j])
        @constraint(model, -β_x[j] <=  λ * A_min * gamma_x[j])
    end

    # -------------------------
    # 5. Outer logistic inequality:
    #    y^i (β_x^T x^i + β0) ≥ -μ_i(0,0) + μ_i(m+1,0)
    # -------------------------
    for i in 1:N
        dag = dags[i]
        s = dag.source
        t = dag.sink

        # Left-hand side: y_i * (β0 + β_x^T x^i)
        lhs = y[i] * (β0 + sum(β_x[j] * X[i, j] for j in 1:n_x))

        # Right-hand side: - μ_i(source) + μ_i(sink)
        rhs = - μ[i, s] + μ[i, t]

        @constraint(model, lhs >= rhs)
    end

    # -------------------------
    # 6. Edge constraints: μ_t(e) - μ_s(e) ≥ w^i(e)
    #
    # - For CatArc(k,c,...):
    #     w^i(e) = - y^i β_{z_k}^T z_k(c)
    #   Reduced encoding aligned with ZGEncodingInfo:
    #     if c < k_z[k], β_{z_k}^T z_k(c) = β_z[z_start[k] + c - 1]
    #     if c = k_z[k], baseline ⇒ 0.
    #
    # - For TermArc(d,g):
    #     w^i(e) =
    #       - y^i β_grp^T φ_g(g)
    #       - log( exp( r_i + λ (B_{g_i} d + C_{g_i} 1[g ≠ g_i]) ) - 1 )
    #
    #   with group reduced encoding:
    #     if g < num_g: β_grp^T φ_g(g) = β_grp[g]
    #     if g = num_g: baseline ⇒ 0.
```

```
    #
    #   Additionally, domain constraints:
    #     r_i + λ (B_{g_i} d + C_{g_i} 1[g ≠ g_i]) ≥ η
    #   ensure that log(exp(inner) − 1) is well-defined.
    # -------------------------
    for i in 1:N
        dag  = dags[i]
        y_i  = y[i]
        g_i  = group[i]

        for arc in dag.arcs
            src = arc.src
            dst = arc.dst

            if arc.kind isa CatArc
                kind = arc.kind::CatArc
                k = kind.k
                c = kind.c
                k_l = k_z[k]

                # Categorical part:
                # w_cat^i(e) = − y_i β_{z_k}^T z_k(c)
                if c < k_l
                    idx = z_start[k] + (c - 1)
                    w_expr = - y_i * β_z[idx]
                else
                    w_expr = 0.0
                end

                @constraint(model, μ[i, dst] - μ[i, src] >= w_expr)

            elseif arc.kind isa TermArc
                kind     = arc.kind::TermArc
                d_val    = kind.d
                g_choice = kind.g

                # Metric coefficient for categorical + group part:
                B_gi = float(B_group[g_i])
                C_gi = float(C_group[g_i])
                cross = (g_choice != g_i) ? C_gi : 0.0
                inner_coeff = B_gi * d_val + cross

                # --- Domain constraint: inner = r[i] + λ * inner_coeff ≥
                @NLconstraint(model, r[i] + λ * inner_coeff >= η)

                # --- w^i(e) constraint with nonlinear log/exp ---
                if g_choice < num_g
                    # group linear term: − y_i * β_grp[g_choice]
                    @NLconstraint(model,
                        μ[i, dst] - μ[i, src] >=
                        - y_i * β_grp[g_choice] -
                        log(exp(r[i] + λ * inner_coeff) - 1)
                    )
                else
                    # baseline group: no β_grp contribution
                    @NLconstraint(model,
                        μ[i, dst] - μ[i, src] >=
                        - log(exp(r[i] + λ * inner_coeff) - 1)
                    )
                end
```

```
            else
                error("Unknown arc kind in DAG.")
            end
        end
    end

    # -------------------------
    # 7. (Optional) Give Ipopt a safe starting point inside the domain
    # -------------------------
    set_start_value(λ, 1.0)
    for i in 1:N
        set_start_value(r[i], 1.0)
    end

    # -------------------------
    # 8. Return model + metadata
    # -------------------------
    meta = (
        encinfo = encinfo,
        dags    = dags,
        n_nodes = n_nodes,

        β0      = β0,
        β_x     = β_x,
        β_z     = β_z,
        β_grp   = β_grp,
        λ       = λ,
        r       = r,
        μ       = μ,
    )

    return model, meta
end
```

```
build_group_dro_graph_model
```

Toy example

In [28]:

```
# ========== Toy data ==========
N   = 2      # two samples
n_x = 1      # one continuous feature
m   = 2      # two categorical components
num_g = 2    # two groups

# Continuous features X: N × n_x
X = [
    0.0;
    1.0
]
X = reshape(X, N, n_x)   # 2×1 matrix

# Categorical features Z: N × m, z_{i,k} ∈ {1,2}
Z = [
    1  1;    # sample 1: (1,1)
    2  2     # sample 2: (2,2)
]

# Group indices g_i ∈ {1,2}
group = [1, 2]
```

```julia
# Labels y_i ∈ {-1, +1}
y = [1, -1]

# Hamming weights δ_k
delta = [1.0, 1.0]

# Metric weights
A_group = [1.0, 2.0]   # continuous part weights A_g
B_group = [1.0, 1.0]   # categorical part weights B_g
C_group = [0.5, 0.5]   # group-change penalty C_g

# Continuous scaling γ_x
gamma_x = [1.0]        # length n_x

# Wasserstein radius
ε = 0.1
```

```
0.1
```

In [29]:

```julia
# ========== Encoding info ==========
k_z = [2, 2]

# reduced encoding block starts for categorical β_z
# block 1 starts at 1, length (k_z[1]-1) = 1
# block 2 starts at 2, length (k_z[2]-1) = 1
z_start = [1, 2]

struct ZGEncodingInfo
    k_z::Vector{Int}
    z_start::Vector{Int}
    num_g::Int
end

encinfo = ZGEncodingInfo(k_z, z_start, num_g)
```

```
ZGEncodingInfo([2, 2], [1, 2], 2)
```

In [30]:

```julia
# ========== Build model ==========
model, meta = build_group_dro_graph_model(
    X,
    Z,
    group,
    y,
    encinfo,
    delta,
    A_group,
    B_group,
    C_group,
    gamma_x,
    ε,
    optimizer_with_attributes(Ipopt.Optimizer) # or just Ipopt.Optimizer
)

println("Model successfully built.")
println(model)
```

```
Model successfully built.
Min 0.1 λ + 0.5 r[1] + 0.5 r[2]
Subject to
 β0 + μ[1,1] - μ[1,7] ≥ 0
 -β0 - β_x[1] + μ[2,1] - μ[2,7] ≥ 0
 β_z[1] - μ[1,1] + μ[1,2] ≥ 0
 -μ[1,1] + μ[1,3] ≥ 0
 β_z[2] - μ[1,2] + μ[1,4] ≥ 0
 -μ[1,2] + μ[1,5] ≥ 0
 β_z[2] - μ[1,3] + μ[1,5] ≥ 0
 -μ[1,3] + μ[1,6] ≥ 0
 -β_z[1] - μ[2,1] + μ[2,2] ≥ 0
 -μ[2,1] + μ[2,3] ≥ 0
 -β_z[2] - μ[2,2] + μ[2,4] ≥ 0
 -μ[2,2] + μ[2,5] ≥ 0
 -β_z[2] - μ[2,3] + μ[2,5] ≥ 0
 -μ[2,3] + μ[2,6] ≥ 0
 -λ + β_x[1] ≤ 0
 -λ - β_x[1] ≤ 0
 λ ≥ 0
 (r[1] + λ * 0.0) - 1.0e-6 ≥ 0
 ((μ[1,7] - μ[1,4]) - (-1.0 * β_grp[1] - log(exp(r[1] + λ * 0.0) - 1.0)))
- 0.0 ≥ 0
 (r[1] + λ * 0.5) - 1.0e-6 ≥ 0
 ((μ[1,7] - μ[1,4]) - -(log(exp(r[1] + λ * 0.5) - 1.0))) - 0.0 ≥ 0
 (r[1] + λ * 1.0) - 1.0e-6 ≥ 0
 ((μ[1,7] - μ[1,5]) - (-1.0 * β_grp[1] - log(exp(r[1] + λ * 1.0) - 1.0)))
- 0.0 ≥ 0
 (r[1] + λ * 1.5) - 1.0e-6 ≥ 0
 ((μ[1,7] - μ[1,5]) - -(log(exp(r[1] + λ * 1.5) - 1.0))) - 0.0 ≥ 0
 (r[1] + λ * 2.0) - 1.0e-6 ≥ 0
 ((μ[1,7] - μ[1,6]) - (-1.0 * β_grp[1] - log(exp(r[1] + λ * 2.0) - 1.0)))
- 0.0 ≥ 0
 (r[1] + λ * 2.5) - 1.0e-6 ≥ 0
 ((μ[1,7] - μ[1,6]) - -(log(exp(r[1] + λ * 2.5) - 1.0))) - 0.0 ≥ 0
 (r[2] + λ * 2.5) - 1.0e-6 ≥ 0
 ((μ[2,7] - μ[2,4]) - (--1.0 * β_grp[1] - log(exp(r[2] + λ * 2.5) - 1.0)))
- 0.0 ≥ 0
 (r[2] + λ * 2.0) - 1.0e-6 ≥ 0
 ((μ[2,7] - μ[2,4]) - -(log(exp(r[2] + λ * 2.0) - 1.0))) - 0.0 ≥ 0
 (r[2] + λ * 1.5) - 1.0e-6 ≥ 0
 ((μ[2,7] - μ[2,5]) - (--1.0 * β_grp[1] - log(exp(r[2] + λ * 1.5) - 1.0)))
- 0.0 ≥ 0
 (r[2] + λ * 1.0) - 1.0e-6 ≥ 0
 ((μ[2,7] - μ[2,5]) - -(log(exp(r[2] + λ * 1.0) - 1.0))) - 0.0 ≥ 0
 (r[2] + λ * 0.5) - 1.0e-6 ≥ 0
 ((μ[2,7] - μ[2,6]) - (--1.0 * β_grp[1] - log(exp(r[2] + λ * 0.5) - 1.0)))
- 0.0 ≥ 0
 (r[2] + λ * 0.0) - 1.0e-6 ≥ 0
 ((μ[2,7] - μ[2,6]) - -(log(exp(r[2] + λ * 0.0) - 1.0))) - 0.0 ≥ 0
```

In [31]:

```julia
optimize!(model)

println("Termination status: ", termination_status(model))
println("Primal status: ", primal_status(model))
```

```
This is Ipopt version 3.14.19, running with linear solver MUMPS 5.8.1.

Number of nonzeros in equality constraint Jacobian...:        0
Number of nonzeros in inequality constraint Jacobian.:      119
Number of nonzeros in Lagrangian Hessian.............:       36

Total number of variables............................:       22
                     variables with only lower bounds:        1
                variables with lower and upper bounds:        0
                     variables with only upper bounds:        0
Total number of equality constraints.................:        0
Total number of inequality constraints...............:       40
        inequality constraints with only lower bounds:       38
   inequality constraints with lower and upper bounds:        0
        inequality constraints with only upper bounds:        2

iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_
pr  ls
   0  1.1000000e+00 0.00e+00 1.50e+00  -1.0 0.00e+00    -  0.00e+00 0.00e+
00   0
   1  1.0651615e+00 0.00e+00 2.94e-01  -1.0 4.35e-01  -4.0 7.71e-01 1.00e+
00f  1
   2  4.0938881e-01 1.55e-02 1.35e-01  -1.7 1.13e+00  -4.5 7.34e-01 1.00e+
00f  1
   3  3.0306360e-01 0.00e+00 1.32e-01  -1.7 2.21e+00    -  8.50e-01 1.00e+
00h  1
   4  4.2268735e-01 0.00e+00 2.47e-01  -1.7 3.95e+00  -5.0 8.50e-01 1.00e+
00h  1
   5  7.6514495e-01 0.00e+00 3.40e-01  -1.7 1.63e+01    -  6.27e-01 9.00e-
01h  1
   6  8.0582093e-01 0.00e+00 4.36e-01  -1.7 8.21e+00    -  1.00e+00 1.00e+
00f  1
   7  8.2313419e-01 0.00e+00 1.42e+00  -1.7 7.13e+00    -  1.00e+00 1.00e+
00h  1
   8  8.3929669e-01 0.00e+00 1.75e-03  -1.7 8.65e-01  -5.4 1.00e+00 1.00e+
00h  1
   9  2.0307814e-01 2.79e-01 1.15e-01  -3.8 3.73e+01    -  1.00e+00 8.08e-
01f  1
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_
pr  ls
  10  1.5454031e-01 1.50e-01 1.13e-01  -3.8 2.41e+00  -5.9 8.41e-01 5.17e-
01h  1
  11  1.3150971e-01 9.76e-02 2.37e-02  -3.8 9.38e-01  -6.4 8.52e-01 8.03e-
01h  1
  12  1.3316334e-01 0.00e+00 4.52e-03  -3.8 9.64e-02  -6.9 1.00e+00 1.00e+
00h  1
  13  1.3348251e-01 0.00e+00 3.02e-05  -3.8 4.44e-02  -7.3 1.00e+00 1.00e+
00h  1
  14  1.3117326e-01 2.24e-04 1.45e-04  -5.7 8.21e-02    -  9.87e-01 9.68e-
01h  1
  15  1.3117411e-01 0.00e+00 4.32e-07  -5.7 1.50e-03  -7.8 1.00e+00 1.00e+
00h  1
  16  1.3114650e-01 0.00e+00 2.54e-08  -8.6 9.18e-04  -8.3 1.00e+00 1.00e+
00h  1
  17  1.3114651e-01 0.00e+00 2.81e-12  -8.6 1.65e-03  -8.8 1.00e+00 1.00e+
00h  1

Number of Iterations....: 17

                                   (scaled)                 (unscaled)
```

```
Objective...............:   1.3114651484864803e-01    1.3114651484864803e-
01
Dual infeasibility......:   2.8143952447955705e-12    2.8143952447955705e-
12
Constraint violation....:   0.0000000000000000e+00    0.0000000000000000e+
00
Variable bound violation:   0.0000000000000000e+00    0.0000000000000000e+
00
Complementarity.........:   2.5071538951071849e-09    2.5071538951071849e-
09
Overall NLP error.......:   2.5071538951071849e-09    2.5071538951071849e-
09


Number of objective function evaluations             = 18
Number of objective gradient evaluations             = 18
Number of equality constraint evaluations            = 0
Number of inequality constraint evaluations          = 18
Number of equality constraint Jacobian evaluations   = 0
Number of inequality constraint Jacobian evaluations = 18
Number of Lagrangian Hessian evaluations             = 17
Total seconds in IPOPT                               = 0.010

EXIT: Optimal Solution Found.
Termination status: LOCALLY_SOLVED
Primal status: FEASIBLE_POINT
```

larger example

In [ ]:

```
using Random
using JuMP
using Ipopt

# -------------------------------------------------------------------
# If ZGEncodingInfo is already defined in your code, comment this out.
# -------------------------------------------------------------------
struct ZGEncodingInfo
    k_z::Vector{Int}     # number of categories for each categorical feat
    z_start::Vector{Int} # starting index of each block in β_z (1-based)
    num_g::Int           # number of groups
end

# Sigmoid
σ(t) = 1 / (1 + exp(-t))

"""
    linpred(β0, βx, βz, βgrp, x, z, g, encinfo)

Compute β0 + βxᵀ x + βzᵀ φ_z(z) + βgrpᵀ φ_g(g)
using the SAME reduced coding as in the DRO model.
"""
function linpred(
    β0::Real,
    βx::AbstractVector,
    βz::AbstractVector,
    βgrp::AbstractVector,
    x::AbstractVector,
    z::AbstractVector,
    g::Int,
```

```
    encinfo::ZGEncodingInfo,
)
    m     = length(encinfo.k_z)
    num_g = encinfo.num_g

    v = β0 + dot(βx, x)

    # categorical part
    for ℓ in 1:m
        kℓ    = encinfo.k_z[ℓ]
        start = encinfo.z_start[ℓ]      # 1-based index in βz
        c     = z[ℓ]                    # category ∈ {1,…,kℓ}

        if c < kℓ
            idx = start + (c - 1)       # reduced dummy: last level is re
            v += βz[idx]
        end
    end

    # group part: reduced dummy, last group as reference
    if g < num_g
        v += βgrp[g]
    end

    return v
end

"""
    generate_synthetic_instance(; N=300)

Generate a synthetic dataset for testing the DRO LR model.

Returns:
    X, Z, group, y, encinfo, β_true

β_true is a NamedTuple:
    (β0 = ..., βx = ..., βz = ..., βgrp = ...)
"""
function generate_synthetic_instance(; N::Int = 500)
    Random.seed!(2025)

    # --- dimensions ---
    n_x   = 2             # two numerical features
    k_z   = [3, 2]        # two categorical features: sizes 3 and 2
    m     = length(k_z)
    num_g = 3             # three groups

    p_z = sum(k_z .- 1)   # total length of βz (reduced dummy)
    p_g = num_g - 1       # length of βgrp

    # --- build encoding info for z ---
    z_start = zeros(Int, m)
    offset = 1
    for ℓ in 1:m
        z_start[ℓ] = offset
        offset += k_z[ℓ] - 1
    end
    encinfo = ZGEncodingInfo(k_z, z_start, num_g)

    # --- true parameters β∗ ---
```

```
    β0_true   = -0.3
    βx_true   = [1.0, -0.7]
    βz_true   = [0.8, -0.5, 0.6]   # length p_z = (3-1)+(2-1) = 3
    βgrp_true = [0.5, -0.4]        # length p_g = 2

    @assert length(βx_true)   == n_x
    @assert length(βz_true)   == p_z
    @assert length(βgrp_true) == p_g

    # --- sample features and labels ---
    X     = randn(N, n_x)
    Z     = zeros(Int, N, m)
    group = zeros(Int, N)
    y     = zeros(Int, N)

    for i in 1:N
        # categorical features
        for ℓ in 1:m
            Z[i,ℓ] = rand(1:k_z[ℓ])
        end

        # group index
        group[i] = rand(1:num_g)

        # linear predictor and label
        η = linpred(β0_true, βx_true, βz_true, βgrp_true,
                    view(X, i, :), view(Z, i, :), group[i], encinfo)
        p = σ(η)
        y[i] = rand() < p ? 1 : -1
    end

    β_true = (β0 = β0_true,
              βx = βx_true,
              βz = βz_true,
              βgrp = βgrp_true)

    return X, Z, group, y, encinfo, β_true
end

"""
    run_synthetic_experiment()

1. Generate synthetic data from a known β*.
2. Build the DRO LR graph-based model.
3. Solve it.
4. Print true vs estimated parameters.

Assumes you already defined `build_group_dro_graph_model`.
Also assumes that function returns `meta` containing JuMP variables
`β0`, `β_x`, `β_z`, `β_grp`.
"""
function run_synthetic_experiment()
    # 1. data
    X, Z, group, y, encinfo, β_true = generate_synthetic_instance(N = 300
    N, n_x = size(X)
    m      = size(Z, 2)
    num_g  = encinfo.num_g

    # 2. metric parameters (simple choice)
    delta   = 1e-2 * ones(m)             # δ_ℓ
```

```
    A_group = ones(num_g)              # A_g
    B_group = ones(num_g)              # B_g
    C_group = ones(num_g)              # C_g
    gamma_x = 1e-2 * ones(n_x)         # γ_j
    ε       = 1e-4

    # 3. build DRO model
    optimizer = optimizer_with_attributes(Ipopt.Optimizer,
                                          "print_level" => 5)

    model, meta = build_group_dro_graph_model(
        X, Z, group, y,
        encinfo,
        delta,
        A_group,
        B_group,
        C_group,
        gamma_x,
        ε,
        optimizer,
    )

    println("Model built. Start optimization...")
    optimize!(model)
    println("Termination status: ", termination_status(model))
    println("Objective value:    ", objective_value(model))

    # 4. extract parameters (adjust if your meta uses other field names)
    β0_hat   = value(meta.β0)
    βx_hat   = value.(meta.β_x)
    βz_hat   = value.(meta.β_z)
    βgrp_hat = value.(meta.β_grp)

    println("\n=== True vs estimated parameters ===")
    println("β0   true = ", β_true.β0,   "   hat = ", β0_hat)
    println("βx   true = ", β_true.βx,   "   hat = ", βx_hat)
    println("βz   true = ", β_true.βz,   "   hat = ", βz_hat)
    println("βgrp true = ", β_true.βgrp, "   hat = ", βgrp_hat)

    return model, meta, β_true, (β0_hat, βx_hat, βz_hat, βgrp_hat)
end

# -------------------------------------------------------------------
# Example usage from REPL / notebook:
#
# include("your_dro_graph_code.jl")   # defines build_group_dro_graph_mod
# include("this_synthetic_test.jl")   # this file
model, meta, β_true, β_hat = run_synthetic_experiment()
# -------------------------------------------------------------------
```

```
Model built. Start optimization...
This is Ipopt version 3.14.19, running with linear solver MUMPS 5.8.1.

Number of nonzeros in equality constraint Jacobian...:        0
Number of nonzeros in inequality constraint Jacobian.:    24908
Number of nonzeros in Lagrangian Hessian.............:     8100

Total number of variables............................:     2409
                     variables with only lower bounds:        1
                variables with lower and upper bounds:        0
                     variables with only upper bounds:        0
Total number of equality constraints.................:        0
Total number of inequality constraints...............:     7804
        inequality constraints with only lower bounds:     7800
   inequality constraints with lower and upper bounds:        0
        inequality constraints with only upper bounds:        4

iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_
pr  ls
   0  1.0001000e+00 0.00e+00 1.78e+00  -1.0 0.00e+00    -  0.00e+00 0.00e+
00   0
   1  1.0305435e+00 0.00e+00 9.75e+00  -1.0 4.20e-01  -4.0 7.92e-01 1.44e-
01f  1
   2  1.7004176e+00 0.00e+00 2.39e+01  -1.0 1.12e+00  -4.5 5.78e-01 1.00e+
00f  1
   3  3.3735854e+00 0.00e+00 7.52e+00  -1.0 2.12e+00  -5.0 4.88e-01 1.00e+
00f  1
   4  6.9104103e+00 0.00e+00 4.91e+00  -1.0 4.17e+00  -5.4 6.48e-01 1.00e+
00f  1
   5  1.6512566e+01 0.00e+00 3.96e+00  -1.0 1.17e+01  -5.9 5.38e-01 1.00e+
00f  1
   6  3.6127969e+01 0.00e+00 2.58e+00  -1.0 2.48e+01  -6.4 6.45e-01 1.00e+
00f  1
   7  8.4408130e+01 0.00e+00 1.32e+00  -1.0 6.67e+01  -6.9 5.36e-01 1.00e+
00f  1
   8  1.6307424e+02 0.00e+00 9.59e-02  -1.0 1.33e+02  -7.3 7.86e-01 1.00e+
00f  1
   9  1.1217965e+02 0.00e+00 8.61e-01  -1.7 1.44e+02  -7.8 3.08e-01 1.00e+
00f  1
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_
pr  ls
  10  8.1734959e+01 0.00e+00 1.42e-02  -1.7 2.27e+02  -8.3 7.64e-01 1.00e+
00f  1
  11  5.8149824e+01 0.00e+00 2.50e-02  -2.5 5.70e+01  -5.2 1.00e+00 7.69e-
01f  1
  12  3.6778207e+01 0.00e+00 7.31e-02  -2.5 5.36e+01  -5.6 1.00e+00 5.27e-
01f  1
  13  3.1838987e+01 0.00e+00 5.47e-02  -2.5 4.97e+01  -6.1 1.00e+00 2.93e-
01f  1
  14  2.6793668e+01 0.00e+00 1.76e-01  -2.5 3.64e+01  -6.6 1.00e+00 9.67e-
01h  1
  15  1.8768376e+01 0.00e+00 1.63e-01  -2.5 5.40e+01  -5.3 9.92e-01 6.80e-
01f  1
  16  1.8845623e+01 0.00e+00 1.65e-02  -2.5 4.41e+00  -3.9 9.56e-01 1.00e+
00h  1
  17  1.8830079e+01 0.00e+00 2.48e-02  -2.5 3.64e+00  -3.5 1.00e+00 1.00e+
00h  1
  18  1.8668280e+01 0.00e+00 8.36e-03  -2.5 1.03e+00  -4.0 9.75e-01 1.00e+
00h  1
  19  1.8106630e+01 0.00e+00 7.92e-03  -2.5 3.70e+00  -4.5 1.00e+00 1.00e+
```

```
00h  1
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_
pr  ls
  20  1.7883107e+01 0.00e+00 1.41e-02  -2.5 1.51e+00  -4.0 1.00e+00 1.00e+
00h  1
  21  1.4572626e+01 0.00e+00 1.58e-01  -3.8 1.41e+01  -4.5 1.00e+00 6.79e-
01f  1
  22  1.3626785e+01 0.00e+00 1.48e-01  -3.8 3.96e+00  -4.1 1.00e+00 7.05e-
01h  1
  23  1.3222876e+01 0.00e+00 5.60e-02  -3.8 1.36e+00  -3.7 1.00e+00 1.00e+
00h  1
  24  1.2780681e+01 0.00e+00 9.64e-02  -3.8 5.50e+00  -4.1 1.00e+00 3.97e-
01h  1
  25  1.2531740e+01 0.00e+00 3.82e-02  -3.8 1.23e+00  -3.7 1.00e+00 1.00e+
00h  1
  26  1.1807275e+01 0.00e+00 6.15e-02  -3.8 3.58e+00  -4.2 1.00e+00 1.00e+
00h  1
  27  1.1590172e+01 0.00e+00 2.30e-02  -3.8 9.82e-01  -3.8 1.00e+00 1.00e+
00h  1
  28  1.0864371e+01 2.37e-01 5.13e-02  -3.8 3.70e+00  -4.2 1.00e+00 1.00e+
00h  1
  29  1.0654291e+01 0.00e+00 2.01e-02  -3.8 1.04e+00  -3.8 1.00e+00 1.00e+
00h  1
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_
pr  ls
  30  1.0261122e+01 7.62e+00 2.80e+00  -3.8 3.47e+00  -4.3 6.84e-01 5.66e-
01h  1
  31  1.0225339e+01 6.65e+00 2.63e+00  -3.8 2.00e+00  -3.9 1.00e+00 1.77e-
01h  1
  32  9.8802066e+00 3.88e+00 1.52e+00  -3.8 3.88e+00  -4.3 2.69e-01 5.10e-
01h  1
  33  9.7281840e+00 1.90e+00 1.72e+00  -3.8 1.19e+00  -3.9 1.00e+00 7.71e-
01h  1
  34  9.3355361e+00 6.46e-01 6.74e-01  -3.8 3.63e+00  -4.4 8.53e-01 6.44e-
01h  1
  35  9.1683561e+00 1.05e-01 2.91e-01  -3.8 1.19e+00  -4.0 1.00e+00 1.00e+
00h  1
  36  8.5759191e+00 5.61e-02 1.42e-01  -3.8 7.49e+00  -4.4 1.00e+00 8.80e-
01h  1
  37  8.3955065e+00 8.69e-02 4.17e-02  -3.8 1.51e+00  -4.0 8.30e-01 9.05e-
01h  1
  38  8.3523064e+00 0.00e+00 2.34e-02  -3.8 1.28e+00  -3.6 5.67e-01 1.00e+
00h  1
  39  8.1671337e+00 0.00e+00 2.51e-02  -3.8 1.59e+00  -4.1 9.17e-01 1.00e+
00h  1
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_
pr  ls
  40  8.1085529e+00 0.00e+00 9.05e-03  -3.8 5.55e-01  -3.6 9.09e-01 1.00e+
00h  1
  41  7.9086498e+00 0.00e+00 2.97e-02  -3.8 1.83e+00  -4.1 1.00e+00 1.00e+
00h  1
  42  7.8451522e+00 0.00e+00 1.06e-02  -3.8 6.26e-01  -3.7 1.00e+00 1.00e+
00h  1
  43  7.6243309e+00 0.00e+00 3.48e-02  -3.8 2.11e+00  -4.2 1.00e+00 1.00e+
00h  1
  44  7.5548220e+00 0.00e+00 1.23e-02  -3.8 7.05e-01  -3.7 1.00e+00 1.00e+
00h  1
  45  7.3057491e+00 0.00e+00 4.07e-02  -3.8 2.42e+00  -4.2 1.00e+00 1.00e+
00h  1
  46  7.2293064e+00 0.00e+00 1.42e-02  -3.8 7.88e-01  -3.8 1.00e+00 1.00e+
```

```
00h  1
  47  6.9465335e+00 0.00e+00 4.70e-02  -3.8 2.76e+00  -4.3 1.00e+00 1.00e+
00h  1
  48  6.8629700e+00 0.00e+00 1.60e-02  -3.8 8.73e-01  -3.8 1.00e+00 1.00e+
00h  1
  49  6.5437913e+00 3.56e-01 5.30e-02  -3.8 3.14e+00  -4.3 1.00e+00 1.00e+
00h  1
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_
pr  ls
  50  6.4545232e+00 0.00e+00 1.72e-02  -3.8 9.60e-01  -3.9 1.00e+00 1.00e+
00h  1
  51  6.1887297e+00 7.60e+00 2.56e+00  -3.8 3.49e+00  -4.4 9.65e-01 7.64e-
01h  1
  52  6.1827010e+00 7.31e+00 2.45e+00  -3.8 2.73e+00  -3.9 1.00e+00 6.32e-
02h  1
  53  5.9255068e+00 7.70e+00 5.57e+00  -3.8 4.72e+00  -4.4 2.50e-01 5.81e-
01h  1
  54  5.8968827e+00 6.35e+00 5.15e+00  -3.8 1.26e+00  -4.0 1.00e+00 2.76e-
01h  1
  55  5.6487756e+00 3.58e+00 3.42e+00  -3.8 6.11e+00  -4.5 3.22e-01 5.03e-
01h  1
  56  5.5610480e+00 1.85e+00 3.15e+00  -3.8 1.49e+00  -4.1 5.84e-01 8.12e-
01h  1
  57  5.2685068e+00 5.89e-01 1.49e+00  -3.8 5.44e+00  -4.5 8.15e-01 6.98e-
01h  1
  58  5.1778464e+00 3.10e-02 4.38e-01  -3.8 3.97e+00  -4.1 2.63e-01 1.00e+
00h  1
  59  4.8141398e+00 5.40e-02 3.22e-01  -3.8 5.19e+00  -4.6 7.95e-01 1.00e+
00h  1
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_
pr  ls
  60  4.8188893e+00 0.00e+00 1.87e-01  -3.8 8.22e-01  -2.3 1.00e+00 1.00e+
00h  1
  61  4.8196510e+00 0.00e+00 1.03e-01  -3.8 6.04e-01  -2.8 6.26e-01 1.00e+
00h  1
  62  4.8196283e+00 0.00e+00 7.37e-02  -3.8 8.24e-01  -2.4 5.91e-01 1.00e+
00h  1
  63  4.8197070e+00 0.00e+00 3.46e-02  -3.8 2.90e-01  -2.0 1.00e+00 1.00e+
00h  1
  64  4.8200865e+00 0.00e+00 4.08e-02  -3.8 2.34e+00  -2.5 2.51e-01 1.00e+
00h  1
  65  4.8210194e+00 0.00e+00 1.95e-02  -3.8 1.08e+00  -2.0 1.00e+00 1.00e+
00h  1
  66  4.8194361e+00 0.00e+00 1.07e-03  -3.8 1.40e-01  -2.5 1.00e+00 1.00e+
00h  1
  67  4.8135455e+00 0.00e+00 5.42e-04  -3.8 1.91e-01  -3.0 1.00e+00 1.00e+
00h  1
  68  4.7950527e+00 0.00e+00 2.13e-03  -3.8 6.12e-01  -3.5 1.00e+00 1.00e+
00h  1
  69  4.7415022e+00 0.00e+00 6.11e-03  -3.8 8.42e-01  -3.9 1.00e+00 1.00e+
00h  1
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_
pr  ls
  70  4.5646365e+00 0.00e+00 2.04e-02  -3.8 2.65e+00  -4.4 1.00e+00 1.00e+
00h  1
  71  4.5118938e+00 0.00e+00 7.11e-03  -3.8 8.77e-01  -4.0 1.00e+00 1.00e+
00h  1
  72  4.3263790e+00 0.00e+00 2.10e-02  -3.8 2.77e+00  -4.5 1.00e+00 1.00e+
00h  1
  73  4.2713646e+00 0.00e+00 7.30e-03  -3.8 9.04e-01  -4.0 1.00e+00 1.00e+
```

```
00h  1
  74  4.0758659e+00 0.00e+00 2.17e-02  -3.8 2.91e+00  -4.5 1.00e+00 1.00e+
00h  1
  75  4.0181704e+00 0.00e+00 7.51e-03  -3.8 9.32e-01  -4.1 1.00e+00 1.00e+
00h  1
  76  3.8928050e+00 0.00e+00 1.67e-02  -3.8 3.15e+00  -4.6 1.00e+00 6.02e-
01h  1
  77  3.8290007e+00 0.00e+00 7.52e-03  -3.8 9.90e-01  -4.1 8.24e-01 1.00e+
00f  1
  78  3.7606327e+00 0.00e+00 1.27e-02  -3.8 3.93e+00  -4.6 1.00e+00 2.93e-
01h  1
  79  3.6895109e+00 0.00e+00 7.82e-03  -3.8 1.08e+00  -4.2 6.61e-01 1.00e+
00f  1
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_
pr  ls
  80  3.6524310e+00 0.00e+00 1.09e-02  -3.8 5.28e+00  -4.7 1.00e+00 1.37e-
01h  1
  81  3.5726715e+00 0.00e+00 8.44e-03  -3.8 1.18e+00  -4.2 5.85e-01 1.00e+
00f  1
  82  3.5514297e+00 0.00e+00 1.04e-02  -3.8 7.56e+00  -4.7 1.00e+00 6.64e-
02h  1
  83  3.4615955e+00 0.00e+00 9.35e-03  -3.8 1.31e+00  -4.3 5.53e-01 1.00e+
00f  1
  84  3.4494047e+00 0.00e+00 1.06e-02  -3.8 1.20e+01  -4.8 9.05e-01 3.10e-
02h  1
  85  3.3477148e+00 0.00e+00 1.05e-02  -3.8 1.47e+00  -4.3 5.46e-01 1.00e+
00f  1
  86  3.3408134e+00 0.00e+00 1.15e-02  -3.8 2.40e+01  -4.8 3.16e-01 1.29e-
02h  1
  87  3.2221593e+00 0.00e+00 1.17e-02  -3.8 1.84e+00  -4.4 1.00e+00 1.00e+
00f  1
  88  3.1837293e+00 0.00e+00 1.63e-02  -3.8 2.71e+01  -4.9 3.59e-01 4.92e-
02h  1
  89  3.0522335e+00 0.00e+00 1.52e-02  -3.8 1.90e+00  -4.4 9.69e-01 1.00e+
00h  1
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_
pr  ls
  90  3.0043228e+00 0.00e+00 3.75e-03  -3.8 9.68e-01  -4.0 1.00e+00 1.00e+
00h  1
  91  2.8609905e+00 0.00e+00 1.30e-02  -3.8 2.09e+00  -4.5 1.00e+00 1.00e+
00h  1
  92  2.8137790e+00 0.00e+00 4.71e-03  -3.8 7.30e-01  -4.1 1.00e+00 1.00e+
00h  1
  93  2.6515251e+00 0.00e+00 1.50e-02  -3.8 2.46e+00  -4.5 1.00e+00 1.00e+
00h  1
  94  2.5983388e+00 0.00e+00 5.38e-03  -3.8 7.87e-01  -4.1 1.00e+00 1.00e+
00h  1
  95  2.4125599e+00 0.00e+00 1.73e-02  -3.8 3.02e+00  -4.6 1.00e+00 1.00e+
00h  1
  96  2.3524139e+00 0.00e+00 6.21e-03  -3.8 8.84e-01  -4.2 1.00e+00 1.00e+
00h  1
  97  2.1374244e+00 0.00e+00 2.04e-02  -3.8 3.73e+00  -4.6 1.00e+00 1.00e+
00h  1
  98  2.0688179e+00 0.00e+00 7.26e-03  -3.8 1.04e+00  -4.2 1.00e+00 1.00e+
00h  1
  99  1.9155207e+00 0.00e+00 1.77e-02  -3.8 4.55e+00  -4.7 1.00e+00 6.10e-
01h  1
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_
pr  ls
 100  1.8389588e+00 0.00e+00 7.97e-03  -3.8 1.11e+00  -4.3 5.23e-01 1.00e+
```

```
00h  1
 101  1.7768936e+00 0.00e+00 1.24e-02  -3.8 5.47e+00  -4.8 1.00e+00 2.12e-
01h  1
 102  1.6911970e+00 0.00e+00 8.11e-03  -3.8 1.25e+00  -4.3 3.95e-01 1.00e+
00f  1
 103  1.5414948e+00 2.17e+00 2.38e-02  -3.8 5.48e+00  -4.8 1.00e+00 4.50e-
01h  1
 104  1.4506536e+00 4.67e-01 1.46e-02  -3.8 1.42e+00  -4.4 2.42e-01 1.00e+
00h  1
 105  1.2153165e+00 4.01e-01 2.04e-02  -3.8 6.69e+00  -4.9 5.84e-01 6.85e-
01h  1
 106  1.1295937e+00 0.00e+00 7.50e-03  -3.8 3.68e+00  -4.4 3.09e-01 1.00e+
00h  1
 107  1.0086186e+00 5.64e-02 1.20e-02  -3.8 5.54e+00  -4.9 6.33e-01 4.52e-
01h  1
 108  8.3160330e-01 5.27e-01 2.17e-02  -3.8 1.81e+01  -5.4 3.27e-01 2.59e-
01h  1
 109  7.7494902e-01 1.19e-01 6.44e-03  -3.8 2.92e+00  -5.0 1.00e+00 1.00e+
00h  1
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_
pr  ls
 110  7.6980750e-01 0.00e+00 8.82e-04  -3.8 2.26e+00  -5.4 1.00e+00 1.00e+
00h  1
 111  7.7147083e-01 0.00e+00 6.24e-05  -3.8 2.05e+00  -5.9 1.00e+00 1.00e+
00h  1
 112  7.7124884e-01 0.00e+00 4.88e-04  -3.8 1.97e+00  -6.4 1.00e+00 1.00e+
00h  1
 113  6.0508542e-01 7.64e-02 5.92e-04  -5.7 2.93e+00  -6.9 7.85e-01 7.87e-
01f  1
 114  5.7902958e-01 1.89e-03 1.22e-04  -5.7 7.57e-01  -7.3 9.88e-01 1.00e+
00h  1
 115  5.7806101e-01 0.00e+00 1.13e-04  -5.7 3.42e+00  -7.8 1.00e+00 1.00e+
00h  1
 116  5.7789302e-01 0.00e+00 1.52e-04  -5.7 1.56e+00  -7.4 1.00e+00 1.00e+
00h  1
 117  5.7786996e-01 0.00e+00 2.76e-05  -5.7 3.03e-01  -7.0 1.00e+00 1.00e+
00h  1
 118  5.7777080e-01 0.00e+00 6.11e-05  -5.7 1.02e+00  -7.4 1.00e+00 1.00e+
00h  1
 119  5.7774783e-01 0.00e+00 1.43e-05  -5.7 2.82e-01  -7.0 1.00e+00 1.00e+
00h  1
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_
pr  ls
 120  5.7540466e-01 1.64e-03 5.67e-05  -8.6 1.44e+00  -7.5 9.47e-01 9.97e-
01h  1
 121  5.7535308e-01 0.00e+00 2.29e-05  -8.6 4.63e-01  -7.1 8.84e-01 1.00e+
00h  1
 122  5.7518739e-01 0.00e+00 4.85e-05  -8.6 1.69e+00  -7.5 1.00e+00 1.00e+
00h  1
 123  5.7483182e-01 0.00e+00 1.10e-04  -8.6 1.20e+01  -8.0 3.34e-01 3.02e-
01h  1
 124  5.7464428e-01 0.00e+00 1.06e-04  -8.6 1.73e+01  -8.5 4.13e-01 1.11e-
01h  1
 125  5.7407040e-01 0.00e+00 8.44e-05  -8.6 2.78e+01  -9.0 4.59e-01 2.11e-
01f  1
 126  5.7238571e-01 0.00e+00 6.61e-05  -8.6 7.92e+01  -9.5 7.72e-01 2.17e-
01h  1
 127  5.4992563e-01 0.00e+00 2.24e-06  -8.6 2.37e+02  -9.9 1.00e+00 9.66e-
01h  1
 128  5.4926259e-01 2.55e-02 2.22e-06  -8.6 7.08e+02 -10.4 1.00e+00 9.56e-
```

```
03h  1
 129  5.4855453e-01 3.34e-02 1.70e-03  -8.6 7.55e+00 -10.9 8.29e-01 1.00e+
00h  1
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_
pr  ls
 130  5.4870612e-01 1.65e-02 1.31e-03  -8.6 1.98e+00 -11.4 2.34e-01 5.10e-
01h  1
 131  5.4879600e-01 7.18e-03 2.24e-04  -8.6 9.56e-01 -11.8 8.26e-01 5.64e-
01h  1
 132  5.4886485e-01 5.26e-04 4.87e-05  -8.6 2.34e+00 -11.4 7.85e-01 9.24e-
01h  1
 133  5.4887077e-01 0.00e+00 3.74e-10  -8.6 1.26e+00 -11.9 1.00e+00 9.99e-
01h  1

Number of Iterations....: 133

                                   (scaled)                 (unscaled)
Objective...............:   5.4887076556059111e-01    5.4887076556059111e-
01
Dual infeasibility......:   3.7365899622253190e-10    3.7365899622253190e-
10
Constraint violation....:   0.0000000000000000e+00    0.0000000000000000e+
00
Variable bound violation:   0.0000000000000000e+00    0.0000000000000000e+
00
Complementarity.........:   3.2711975698061975e-09    3.2711975698061975e-
09
Overall NLP error.......:   3.2711975698061975e-09    3.2711975698061975e-
09


Number of objective function evaluations             = 134
Number of objective gradient evaluations             = 134
Number of equality constraint evaluations            = 0
Number of inequality constraint evaluations          = 134
Number of equality constraint Jacobian evaluations   = 0
Number of inequality constraint Jacobian evaluations = 134
Number of Lagrangian Hessian evaluations             = 133
Total seconds in IPOPT                               = 6.498

EXIT: Optimal Solution Found.
Termination status: LOCALLY_SOLVED
Objective value:    0.5488707655605911

=== True vs estimated parameters ===
β0   true = -0.3   hat = -0.26966141856599213
βx   true = [1.0, -0.7]   hat = [0.7268207209228954, -0.8371227622608812]
βz   true = [0.8, -0.5, 0.6]   hat = [0.7977649681015079, -0.1533352980175
6676, 0.5879875082314706]
βgrp true = [0.5, -0.4]   hat = [0.8779178610381262, -0.6205919690109933]
```

```
(A JuMP Model
├ solver: Ipopt
├ objective_sense: MIN_SENSE
│ └ objective_function_type: AffExpr
├ num_variables: 2409
├ num_constraints: 7805
│ ├ AffExpr in MOI.GreaterThan{Float64}: 2400
│ ├ AffExpr in MOI.LessThan{Float64}: 4
│ ├ VariableRef in MOI.GreaterThan{Float64}: 1
│ └ Nonlinear: 5400
└ Names registered in the model
  └ :r, :β0, :β_grp, :β_x, :β_z, :λ, :μ, (encinfo = ZGEncodingInfo([3, 2],
[1, 3], 3), dags = SampleDAG[SampleDAG(1, [(0, 0.0), (1, 0.01), (1, 0.0),
(2, 0.01), (2, 0.02), (2, 0.0), (3, 0.0)], Arc[Arc(1, 2, CatArc(1, 1, 0.0,
0.01)), Arc(1, 2, CatArc(1, 2, 0.0, 0.01)), Arc(1, 3, CatArc(1, 3, 0.0, 0.
0)), Arc(2, 4, CatArc(2, 1, 0.01, 0.01)), Arc(2, 5, CatArc(2, 2, 0.01, 0.0
2)), Arc(3, 6, CatArc(2, 1, 0.0, 0.0)), Arc(3, 4, CatArc(2, 2, 0.0, 0.0
1)), Arc(4, 7, TermArc(0.01, 1)), Arc(4, 7, TermArc(0.01, 2)), Arc(4, 7, T
ermArc(0.01, 3)), Arc(5, 7, TermArc(0.02, 1)), Arc(5, 7, TermArc(0.02,
2)), Arc(5, 7, TermArc(0.02, 3)), Arc(6, 7, TermArc(0.0, 1)), Arc(6, 7, Te
rmArc(0.0, 2)), Arc(6, 7, TermArc(0.0, 3))], 1, 7), SampleDAG(2, [(0, 0.
0), (1, 0.01), (1, 0.0), (2, 0.01), (2, 0.02), (2, 0.0), (3, 0.0)], Arc[Ar
c(1, 2, CatArc(1, 1, 0.0, 0.01)), Arc(1, 2, CatArc(1, 2, 0.0, 0.01)), Arc
(1, 3, CatArc(1, 3, 0.0, 0.0)), Arc(2, 4, CatArc(2, 1, 0.01, 0.01)), Arc
(2, 5, CatArc(2, 2, 0.01, 0.02)), Arc(3, 6, CatArc(2, 1, 0.0, 0.0)), Arc
(3, 4, CatArc(2, 2, 0.0, 0.01)), Arc(4, 7, TermArc(0.01, 1)), Arc(4, 7, Te
rmArc(0.01, 2)), Arc(4, 7, TermArc(0.01, 3)), Arc(5, 7, TermArc(0.02, 1)),
Arc(5, 7, TermArc(0.02, 2)), Arc(5, 7, TermArc(0.02, 3)), Arc(6, 7, TermAr
c(0.0, 1)), Arc(6, 7, TermArc(0.0, 2)), Arc(6, 7, TermArc(0.0, 3))], 1,
7), SampleDAG(3, [(0, 0.0), (1, 0.01), (1, 0.0), (2, 0.01), (2, 0.02), (2,
0.0), (3, 0.0)], Arc[Arc(1, 2, CatArc(1, 1, 0.0, 0.01)), Arc(1, 2, CatArc
(1, 2, 0.0, 0.01)), Arc(1, 3, CatArc(1, 3, 0.0, 0.0)), Arc(2, 4, CatArc(2,
1, 0.01, 0.01)), Arc(2, 5, CatArc(2, 2, 0.01, 0.02)), Arc(3, 6, CatArc(2,
1, 0.0, 0.0)), Arc(3, 4, CatArc(2, 2, 0.0, 0.01)), Arc(4, 7, TermArc(0.01,
1)), Arc(4, 7, TermArc(0.01, 2)), Arc(4, 7, TermArc(0.01, 3)), Arc(5, 7, T
ermArc(0.02, 1)), Arc(5, 7, TermArc(0.02, 2)), Arc(5, 7, TermArc(0.02,
3)), Arc(6, 7, TermArc(0.0, 1)), Arc(6, 7, TermArc(0.0, 2)), Arc(6, 7, Ter
mArc(0.0, 3))], 1, 7), SampleDAG(4, [(0, 0.0), (1, 0.0), (1, 0.01), (2, 0.
01), (2, 0.0), (2, 0.02), (3, 0.0)], Arc[Arc(1, 2, CatArc(1, 1, 0.0, 0.
0)), Arc(1, 3, CatArc(1, 2, 0.0, 0.01)), Arc(1, 3, CatArc(1, 3, 0.0, 0.0
1)), Arc(2, 4, CatArc(2, 1, 0.0, 0.01)), Arc(2, 5, CatArc(2, 2, 0.0, 0.
0)), Arc(3, 6, CatArc(2, 1, 0.01, 0.02)), Arc(3, 4, CatArc(2, 2, 0.01, 0.0
1)), Arc(4, 7, TermArc(0.01, 1)), Arc(4, 7, TermArc(0.01, 2)), Arc(4, 7, T
ermArc(0.01, 3)), Arc(5, 7, TermArc(0.0, 1)), Arc(5, 7, TermArc(0.0, 2)),
Arc(5, 7, TermArc(0.0, 3)), Arc(6, 7, TermArc(0.02, 1)), Arc(6, 7, TermArc
(0.02, 2)), Arc(6, 7, TermArc(0.02, 3))], 1, 7), SampleDAG(5, [(0, 0.0),
(1, 0.01), (1, 0.0), (2, 0.02), (2, 0.01), (2, 0.0), (3, 0.0)], Arc[Arc(1,
2, CatArc(1, 1, 0.0, 0.01)), Arc(1, 2, CatArc(1, 2, 0.0, 0.01)), Arc(1, 3,
CatArc(1, 3, 0.0, 0.0)), Arc(2, 4, CatArc(2, 1, 0.01, 0.02)), Arc(2, 5, Ca
tArc(2, 2, 0.01, 0.01)), Arc(3, 5, CatArc(2, 1, 0.0, 0.01)), Arc(3, 6, Cat
Arc(2, 2, 0.0, 0.0)), Arc(4, 7, TermArc(0.02, 1)), Arc(4, 7, TermArc(0.02,
2)), Arc(4, 7, TermArc(0.02, 3)), Arc(5, 7, TermArc(0.01, 1)), Arc(5, 7, T
ermArc(0.01, 2)), Arc(5, 7, TermArc(0.01, 3)), Arc(6, 7, TermArc(0.0, 1)),
Arc(6, 7, TermArc(0.0, 2)), Arc(6, 7, TermArc(0.0, 3))], 1, 7), SampleDAG
(6, [(0, 0.0), (1, 0.01), (1, 0.0), (2, 0.02), (2, 0.01), (2, 0.0), (3, 0.
0)], Arc[Arc(1, 2, CatArc(1, 1, 0.0, 0.01)), Arc(1, 3, CatArc(1, 2, 0.0,
0.0)), Arc(1, 2, CatArc(1, 3, 0.0, 0.01)), Arc(2, 4, CatArc(2, 1, 0.01, 0.
02)), Arc(2, 5, CatArc(2, 2, 0.01, 0.01)), Arc(3, 5, CatArc(2, 1, 0.0, 0.0
1)), Arc(3, 6, CatArc(2, 2, 0.0, 0.0)), Arc(4, 7, TermArc(0.02, 1)), Arc
(4, 7, TermArc(0.02, 2)), Arc(4, 7, TermArc(0.02, 3)), Arc(5, 7, TermArc
```

(0.01, 1)), Arc(5, 7, TermArc(0.01, 2)), Arc(5, 7, TermArc(0.01, 3)), Arc(6, 7, TermArc(0.0, 1)), Arc(6, 7, TermArc(0.0, 2)), Arc(6, 7, TermArc(0.0, 3))], 1, 7), SampleDAG(7, [(0, 0.0), (1, 0.01), (1, 0.0), (2, 0.02), (2, 0.01), (2, 0.0), (3, 0.0)], Arc[Arc(1, 2, CatArc(1, 1, 0.0, 0.01)), Arc(1, 3, CatArc(1, 2, 0.0, 0.0)), Arc(1, 2, CatArc(1, 3, 0.0, 0.01)), Arc(2, 4, CatArc(2, 1, 0.01, 0.02)), Arc(2, 5, CatArc(2, 2, 0.01, 0.01)), Arc(3, 5, CatArc(2, 1, 0.0, 0.01)), Arc(3, 6, CatArc(2, 2, 0.0, 0.0)), Arc(4, 7, TermArc(0.02, 1)), Arc(4, 7, TermArc(0.02, 2)), Arc(4, 7, TermArc(0.02, 3)), Arc(5, 7, TermArc(0.01, 1)), Arc(5, 7, TermArc(0.01, 2)), Arc(5, 7, TermArc(0.01, 3)), Arc(6, 7, TermArc(0.0, 1)), Arc(6, 7, TermArc(0.0, 2)), Arc(6, 7, TermArc(0.0, 3))], 1, 7), SampleDAG(8, [(0, 0.0), (1, 0.01), (1, 0.0), (2, 0.02), (2, 0.01), (2, 0.0), (3, 0.0)], Arc[Arc(1, 2, CatArc(1, 1, 0.0, 0.01)), Arc(1, 3, CatArc(1, 2, 0.0, 0.0)), Arc(1, 2, CatArc(1, 3, 0.0, 0.01)), Arc(2, 4, CatArc(2, 1, 0.01, 0.02)), Arc(2, 5, CatArc(2, 2, 0.01, 0.01)), Arc(3, 5, CatArc(2, 1, 0.0, 0.01)), Arc(3, 6, CatArc(2, 2, 0.0, 0.0)), Arc(4, 7, TermArc(0.02, 1)), Arc(4, 7, TermArc(0.02, 2)), Arc(4, 7, TermArc(0.02, 3)), Arc(5, 7, TermArc(0.01, 1)), Arc(5, 7, TermArc(0.01, 2)), Arc(5, 7, TermArc(0.01, 3)), Arc(6, 7, TermArc(0.0, 1)), Arc(6, 7, TermArc(0.0, 2)), Arc(6, 7, TermArc(0.0, 3))], 1, 7), SampleDAG(9, [(0, 0.0), (1, 0.0), (1, 0.01), (2, 0.01), (2, 0.0), (2, 0.02), (3, 0.0)], Arc[Arc(1, 2, CatArc(1, 1, 0.0, 0.0)), Arc(1, 3, CatArc(1, 2, 0.0, 0.01)), Arc(1, 3, CatArc(1, 3, 0.0, 0.01)), Arc(2, 4, CatArc(2, 1, 0.0, 0.01)), Arc(2, 5, CatArc(2, 2, 0.0, 0.0)), Arc(3, 6, CatArc(2, 1, 0.01, 0.02)), Arc(3, 4, CatArc(2, 2, 0.01, 0.01)), Arc(4, 7, TermArc(0.01, 1)), Arc(4, 7, TermArc(0.01, 2)), Arc(4, 7, TermArc(0.01, 3)), Arc(5, 7, TermArc(0.0, 1)), Arc(5, 7, TermArc(0.0, 2)), Arc(5, 7, TermArc(0.0, 3)), Arc(6, 7, TermArc(0.02, 1)), Arc(6, 7, TermArc(0.02, 2)), Arc(6, 7, TermArc(0.02, 3))], 1, 7), SampleDAG(10, [(0, 0.0), (1, 0.0), (1, 0.01), (2, 0.01), (2, 0.0), (2, 0.02), (3, 0.0)], Arc[Arc(1, 2, CatArc(1, 1, 0.0, 0.0)), Arc(1, 3, CatArc(1, 2, 0.0, 0.01)), Arc(1, 3, CatArc(1, 3, 0.0, 0.01)), Arc(2, 4, CatArc(2, 1, 0.0, 0.01)), Arc(2, 5, CatArc(2, 2, 0.0, 0.0)), Arc(3, 6, CatArc(2, 1, 0.01, 0.02)), Arc(3, 4, CatArc(2, 2, 0.01, 0.01)), Arc(4, 7, TermArc(0.01, 1)), Arc(4, 7, TermArc(0.01, 2)), Arc(4, 7, TermArc(0.01, 3)), Arc(5, 7, TermArc(0.0, 1)), Arc(5, 7, TermArc(0.0, 2)), Arc(5, 7, TermArc(0.0, 3)), Arc(6, 7, TermArc(0.02, 1)), Arc(6, 7, TermArc(0.02, 2)), Arc(6, 7, TermArc(0.02, 3))], 1, 7) … SampleDAG(291, [(0, 0.0), (1, 0.01), (1, 0.0), (2, 0.02), (2, 0.01), (2, 0.0), (3, 0.0)], Arc[Arc(1, 2, CatArc(1, 1, 0.0, 0.01)), Arc(1, 2, CatArc(1, 2, 0.0, 0.01)), Arc(1, 3, CatArc(1, 3, 0.0, 0.0)), Arc(2, 4, CatArc(2, 1, 0.01, 0.02)), Arc(2, 5, CatArc(2, 2, 0.01, 0.01)), Arc(3, 5, CatArc(2, 1, 0.0, 0.01)), Arc(3, 6, CatArc(2, 2, 0.0, 0.0)), Arc(4, 7, TermArc(0.02, 1)), Arc(4, 7, TermArc(0.02, 2)), Arc(4, 7, TermArc(0.02, 3)), Arc(5, 7, TermArc(0.01, 1)), Arc(5, 7, TermArc(0.01, 2)), Arc(5, 7, TermArc(0.01, 3)), Arc(6, 7, TermArc(0.0, 1)), Arc(6, 7, TermArc(0.0, 2)), Arc(6, 7, TermArc(0.0, 3))], 1, 7), SampleDAG(292, [(0, 0.0), (1, 0.0), (1, 0.01), (2, 0.0), (2, 0.01), (2, 0.02), (3, 0.0)], Arc[Arc(1, 2, CatArc(1, 1, 0.0, 0.0)), Arc(1, 3, CatArc(1, 2, 0.0, 0.01)), Arc(1, 3, CatArc(1, 3, 0.0, 0.01)), Arc(2, 4, CatArc(2, 1, 0.0, 0.0)), Arc(2, 5, CatArc(2, 2, 0.0, 0.01)), Arc(3, 5, CatArc(2, 1, 0.01, 0.01)), Arc(3, 6, CatArc(2, 2, 0.01, 0.02)), Arc(4, 7, TermArc(0.0, 1)), Arc(4, 7, TermArc(0.0, 2)), Arc(4, 7, TermArc(0.0, 3)), Arc(5, 7, TermArc(0.01, 1)), Arc(5, 7, TermArc(0.01, 2)), Arc(5, 7, TermArc(0.01, 3)), Arc(6, 7, TermArc(0.02, 1)), Arc(6, 7, TermArc(0.02, 2)), Arc(6, 7, TermArc(0.02, 3))], 1, 7), SampleDAG(293, [(0, 0.0), (1, 0.01), (1, 0.0), (2, 0.02), (2, 0.01), (2, 0.0), (3, 0.0)], Arc[Arc(1, 2, CatArc(1, 1, 0.0, 0.01)), Arc(1, 3, CatArc(1, 2, 0.0, 0.0)), Arc(1, 2, CatArc(1, 3, 0.0, 0.01)), Arc(2, 4, CatArc(2, 1, 0.01, 0.02)), Arc(2, 5, CatArc(2, 2, 0.01, 0.01)), Arc(3, 5, CatArc(2, 1, 0.0, 0.01)), Arc(3, 6, CatArc(2, 2, 0.0, 0.0)), Arc(4, 7, TermArc(0.02, 1)), Arc(4, 7, TermArc(0.02, 2)), Arc(4, 7, TermArc(0.02, 3)), Arc(5, 7, TermArc(0.01, 1)), Arc(5, 7, TermArc(0.01, 2)), Arc(5, 7, TermArc(0.01, 3)), Arc(6, 7, TermArc(0.0, 1)), Arc(6, 7, TermArc(0.0, 2)), Arc(6,

```
7, TermArc(0.0, 3))], 1, 7), SampleDAG(294, [(0, 0.0), (1, 0.0), (1, 0.0
1), (2, 0.0), (2, 0.01), (2, 0.02), (3, 0.0)], Arc[Arc(1, 2, CatArc(1, 1,
0.0, 0.0)), Arc(1, 3, CatArc(1, 2, 0.0, 0.01)), Arc(1, 3, CatArc(1, 3, 0.
0, 0.01)), Arc(2, 4, CatArc(2, 1, 0.0, 0.0)), Arc(2, 5, CatArc(2, 2, 0.0,
0.01)), Arc(3, 5, CatArc(2, 1, 0.01, 0.01)), Arc(3, 6, CatArc(2, 2, 0.01,
0.02)), Arc(4, 7, TermArc(0.0, 1)), Arc(4, 7, TermArc(0.0, 2)), Arc(4, 7,
TermArc(0.0, 3)), Arc(5, 7, TermArc(0.01, 1)), Arc(5, 7, TermArc(0.01,
2)), Arc(5, 7, TermArc(0.01, 3)), Arc(6, 7, TermArc(0.02, 1)), Arc(6, 7, T
ermArc(0.02, 2)), Arc(6, 7, TermArc(0.02, 3))], 1, 7), SampleDAG(295, [(0,
0.0), (1, 0.0), (1, 0.01), (2, 0.01), (2, 0.0), (2, 0.02), (3, 0.0)], Arc
[Arc(1, 2, CatArc(1, 1, 0.0, 0.0)), Arc(1, 3, CatArc(1, 2, 0.0, 0.01)), Ar
c(1, 3, CatArc(1, 3, 0.0, 0.01)), Arc(2, 4, CatArc(2, 1, 0.0, 0.01)), Arc
(2, 5, CatArc(2, 2, 0.0, 0.0)), Arc(3, 6, CatArc(2, 1, 0.01, 0.02)), Arc
(3, 4, CatArc(2, 2, 0.01, 0.01)), Arc(4, 7, TermArc(0.01, 1)), Arc(4, 7, T
ermArc(0.01, 2)), Arc(4, 7, TermArc(0.01, 3)), Arc(5, 7, TermArc(0.0, 1)),
Arc(5, 7, TermArc(0.0, 2)), Arc(5, 7, TermArc(0.0, 3)), Arc(6, 7, TermArc
(0.02, 1)), Arc(6, 7, TermArc(0.02, 2)), Arc(6, 7, TermArc(0.02, 3))], 1,
7), SampleDAG(296, [(0, 0.0), (1, 0.01), (1, 0.0), (2, 0.01), (2, 0.02),
(2, 0.0), (3, 0.0)], Arc[Arc(1, 2, CatArc(1, 1, 0.0, 0.01)), Arc(1, 2, Cat
Arc(1, 2, 0.0, 0.01)), Arc(1, 3, CatArc(1, 3, 0.0, 0.0)), Arc(2, 4, CatArc
(2, 1, 0.01, 0.01)), Arc(2, 5, CatArc(2, 2, 0.01, 0.02)), Arc(3, 6, CatArc
(2, 1, 0.0, 0.0)), Arc(3, 4, CatArc(2, 2, 0.0, 0.01)), Arc(4, 7, TermArc
(0.01, 1)), Arc(4, 7, TermArc(0.01, 2)), Arc(4, 7, TermArc(0.01, 3)), Arc
(5, 7, TermArc(0.02, 1)), Arc(5, 7, TermArc(0.02, 2)), Arc(5, 7, TermArc
(0.02, 3)), Arc(6, 7, TermArc(0.0, 1)), Arc(6, 7, TermArc(0.0, 2)), Arc(6,
7, TermArc(0.0, 3))], 1, 7), SampleDAG(297, [(0, 0.0), (1, 0.01), (1, 0.
0), (2, 0.02), (2, 0.01), (2, 0.0), (3, 0.0)], Arc[Arc(1, 2, CatArc(1, 1,
0.0, 0.01)), Arc(1, 2, CatArc(1, 2, 0.0, 0.01)), Arc(1, 3, CatArc(1, 3, 0.
0, 0.0)), Arc(2, 4, CatArc(2, 1, 0.01, 0.02)), Arc(2, 5, CatArc(2, 2, 0.0
1, 0.01)), Arc(3, 5, CatArc(2, 1, 0.0, 0.01)), Arc(3, 6, CatArc(2, 2, 0.0,
0.0)), Arc(4, 7, TermArc(0.02, 1)), Arc(4, 7, TermArc(0.02, 2)), Arc(4, 7,
TermArc(0.02, 3)), Arc(5, 7, TermArc(0.01, 1)), Arc(5, 7, TermArc(0.01,
2)), Arc(5, 7, TermArc(0.01, 3)), Arc(6, 7, TermArc(0.0, 1)), Arc(6, 7, Te
rmArc(0.0, 2)), Arc(6, 7, TermArc(0.0, 3))], 1, 7), SampleDAG(298, [(0, 0.
0), (1, 0.01), (1, 0.0), (2, 0.01), (2, 0.02), (2, 0.0), (3, 0.0)], Arc[Ar
c(1, 2, CatArc(1, 1, 0.0, 0.01)), Arc(1, 3, CatArc(1, 2, 0.0, 0.0)), Arc
(1, 2, CatArc(1, 3, 0.0, 0.01)), Arc(2, 4, CatArc(2, 1, 0.01, 0.01)), Arc
(2, 5, CatArc(2, 2, 0.01, 0.02)), Arc(3, 6, CatArc(2, 1, 0.0, 0.0)), Arc
(3, 4, CatArc(2, 2, 0.0, 0.01)), Arc(4, 7, TermArc(0.01, 1)), Arc(4, 7, Te
rmArc(0.01, 2)), Arc(4, 7, TermArc(0.01, 3)), Arc(5, 7, TermArc(0.02, 1)),
Arc(5, 7, TermArc(0.02, 2)), Arc(5, 7, TermArc(0.02, 3)), Arc(6, 7, TermAr
c(0.0, 1)), Arc(6, 7, TermArc(0.0, 2)), Arc(6, 7, TermArc(0.0, 3))], 1,
7), SampleDAG(299, [(0, 0.0), (1, 0.01), (1, 0.0), (2, 0.01), (2, 0.02),
(2, 0.0), (3, 0.0)], Arc[Arc(1, 2, CatArc(1, 1, 0.0, 0.01)), Arc(1, 3, Cat
Arc(1, 2, 0.0, 0.0)), Arc(1, 2, CatArc(1, 3, 0.0, 0.01)), Arc(2, 4, CatArc
(2, 1, 0.01, 0.01)), Arc(2, 5, CatArc(2, 2, 0.01, 0.02)), Arc(3, 6, CatArc
(2, 1, 0.0, 0.0)), Arc(3, 4, CatArc(2, 2, 0.0, 0.01)), Arc(4, 7, TermArc
(0.01, 1)), Arc(4, 7, TermArc(0.01, 2)), Arc(4, 7, TermArc(0.01, 3)), Arc
(5, 7, TermArc(0.02, 1)), Arc(5, 7, TermArc(0.02, 2)), Arc(5, 7, TermArc
(0.02, 3)), Arc(6, 7, TermArc(0.0, 1)), Arc(6, 7, TermArc(0.0, 2)), Arc(6,
7, TermArc(0.0, 3))], 1, 7), SampleDAG(300, [(0, 0.0), (1, 0.01), (1, 0.
0), (2, 0.02), (2, 0.01), (2, 0.0), (3, 0.0)], Arc[Arc(1, 2, CatArc(1, 1,
0.0, 0.01)), Arc(1, 2, CatArc(1, 2, 0.0, 0.01)), Arc(1, 3, CatArc(1, 3, 0.
0, 0.0)), Arc(2, 4, CatArc(2, 1, 0.01, 0.02)), Arc(2, 5, CatArc(2, 2, 0.0
1, 0.01)), Arc(3, 5, CatArc(2, 1, 0.0, 0.01)), Arc(3, 6, CatArc(2, 2, 0.0,
0.0)), Arc(4, 7, TermArc(0.02, 1)), Arc(4, 7, TermArc(0.02, 2)), Arc(4, 7,
TermArc(0.02, 3)), Arc(5, 7, TermArc(0.01, 1)), Arc(5, 7, TermArc(0.01,
2)), Arc(5, 7, TermArc(0.01, 3)), Arc(6, 7, TermArc(0.0, 1)), Arc(6, 7, Te
rmArc(0.0, 2)), Arc(6, 7, TermArc(0.0, 3))], 1, 7)], n_nodes = [7, 7, 7,
7, 7, 7, 7, 7, 7, 7  …  7, 7, 7, 7, 7, 7, 7, 7, 7, 7], β0 = β0, β_x = Vari
```

```
ableRef[β_x[1], β_x[2]], β_z = VariableRef[β_z[1], β_z[2], β_z[3]], β_grp
= VariableRef[β_grp[1], β_grp[2]], λ = λ, r = VariableRef[r[1], r[2], r
[3], r[4], r[5], r[6], r[7], r[8], r[9], r[10]  …  r[291], r[292], r[293],
r[294], r[295], r[296], r[297], r[298], r[299], r[300]], μ = VariableRef[μ
[1,1] μ[1,2] … μ[1,6] μ[1,7]; μ[2,1] μ[2,2] … μ[2,6] μ[2,7]; … ; μ[299,1]
μ[299,2] … μ[299,6] μ[299,7]; μ[300,1] μ[300,2] … μ[300,6] μ[300,7]]), (β0
= -0.3, βx = [1.0, -0.7], βz = [0.8, -0.5, 0.6], βgrp = [0.5, -0.4]), (-0.
26966141856599213, [0.7268207209228954, -0.8371227622608812], [0.797764968
1015079, -0.15333529801756676, 0.5879875082314706], [0.8779178610381262, -
0.6205919690109933]))
```

Experiments

some helper functions

In [79]:

```
##########################################################
# 1. Parameter containers
##########################################################
"""
DROParams

Stores all hyperparameters that define the subgroup
Wasserstein ground metric and DRO radius:

d(ξ^i, ξ) =
  A_g * Σ_j γ_j |x_j - x_j^i|
+ B_g * Σ_ℓ δ_ℓ 1[z_ℓ ≠ z_ℓ^i]
+ C_g * 1[g ≠ g^i]
+ ∞ * 1[y ≠ y^i].
"""
struct DROParams
    delta::Vector{Float64}       # length m, δ_ℓ for categorical features
    A_group::Vector{Float64}     # length num_g, A_g for continuous part
    B_group::Vector{Float64}     # length num_g, B_g for categorical part
    C_group::Vector{Float64}     # length num_g, C_g for group jumps
    gamma_x::Vector{Float64}     # length n_x, γ_j for continuous features
    epsilon::Float64             # Wasserstein radius ε
end


"""
LogitParams

Hyperparameters for standard (non-robust) logistic regression.
"""
struct LogitParams
    lambda_l2::Float64           # L2 regularization coefficient
end


"""
PerturbParams

Extra knobs for generating perturbed test sets.

The actual "cost" of moving features comes from DROParams
(via A_g * γ_j and B_g * δ_ℓ). Here we only store:
```

```
- mode : scenario label
- w    : half-width for continuous noise U(-w, w)
"""
struct PerturbParams
    mode::Symbol
    w::Float64
end

########################################################
# 2. Simple helper: DRO → Perturb
########################################################

"""
    dro_to_perturb_params(dro; w=0.4, mode=:generic)

Create a PerturbParams object from a DROParams, choosing only
the non-metric perturbation hyperparameters (mode, w).

The metric itself (A_group, B_group, gamma_x, delta) is used
directly inside the perturbation routine, not stored here.
"""
function dro_to_perturb_params(
    dro::DROParams;
    w::Float64 = 0.4,
    mode::Symbol = :generic,
)
    return PerturbParams(mode, w)
end
```

```
dro_to_perturb_params
```

In [80]:

```
########################################################
# 3. Model fitting (DRO vs Logistic) + shared prediction
########################################################

using JuMP
using Ipopt

"""
    fit_dro_model(X, Z, group, y, encinfo, params; optimizer)

Fit the subgroup-Wasserstein DRO logistic regression using
`build_group_dro_graph_model` and return coefficients for prediction.

Inputs
------
- X         : N×n_x continuous features
- Z         : N×m integer-coded categorical features
- group     : length-N group indices in {1,…,num_g}
- y         : length-N labels in {-1,+1}
- encinfo   : ZGEncodingInfo (reduced encoding info)
- params    : DROParams
- optimizer : JuMP optimizer constructor (default Ipopt)

Returns
-------
NamedTuple (β0, β_x, β_z, β_grp, encinfo)
"""
function fit_dro_model(
```

```
    X::AbstractMatrix,
    Z::AbstractMatrix{<:Integer},
    group::AbstractVector{<:Integer},
    y::AbstractVector{<:Integer},
    encinfo::ZGEncodingInfo,
    params::DROParams;
    optimizer = optimizer_with_attributes(Ipopt.Optimizer, "print_level"
)
    model, meta = build_group_dro_graph_model(
        X,
        Z,
        group,
        y,
        encinfo,
        params.delta,
        params.A_group,
        params.B_group,
        params.C_group,
        params.gamma_x,
        params.epsilon,
        optimizer,
    )

    optimize!(model)

    β0̂   = value(meta.β0)
    βx̂   = value.(meta.β_x)
    βẑ   = value.(meta.β_z)
    βgrp̂ = value.(meta.β_grp)

    return (
        β0    = β0̂,
        β_x   = βx̂,
        β_z   = βẑ,
        β_grp = βgrp̂,
        encinfo = encinfo,
    )
end


"""
    fit_logistic_model(Xtr, Zenc_tr, Genc_tr, ytr, encinfo, params; optim

Fit standard logistic regression with L2 regularization, using the
same reduced encoding as the DRO model:

    f_β(x,z,g) = β₀ + β_xᵀ x + β_zᵀ φ_z(z) + β_grpᵀ φ_g(g)

and

    min_β (1/N) Σ log(1 + exp(-y_i f_β(x_i,z_i,g_i)))
          + (λ/2)(‖β_x‖² + ‖β_z‖² + ‖β_grp‖²),

with λ = params.lambda_l2.
"""
function fit_logistic_model(
    Xtr::AbstractMatrix,
    Zenc_tr::AbstractMatrix,
    Genc_tr::AbstractMatrix,
    ytr::AbstractVector{<:Integer},
```

```
    encinfo::ZGEncodingInfo,
    params::LogitParams;
    optimizer = optimizer_with_attributes(Ipopt.Optimizer, "print_level"
)
    N, n_x  = size(Xtr)
    N2, p_z = size(Zenc_tr)
    N3, p_g = size(Genc_tr)

    @assert N2 == N "Zenc_tr must have same rows as Xtr"
    @assert N3 == N "Genc_tr must have same rows as Xtr"

    λ = params.lambda_l2

    model = Model(optimizer)

    @variable(model, β0)
    @variable(model, β_x[1:n_x])
    @variable(model, β_z[1:p_z])
    @variable(model, β_grp[1:p_g])

    # Logistic loss + L2 penalty (no penalty on intercept)
    @NLobjective(model, Min,
        (1.0 / N) * sum(
            log(1 + exp(-ytr[i] * (
                β0
                + sum(β_x[j] * Xtr[i, j] for j in 1:n_x)
                + sum(β_z[k] * Zenc_tr[i, k] for k in 1:p_z)
                + sum(β_grp[h] * Genc_tr[i, h] for h in 1:p_g)
            )))
            for i in 1:N
        )
        + (λ / 2.0) * (
            sum(β_x[j]^2 for j in 1:n_x) +
            sum(β_z[k]^2 for k in 1:p_z) +
            sum(β_grp[h]^2 for h in 1:p_g)
        )
    )

    optimize!(model)

    return (
        β0    = value(β0),
        β_x   = value.(β_x),
        β_z   = value.(β_z),
        β_grp = value.(β_grp),
        encinfo = encinfo,
    )
end


#########################################################
# Shared prediction functions
#########################################################

# Logistic link
σ(t) = 1.0 / (1.0 + exp(-t))

"""
    predict_scores(β, X, Z, group)
```

```
Compute linear scores f_β(x,z,g) for each row, using the
encoding described by β.encinfo.
"""
function predict_scores(
    β,
    X::AbstractMatrix,
    Z::AbstractMatrix{<:Integer},
    group::AbstractVector{<:Integer},
)
    N, n_x = size(X)
    k_z     = β.encinfo.k_z
    z_start = β.encinfo.z_start
    num_g   = β.encinfo.num_g

    m = length(k_z)
    @assert size(Z, 2) == m
    @assert length(group) == N

    scores = zeros(Float64, N)

    for i in 1:N
        s = β.β0 + dot(β.β_x, view(X, i, :))

        # categorical part
        for ℓ in 1:m
            val = Z[i, ℓ]
            k_l = k_z[ℓ]
            if val < k_l
                idx = z_start[ℓ] + (val - 1)
                s += β.β_z[idx]
            end
        end

        # group part
        g_i = group[i]
        if g_i < num_g
            s += β.β_grp[g_i]
        end

        scores[i] = s
    end

    return scores
end

"""
    predict_proba(β, X, Z, group)

Return P(y=+1 | x,z,g) = σ(f_β(x,z,g)) for each row.
"""
predict_proba(β, X, Z, group) = σ.(predict_scores(β, X, Z, group))

"""
    predict_label(β, X, Z, group)

Return hard labels in {-1,+1} using sign(f_β).
"""
function predict_label(
    β,
    X::AbstractMatrix,
```

```
    Z::AbstractMatrix{<:Integer},
    group::AbstractVector{<:Integer},
)
    scores = predict_scores(β, X, Z, group)
    yhat   = Vector{Int}(undef, length(scores))
    @inbounds for i in eachindex(scores)
        yhat[i] = scores[i] >= 0 ? 1 : -1
    end
    return yhat
end
```

predict_label

In [88]:

```
########################################################
# 4. Group-aware test-set perturbation driven by metric
########################################################

using Random
using Distributions   # for Laplace

"""
    perturb_testset(X, Z, group, y, encinfo, dro, pert; rng)

Generate one perturbed test set (X̃, Z̃, g̃, ỹ) from
(X, Z, group, y), using the subgroup Wasserstein metric:

d(ξ^i, ξ) =
  A_g Σ_j γ_j |x_j - x_j^i|
+ B_g Σ_ℓ δ_ℓ 1[z_ℓ ≠ z_ℓ^i]
+ C_g 1[g ≠ g^i].

We use these products A_g·γ_j, B_g·δ_ℓ, C_g to control the
amount of noise:

Continuous features:
  For sample i and feature j (group g_i):
    cost_x = A_{g_i} * γ_j
    scale  = w / cost_x
    Δ_raw  ~ Laplace(0, scale)
    Δ      = clamp(Δ_raw, -w, w)
    X̃[i,j] = X[i,j] + Δ

Categorical features:
  For sample i and feature ℓ (group g_i):
    cost_z = B_{g_i} * δ_ℓ
    q      = 1 - exp(- w / cost_z)   ∈ (0,1)
    With prob 1-q: keep z_ℓ^i
    With prob q  : change uniformly to one of the other levels.

Group index:
  For sample i:
    cost_g = C_{g_i}
    qg     = 1 - exp(- w / cost_g)   ∈ (0,1)
    With prob 1-qg: keep g_i
    With prob qg  : change to a different group, uniformly.

Inputs:
- X, Z, group, y : test data
- encinfo        : ZGEncodingInfo (k_z, num_g)
- dro            : DROParams (delta, A_group, B_group, C_group, gamma_x,
```

```
- pert          : PerturbParams (mode, w)
- rng           : random number generator

Returns:
- X̃, Z̃, g̃, ỹ : perturbed copies (same shapes as inputs)
"""
function perturb_testset(
    X::AbstractMatrix,
    Z::AbstractMatrix{<:Integer},
    group::AbstractVector{<:Integer},
    y::AbstractVector,
    encinfo::ZGEncodingInfo,
    dro::DROParams,
    pert::PerturbParams;
    rng = Random.default_rng(),
)
    N, n_x = size(X)
    _, m   = size(Z)
    k_z    = encinfo.k_z
    num_g  = encinfo.num_g

    @assert length(dro.gamma_x) == n_x
    @assert length(dro.delta)   == m
    @assert length(dro.A_group) == num_g
    @assert length(dro.B_group) == num_g
    @assert length(dro.C_group) == num_g

    X̃ = copy(X)
    Z̃ = copy(Z)
    g̃ = copy(group)
    ỹ = copy(y)

    w = pert.w
    @assert w > 0 "pert.w must be positive"

    for i in 1:N
        g_i = group[i]
        @assert 1 ≤ g_i ≤ num_g

        A_gi = dro.A_group[g_i]
        B_gi = dro.B_group[g_i]
        C_gi = max(dro.C_group[g_i], 1e-6)  # avoid zero

        # -------------------------
        # 1. Continuous features
        # -------------------------
        for j in 1:n_x
            cost_x = max(A_gi * dro.gamma_x[j], 1e-6)
            scale  = w / cost_x
            Δ_raw  = rand(rng, Laplace(0.0, scale))
            # clip extreme jumps; still O(w / (A_gγ))
            Δ      = clamp(Δ_raw, -w, w)
            X̃[i, j] = X[i, j] + Δ
        end

        # -------------------------
        # 2. Categorical features
        # -------------------------
        for ℓ in 1:m
            k_l   = k_z[ℓ]
```

```
                cost_z = max(B_gi * dro.delta[ℓ], 1e-6)
                # higher cost_z → smaller q
                q = 1.0 - exp(- w / cost_z)
                q = clamp(q, 0.0, 0.99)

                if rand(rng) < q
                    old = Z[i, ℓ]
                    # choose any other category uniformly
                    choices = Vector{Int}(undef, k_l - 1)
                    idx = 1
                    for c in 1:k_l
                        if c != old
                            choices[idx] = c
                            idx += 1
                        end
                    end
                    Z̃[i, ℓ] = rand(rng, choices)
                end
            end

            # --------------------------
            # 3. Group index
            # --------------------------
            # If C_g very large, qg is very small (hard to move group)
            qg = 1.0 - exp(- w / C_gi)
            qg = clamp(qg, 0.0, 0.99)

            if rand(rng) < qg
                choices_g = Vector{Int}(undef, num_g - 1)
                idxg = 1
                for gg in 1:num_g
                    if gg != g_i
                        choices_g[idxg] = gg
                        idxg += 1
                    end
                end
                g̃[i] = rand(rng, choices_g)
            end
        end

    return X̃, Z̃, g̃, ỹ
end
```

perturb_testset

In [89]:

```
##########################################################
# 5. Evaluation metrics: AUC and ACE
##########################################################

using Statistics

"""
    auc_binary(scores, y)

Binary AUC (ROC area).

- `scores`: model scores (logits or probabilities).
- `y`: true labels, positives are `y > 0`.

Implements the classic rank-based formula (Mann–Whitney U):
```

```
    AUC = (∑ rank(pos) − P(P+1)/2) / (P·N),

where P = #positives, N = #negatives.
Returns 0.5 if all labels are the same.
"""
function auc_binary(scores::AbstractVector{<:Real},
                    y::AbstractVector{<:Real})
    @assert length(scores) == length(y)
    n = length(scores)

    # map labels to 0/1
    y01 = Vector{Int}(undef, n)
    for i in 1:n
        y01[i] = y[i] > 0 ? 1 : 0
    end

    P = sum(y01)
    N = n - P
    (P == 0 || N == 0) && return 0.5

    # ranks of scores (ascending)
    idx   = sortperm(scores)
    ranks = similar(scores, Float64)
    for (r, i) in enumerate(idx)
        ranks[i] = r
    end

    # sum of ranks for positives
    sum_r_pos = 0.0
    for i in 1:n
        if y01[i] == 1
            sum_r_pos += ranks[i]
        end
    end

    auc = (sum_r_pos - P * (P + 1) / 2) / (P * N)
    return auc
end


"""
    ace_binary(probs, y; B = 10)

Adaptive Calibration Error (ACE) for binary classification.

- `probs`: predicted P(y=1) ∈ [0,1].
- `y`: true labels, positives are `y > 0`.
- `B`: number of equal–mass bins (default 10).

Procedure:
1. Sort by `probs`.
2. Split into B bins with (almost) equal size.
3. For each bin:
       | mean(probs) − mean(labels) |
4. ACE = average of these B values.

Lower ACE = better calibration.
"""
function ace_binary(probs::AbstractVector{<:Real},
```

```
                    y::AbstractVector{<:Real};
                    B::Int = 10)
    @assert length(probs) == length(y)
    n = length(probs)
    n == 0 && return 0.0

    # map labels to 0/1
    y01 = Vector{Float64}(undef, n)
    for i in 1:n
        y01[i] = y[i] > 0 ? 1.0 : 0.0
    end

    B = min(B, n)

    # sort by predicted probability
    idx = sortperm(probs)

    base  = div(n, B)
    extra = n % B

    start   = 1
    err_sum = 0.0
    for b in 1:B
        sz   = base + (b <= extra ? 1 : 0)
        stop = start + sz - 1

        inds = idx[start:stop]
        mean_conf = mean(@view probs[inds])
        mean_acc  = mean(@view y01[inds])

        err_sum += abs(mean_acc - mean_conf)
        start = stop + 1
    end

    return err_sum / B
end


"""
    default_metrics(probs, y)

Convenience wrapper:

- `auc`: ROC AUC (higher is better)
- `ace`: Adaptive Calibration Error (lower is better)
"""
default_metrics(probs::AbstractVector{<:Real},
                y::AbstractVector{<:Real}) = (
    auc = auc_binary(probs, y),
    ace = ace_binary(probs, y; B = 10),
)
```

default_metrics

In [90]:

```
########################################################
# 6. High-level experiment driver (UNPAIRED)
########################################################

"""
    run_experiment(
```

```
        X, Z, group, y;
        train_ratio = 0.7,
        method = :dro,
        dro_params = nothing,
        logit_params = nothing,
        pert_params::PerturbParams,
        n_splits = 1,
        n_pert_per_split = 100,
        metric_fun = default_metrics,
        rng = Random.default_rng(),
        optimizer = optimizer_with_attributes(Ipopt.Optimizer, "print_lev
    )

Run a robustness experiment for one model type:

- `method = :dro`       → subgroup DRO logistic regression
- `method = :logistic`  → standard L2 logistic regression

For each split s = 1,…,n_splits:

1. Randomly split data into train / test (ratio = `train_ratio`).
2. Fit the chosen model on the train set.
3. Generate `n_pert_per_split` independent perturbed test sets
   with `perturb_testset`, using the *same* DRO metric (`dro_params`)
   but possibly different model (`method`).
4. On each perturbed test set, compute metrics via `metric_fun`.
5. Aggregate metrics over perturbations into:
   - average performance (mean AUC / ACE),
   - worst-case performance (min AUC / max ACE).

Note (important):
- Even for `method = :logistic`, `dro_params` is still required: it
  defines the ground metric that drives the test-set perturbations.
- DRO vs logistic are compared under the same *shift model*.

Returns:
- `Vector{NamedTuple}`, one per split, with fields:

    (split, auc_avg, auc_min, ace_avg, ace_max)
"""
function run_experiment(
    X::AbstractMatrix,
    Z::AbstractMatrix{<:Integer},
    group::AbstractVector{<:Integer},
    y::AbstractVector,
    ;
    train_ratio::Float64 = 0.7,
    method::Symbol = :dro,
    dro_params::Union{DROParams,Nothing} = nothing,
    logit_params::Union{LogitParams,Nothing} = nothing,
    pert_params::PerturbParams,
    n_splits::Int = 1,
    n_pert_per_split::Int = 100,
    metric_fun = default_metrics,
    rng = Random.default_rng(),
    optimizer = optimizer_with_attributes(Ipopt.Optimizer, "print_level"
)
    N = size(X, 1)
    @assert 0.0 < train_ratio < 1.0 "train_ratio must be in (0,1)"
    @assert size(Z, 1) == N
```

```
    @assert length(group) == N
    @assert length(y) == N
    @assert dro_params !== nothing "dro_params is required (also for :log

    dro = dro_params::DROParams

    # Encode Z and group once, so DRO and logistic share the same encodin
    Z_enc_all, G_enc_all, encinfo = encode_zg_reduced(Z, group)

    results = NamedTuple[]

    for s in 1:n_splits
        # 1. Random train/test split
        idx      = randperm(rng, N)
        n_train  = max(1, min(N - 1, round(Int, train_ratio * N)))
        train_idx = idx[1:n_train]
        test_idx  = idx[n_train+1:end]

        Xtr = X[train_idx, :]
        Ztr = Z[train_idx, :]
        gtr = group[train_idx]
        ytr = y[train_idx]

        Xte = X[test_idx, :]
        Zte = Z[test_idx, :]
        gte = group[test_idx]
        yte = y[test_idx]

        Zenc_tr = Z_enc_all[train_idx, :]
        Genc_tr = G_enc_all[train_idx, :]

        # 2. Fit model
        β =
            if method == :dro
                fit_dro_model(
                    Xtr, Ztr, gtr, ytr,
                    encinfo, dro;
                    optimizer = optimizer,
                )
            elseif method == :logistic
                @assert logit_params !== nothing "logit_params must be pr
                fit_logistic_model(
                    Xtr, Zenc_tr, Genc_tr, ytr,
                    encinfo, logit_params;
                    optimizer = optimizer,
                )
            else
                error("Method $(method) not implemented. Use :dro or :log
            end

        # 3. Monte Carlo over perturbed test sets
        metric_vals = Vector{NamedTuple}(undef, n_pert_per_split)

        for r in 1:n_pert_per_split
            X̃, Z̃, g̃, ỹ = perturb_testset(
                Xte, Zte, gte, yte,
                encinfo, dro, pert_params;
                rng = rng,
            )
            probs = predict_proba(β, X̃, Z̃, g̃)
```

```
            metric_vals[r] = metric_fun(probs, ỹ)
        end

        # 4. Aggregate to average / worst-case
        aucs = [mv.auc for mv in metric_vals]
        aces = [mv.ace for mv in metric_vals]

        push!(results, (
            split   = s,
            auc_avg = mean(aucs),
            auc_min = minimum(aucs),
            ace_avg = mean(aces),
            ace_max = maximum(aces),
        ))
    end

    return results
end
```

```
run_experiment
```

synthetic data experiment

In [91]:
```
#########################################################
# Synthetic mixed-feature dataset generator
# (3 groups, 2D continuous, 1 categorical with 3 levels)
#########################################################


"""
    generate_synthetic_mixed_data(N; rng = Random.default_rng())

Generate a synthetic dataset with:
- 3 groups g ∈ {1,2,3},
- 2 continuous features x ∈ ℝ²,
- 1 categorical feature z ∈ {1,2,3},
- labels y ∈ {-1,+1} drawn from a *known* logistic model f_{β*}.

The true logistic model uses the *same reduced encoding convention*
as our DRO / logistic models:

    f_{β*}(x,z,g)
      = β0
        + β_xᵀ x
        + β_zᵀ φ_z(z)
        + β_grpᵀ φ_g(g),

where:
- categorical z has 3 levels, with reduced dummies for {1,2} and
  level 3 as baseline;
- group g has 3 levels, with reduced dummies for {1,2} and
  group 3 as baseline.

We fix (for illustration):

    β0    = 0.0
    β_x   = [1.0, -0.5]
    β_z   = [0.8, -0.8]     # z = 1 positive shift, z = 2 negative shift,
    β_grp = [-0.5, 0.5]     # group 1 is slightly negative, group 2 sligh
```

```
Data-generating process
-----------------------
For each sample i = 1,...,N:

1. Draw group g_i:
       P(g=1)=0.5, P(g=2)=0.3, P(g=3)=0.2.

2. Conditional on g_i, draw categorical z_i:
   - if g=1: P(z=1,2,3) = (0.5, 0.3, 0.2)
   - if g=2: P(z=1,2,3) = (0.2, 0.5, 0.3)
   - if g=3: P(z=1,2,3) = (0.3, 0.2, 0.5)

3. Draw continuous features x_i ∈ ℝ²:
   - base group means:
         μ₁ = (0, 0),
         μ₂ = (1, 1),
         μ₃ = (-1, 1);
   - group-specific standard deviations:
         σ₁ = 0.5, σ₂ = 1.0, σ₃ = 0.7;
   - category-specific shifts:
         Δ₁ = (0.5, 0.0),
         Δ₂ = (0.0, -0.5),
         Δ₃ = (0.0, 0.0).
   - set μ_{g,z} = μ_g + Δ_z and draw
         x_i[j] ~ Normal( μ_{g,z}[j], σ_g² ), independently for j=1,2.

4. Compute the “true” logit:
       η_i = β0 + β_xᵀ x_i + β_zᵀ φ_z(z_i) + β_grpᵀ φ_g(g_i),
   then draw
       y_i ~ Bernoulli( σ(η_i) ),
   and map {0,1} to {-1,+1}.

Outputs
-------
- X      :: Matrix{Float64}  (N × 2)      continuous features
- Z      :: Matrix{Int}      (N × 1)      categorical feature (values 1,2
- group  :: Vector{Int}      (length N)   group indices (1,2,3)
- y      :: Vector{Int}      (length N)   labels in {-1,+1}
- β_true :: NamedTuple       (β0, β_x, β_z, β_grp), using *reduced encodi
      β_true.β_z   has length 2  (for z=1,2; z=3 baseline)
      β_true.β_grp has length 2  (for g=1,2; g=3 baseline)
"""

function generate_synthetic_mixed_data(
    N::Int;
    rng = Random.default_rng(),
)
    # --------------------------
    # 0. Dimensions
    # --------------------------
    num_g = 3          # number of groups
    n_x   = 2          # continuous features
    m     = 1          # categorical components
    k1    = 3          # categories for the single categorical feature

    # --------------------------
    # 1. True logistic parameters in reduced encoding
    # --------------------------
    β0_true    = 0.0
```

```
β_x_true   = [1.0, -0.5]                # length 2

# categorical z: 3 levels → 2 reduced dummies
#   z=1 : φ_z(z) = (1,0)   → coefficient β_z[1] = +0.8
#   z=2 : φ_z(z) = (0,1)   → coefficient β_z[2] = -0.8
#   z=3 : baseline         → (0,0)
β_z_true   = [0.8, -0.8]               # length 2

# group g: 3 levels → 2 reduced dummies
#   g=1 : φ_g(g) = (1,0)   → coefficient β_grp[1] = -0.5
#   g=2 : φ_g(g) = (0,1)   → coefficient β_grp[2] = +0.5
#   g=3 : baseline         → (0,0)
β_grp_true = [-0.5, 0.5]               # length 2

# -------------------------
# 2. Group and categorical distributions
# -------------------------
# Group probabilities: (0.5, 0.3, 0.2)
group_probs = [0.5, 0.3, 0.2]
group_dist  = Categorical(group_probs)

# Conditional categorical probabilities P(z | g)
# rows: g = 1,2,3; columns: z = 1,2,3
z_probs = [
    0.5  0.3  0.2;   # g = 1
    0.2  0.5  0.3;   # g = 2
    0.3  0.2  0.5    # g = 3
]
z_dists = [Categorical(z_probs[g, :]) for g in 1:num_g]

# -------------------------
# 3. Continuous feature means and std per group/category
# -------------------------
# base means μ_g
μ_g = [
    [0.0, 0.0],      # g = 1
    [1.0, 1.0],      # g = 2
    [-1.0, 1.0],     # g = 3
]

# group-specific standard deviations
σ_g = [0.5, 1.0, 0.7]

# category-specific shifts Δ_z
Δ_z = [
    [0.5,  0.0],     # z = 1
    [0.0, -0.5],     # z = 2
    [0.0,  0.0],     # z = 3
]

# -------------------------
# 4. Allocate output arrays
# -------------------------
X      = zeros(Float64, N, n_x)  # N × 2
Z      = Matrix{Int}(undef, N, m)  # N × 1
group  = Vector{Int}(undef, N)
y      = Vector{Int}(undef, N)

# -------------------------
# 5. Generate samples
```

```
    # --------------------------
    for i in 1:N
        # (a) sample group
        g_i = rand(rng, group_dist)
        group[i] = g_i

        # (b) conditional categorical
        z_i = rand(rng, z_dists[g_i])   # ∈ {1,2,3}
        Z[i, 1] = z_i

        # (c) continuous x given (g,z)
        μ_base = μ_g[g_i]
        σ      = σ_g[g_i]
        μ_gz   = [μ_base[1] + Δ_z[z_i][1],
                  μ_base[2] + Δ_z[z_i][2]]

        X[i, 1] = rand(rng, Normal(μ_gz[1], σ))
        X[i, 2] = rand(rng, Normal(μ_gz[2], σ))

        # (d) true logit η_i using the same reduced encoding as our model
        η = β0_true + β_x_true[1]*X[i,1] + β_x_true[2]*X[i,2]

        # categorical contribution
        if z_i == 1
            η += β_z_true[1]
        elseif z_i == 2
            η += β_z_true[2]
        end


        # group contribution
        if g_i == 1
            η += β_grp_true[1]
        elseif g_i == 2
            η += β_grp_true[2]
        end

        # (e) sample label y_i ∈ {-1,+1}
        p = 1.0 / (1.0 + exp(-η))   # σ(η)
        y[i] = rand(rng) < p ? 1 : -1
    end

    β_true = (
        β0    = β0_true,
        β_x   = β_x_true,
        β_z   = β_z_true,
        β_grp = β_grp_true,
    )

    return X, Z, group, y, β_true
end
```

generate_synthetic_mixed_data (generic function with 1 method)

In [92]:

```
#########################################################
# Scenario U / V / R for synthetic experiment
#########################################################

# Metric scenarios for (A_g, B_g, C_g), g = 1,2,3.
# U = uniform      : all groups same robustness
```

```
# V = "vulnerable" : group 1 easy to move, 3 hardest
# R = reversed     : group 1 hardest, 3 easiest


struct ScenarioSpec
    name::Symbol
    A_group::Vector{Float64}
    B_group::Vector{Float64}
    C_group::Vector{Float64}
end

const SCENARIO_U = ScenarioSpec(
    :U,
    [1.0, 1.0, 1.0],   # A_g
    [1.0, 1.0, 1.0],   # B_g
    [1.0, 1.0, 1.0],   # C_g
)

const SCENARIO_V = ScenarioSpec(
    :V,
    [0.5, 1.0, 2.0],   # group 1 cheap, group 3 expensive
    [0.5, 1.0, 2.0],
    [0.5, 1.0, 2.0],
)


const SCENARIO_R = ScenarioSpec(
    :R,
    [2.0, 1.0, 0.5],   # reverse of V
    [2.0, 1.0, 0.5],
    [2.0, 1.0, 0.5],
)


const SCENARIOS_UVR = [SCENARIO_U, SCENARIO_V, SCENARIO_R]


#########################################################
# Build DROParams for the synthetic setting
#########################################################

"""
    build_synth_dro_params(scen; theta=0.75)

Construct DROParams for the synthetic 2D+1cat, 3-group model
given a metric scenario `scen` and robustness level `theta`.

- gamma_x = [1, 1]
- delta   = [1]
- epsilon = -log(theta)
"""
function build_synth_dro_params(
    scen::ScenarioSpec;
    theta::Float64 = 0.75,
)
    n_x = 2   # 2 continuous features
    m   = 1   # 1 categorical component

    gamma_x = ones(Float64, n_x)
    delta   = ones(Float64, m)
```

```
    epsilon = -log(theta)

    return DROParams(
        delta,
        scen.A_group,
        scen.B_group,
        scen.C_group,
        gamma_x,
        epsilon,
    )
end


##########################################################
# UVR synthetic experiment driver (unpaired)
##########################################################

using Random
using Statistics
using Ipopt
using JuMP

"""
    run_synthetic_UVR(;
        N = 3000,
        thetas = [0.5, 0.75, 0.9],
        w = 1.0,
        n_splits = 5,
        n_pert_per_split = 100,
        lambda_l2 = 0.01,
        seed = 2025,
        optimizer = optimizer_with_attributes(Ipopt.Optimizer, "print_lev
    )

    Run the synthetic experiment for scenarios U/V/R and both models
(:dro, :logistic).

For each scenario s ∈ {U,V,R}, each θ, and each method:
- Generate a fresh dataset of size N.
- Random 70/30 train/test split.
- Build DROParams using the scenario + θ.
- Fit DRO or logistic model on training data.
- For each of `n_pert_per_split` draws:
    * perturb the test set using the metric-based `perturb_testset`
      with global noise scale w;
    * compute metrics via `default_metrics`.
- Aggregate metrics over perturbations and over `n_splits`.

Returns a Vector of NamedTuples with fields:
  (scenario, theta, method, auc_avg_mean, auc_min_mean,
   ace_avg_mean, ace_max_mean)
"""
function run_synthetic_UVR(;
    N::Int = 3000,
    thetas::Vector{Float64} = [0.5, 0.75, 0.9],
    w::Float64 = 1.0,
    n_splits::Int = 5,
    n_pert_per_split::Int = 100,
    lambda_l2::Float64 = 0.01,
    seed::Int = 2025,
```

```
    optimizer = optimizer_with_attributes(Ipopt.Optimizer, "print_level"
)
    rng = MersenneTwister(seed)

    results = NamedTuple[]

    for scen in SCENARIOS_UVR
        for theta in thetas
            # metric / DRO hyper-params (same for both models, used in pe
            dro_params = build_synth_dro_params(scen; theta = theta)
            pert_params = PerturbParams(:synthetic, w)
            logit_params = LogitParams(lambda_l2)

            for method in (:dro, :logistic)
                # store per-split summaries
                split_metrics = NamedTuple[]

                for s in 1:n_splits
                    # ------------------------------------------
                    # 1) Generate data + encode
                    # ------------------------------------------
                    X, Z, group, y, _ = generate_synthetic_mixed_data(N;

                    Ntot = size(X, 1)
                    idx  = randperm(rng, Ntot)
                    n_tr = max(1, min(Ntot - 1, round(Int, 0.7 * Ntot)))
                    tr_idx = idx[1:n_tr]
                    te_idx = idx[n_tr+1:end]

                    Xtr = X[tr_idx, :]
                    Ztr = Z[tr_idx, :]
                    gtr = group[tr_idx]
                    ytr = y[tr_idx]

                    Xte = X[te_idx, :]
                    Zte = Z[te_idx, :]
                    gte = group[te_idx]
                    yte = y[te_idx]

                    Z_enc_all, G_enc_all, encinfo = encode_zg_reduced(Z,
                    Zenc_tr = Z_enc_all[tr_idx, :]
                    Genc_tr = G_enc_all[tr_idx, :]

                    # ------------------------------------------
                    # 2) Fit model
                    # ------------------------------------------
                    β =
                        if method == :dro
                            fit_dro_model(
                                Xtr, Ztr, gtr, ytr,
                                encinfo, dro_params;
                                optimizer = optimizer,
                            )
                        else
                            fit_logistic_model(
                                Xtr, Zenc_tr, Genc_tr, ytr,
                                encinfo, logit_params;
                                optimizer = optimizer,
                            )
                        end
```

```
                    # -------------------------------------------
                    # 3) Monte Carlo over perturbed test sets
                    # -------------------------------------------
                    mvals = Vector{NamedTuple}(undef, n_pert_per_split)
                    for r in 1:n_pert_per_split
                        X̃, Z̃, g̃, ỹ = perturb_testset(
                            Xte, Zte, gte, yte,
                            encinfo, dro_params, pert_params;
                            rng = rng,
                        )
                        probs = predict_proba(β, X̃, Z̃, g̃)
                        mvals[r] = default_metrics(probs, ỹ)
                    end

                    aucs = [mv.auc for mv in mvals]
                    aces = [mv.ace for mv in mvals]

                    push!(split_metrics, (
                        auc_avg = mean(aucs),
                        auc_min = minimum(aucs),
                        ace_avg = mean(aces),
                        ace_max = maximum(aces),
                    ))
                end

                # average over splits
                auc_avg_mean = mean(getfield.(split_metrics, :auc_avg))
                auc_min_mean = mean(getfield.(split_metrics, :auc_min))
                ace_avg_mean = mean(getfield.(split_metrics, :ace_avg))
                ace_max_mean = mean(getfield.(split_metrics, :ace_max))

                push!(results, (
                    scenario     = scen.name,
                    theta        = theta,
                    method       = method,
                    auc_avg_mean = auc_avg_mean,
                    auc_min_mean = auc_min_mean,
                    ace_avg_mean = ace_avg_mean,
                    ace_max_mean = ace_max_mean,
                ))
            end
        end
    end
    return results
end
```

run_synthetic_UVR

In [93]:
```
res_uvr = run_synthetic_UVR(
    N = 3000,
    thetas = [0.5, 0.75, 0.9],
    w = 1.0,
    n_splits = 3,
    n_pert_per_split = 100,
    lambda_l2 = 0.01,
    seed = 2025,
)
foreach(println, res_uvr)
```

```
(scenario = :U, theta = 0.5, method = :dro, auc_avg_mean = 0.6369017126449
766, auc_min_mean = 0.5982223959083725, ace_avg_mean = 0.1052586840305108
6, ace_max_mean = 0.13648512982352676)
(scenario = :U, theta = 0.5, method = :logistic, auc_avg_mean = 0.67782414
81233609, auc_min_mean = 0.646467419975348, ace_avg_mean = 0.1064596626861
5104, ace_max_mean = 0.13705157976967552)
(scenario = :U, theta = 0.75, method = :dro, auc_avg_mean = 0.643134442169
4218, auc_min_mean = 0.6003904143220631, ace_avg_mean = 0.0544526856574553
34, ace_max_mean = 0.08504948145570464)
(scenario = :U, theta = 0.75, method = :logistic, auc_avg_mean = 0.6795266
58752378, auc_min_mean = 0.6450369959346806, ace_avg_mean = 0.100066682452
58081, ace_max_mean = 0.12841126480213508)
(scenario = :U, theta = 0.9, method = :dro, auc_avg_mean = 0.6738718272725
711, auc_min_mean = 0.6376848680042937, ace_avg_mean = 0.1217474648982071
5, ace_max_mean = 0.14979496785620589)
(scenario = :U, theta = 0.9, method = :logistic, auc_avg_mean = 0.67990188
42579884, auc_min_mean = 0.6485459954142211, ace_avg_mean = 0.105806317472
30236, ace_max_mean = 0.13676428970281215)
(scenario = :V, theta = 0.5, method = :dro, auc_avg_mean = 0.6489033043393
408, auc_min_mean = 0.6126344510490052, ace_avg_mean = 0.1176209680086084
8, ace_max_mean = 0.14465760741437364)
(scenario = :V, theta = 0.5, method = :logistic, auc_avg_mean = 0.68176057
0456975, auc_min_mean = 0.6520507755459092, ace_avg_mean = 0.1089024945960
9053, ace_max_mean = 0.13765093246668003)
(scenario = :V, theta = 0.75, method = :dro, auc_avg_mean = 0.652476438175
8364, auc_min_mean = 0.6127705087033446, ace_avg_mean = 0.1637115819556284
3, ace_max_mean = 0.19408163063056727)
(scenario = :V, theta = 0.75, method = :logistic, auc_avg_mean = 0.6690869
944919505, auc_min_mean = 0.6338598393608436, ace_avg_mean = 0.11547342307
60941, ace_max_mean = 0.1449506996548765)
(scenario = :V, theta = 0.9, method = :dro, auc_avg_mean = 0.6508075290759
426, auc_min_mean = 0.6195720811114772, ace_avg_mean = 0.2612878281711342,
ace_max_mean = 0.2954031485307242)
(scenario = :V, theta = 0.9, method = :logistic, auc_avg_mean = 0.66373709
52903664, auc_min_mean = 0.6305225439162515, ace_avg_mean = 0.124598500465
38871, ace_max_mean = 0.15221074707276644)
(scenario = :R, theta = 0.5, method = :dro, auc_avg_mean = 0.6478109398462
201, auc_min_mean = 0.6169224657208989, ace_avg_mean = 0.1124988843627217
1, ace_max_mean = 0.1406319234818605)
(scenario = :R, theta = 0.5, method = :logistic, auc_avg_mean = 0.69531659
6369218, auc_min_mean = 0.6552489433493119, ace_avg_mean = 0.0869355273722
3868, ace_max_mean = 0.12527254933759782)
(scenario = :R, theta = 0.75, method = :dro, auc_avg_mean = 0.631805362391
2702, auc_min_mean = 0.593463122354056, ace_avg_mean = 0.1015542186644509
1, ace_max_mean = 0.13764007608361709)
(scenario = :R, theta = 0.75, method = :logistic, auc_avg_mean = 0.7077750
451702617, auc_min_mean = 0.6773097190912557, ace_avg_mean = 0.07687866114
552354, ace_max_mean = 0.10425795074675649)
(scenario = :R, theta = 0.9, method = :dro, auc_avg_mean = 0.6585341836979
524, auc_min_mean = 0.623625140882094, ace_avg_mean = 0.13056722453192135,
ace_max_mean = 0.1559706987798008)
(scenario = :R, theta = 0.9, method = :logistic, auc_avg_mean = 0.70062344
51115724, auc_min_mean = 0.6644491733075465, ace_avg_mean = 0.077460024677
60983, ace_max_mean = 0.10477428508923033)
```

Real-World Data

In [69]:
```
using Statistics
"""
```

```
    summarize_experiment(res_vec)

Given the vector of per-split results returned by `run_experiment`,
compute simple means across splits.

Each element of `res_vec` is expected to have fields:
    (split, auc_avg, auc_min, ace_avg, ace_max)

Returns a NamedTuple:
    (auc_avg_mean, auc_min_mean, ace_avg_mean, ace_max_mean)
"""
function summarize_experiment(res_vec::Vector{<:NamedTuple})
    auc_avg_mean = mean(r.auc_avg for r in res_vec)
    auc_min_mean = mean(r.auc_min for r in res_vec)
    ace_avg_mean = mean(r.ace_avg for r in res_vec)
    ace_max_mean = mean(r.ace_max for r in res_vec)
    return (
        auc_avg_mean = auc_avg_mean,
        auc_min_mean = auc_min_mean,
        ace_avg_mean = ace_avg_mean,
        ace_max_mean = ace_max_mean,
    )
end
```

```
summarize_experiment
```

In [73]:
```
using Random
using Ipopt
using CSV, DataFrames   # harmless if already loaded

"""
    run_churn_scenarios(
        path;
        severities = [:mild, :strong],
        thetas = [0.5, 0.75, 0.9],
        train_ratio = 0.7,
        n_splits = 2,
        n_pert_per_split = 100,
        lambda_l2 = 0.01,
        seed = 2025,
        max_samples_per_scenario = nothing,
        optimizer = optimizer_with_attributes(Ipopt.Optimizer, "print_lev
    )

Run subgroup-DRO vs standard logistic regression on the Bank Customer
Churn dataset, under a grid of (severity, θ) settings.

For each (severity, θ):
  1. Optionally subsample at most `max_samples_per_scenario` points.
  2. Build DROParams and PerturbParams via `build_churn_params`.
  3. Run `run_experiment` with method = :dro.
  4. Run `run_experiment` with method = :logistic.
  5. Summarize each via `summarize_experiment`.

Inputs
------
- path                      : path to "Bank Customer Churn Prediction.csv".
- severities                : e.g. [:mild, :strong].
- thetas                    : robustness levels θ in (0,1).
- train_ratio               : fraction of samples used for training in each
```

```
- n_splits                 : number of random train/test splits.
- n_pert_per_split         : number of perturbed test sets per split.
- lambda_l2                : L2-regularization coefficient for logistic ba
- seed                     : base random seed.
- max_samples_per_scenario: if not `nothing`, limit each (severity, θ) sc
                            to this many samples by random subsampling.
- optimizer                : JuMP optimizer for both models.

Returns
-------
- results :: Vector{NamedTuple}

Each element has fields:
    (severity, theta, method,
     auc_avg_mean, auc_min_mean, ace_avg_mean, ace_max_mean)
"""
function run_churn_scenarios(
    path::AbstractString;
    severities = [:mild, :strong],
    thetas = [0.5, 0.75, 0.9],
    train_ratio::Float64 = 0.7,
    n_splits::Int = 2,
    n_pert_per_split::Int = 100,
    lambda_l2::Float64 = 0.01,
    seed::Int = 2025,
    max_samples_per_scenario::Union{Nothing,Int} = nothing,
    optimizer = optimizer_with_attributes(Ipopt.Optimizer, "print_level"
)
    # Load the full dataset once
    X_full, Z_full, g_full, y_full, meta = load_churn_dataset(path)
    N_full = size(X_full, 1)

    # Logistic baseline hyperparameter
    logit_params = LogitParams(lambda_l2)

    # One RNG is enough here; runs are unpaired but share the same distri
    rng = Random.MersenneTwister(seed)

    results = NamedTuple[]

    # Loop over (severity, theta)
    for sev in severities
        for θ in thetas
            # Decide which subset of samples to use for this (sev, θ) sce
            if max_samples_per_scenario === nothing || max_samples_per_sc
                # Use all samples
                X = X_full
                Z = Z_full
                g = g_full
                y = y_full
            else
                # Randomly choose a subset of size max_samples_per_scenar
                idx_sub = randperm(rng, N_full)[1:max_samples_per_scenari
                X = X_full[idx_sub, :]
                Z = Z_full[idx_sub, :]
                g = g_full[idx_sub]
                y = y_full[idx_sub]
            end

            n_x  = size(X, 2)
```

```
            m    = size(Z, 2)
            num_g = maximum(g)

            # Build DRO and perturbation parameters for this scenario
            dro_params, pert_params =
                build_churn_params(sev, θ; n_x = n_x, m = m, num_g = num_

            # Run DRO model on this (possibly subsampled) dataset
            res_dro = run_experiment(
                X, Z, g, y;
                train_ratio       = train_ratio,
                method            = :dro,
                dro_params        = dro_params,
                pert_params       = pert_params,
                n_splits          = n_splits,
                n_pert_per_split  = n_pert_per_split,
                rng               = rng,
                optimizer         = optimizer,
            )
            sum_dro = summarize_experiment(res_dro)

            push!(results, (
                severity      = sev,
                theta         = θ,
                method        = :dro,
                auc_avg_mean  = sum_dro.auc_avg_mean,
                auc_min_mean  = sum_dro.auc_min_mean,
                ace_avg_mean  = sum_dro.ace_avg_mean,
                ace_max_mean  = sum_dro.ace_max_mean,
            ))

            # Run logistic baseline on the same (possibly subsampled) dat
            res_log = run_experiment(
                X, Z, g, y;
                train_ratio       = train_ratio,
                method            = :logistic,
                logit_params      = logit_params,
                pert_params       = pert_params,
                n_splits          = n_splits,
                n_pert_per_split  = n_pert_per_split,
                rng               = rng,
                optimizer         = optimizer,
            )
            sum_log = summarize_experiment(res_log)

            push!(results, (
                severity      = sev,
                theta         = θ,
                method        = :logistic,
                auc_avg_mean  = sum_log.auc_avg_mean,
                auc_min_mean  = sum_log.auc_min_mean,
                ace_avg_mean  = sum_log.ace_avg_mean,
                ace_max_mean  = sum_log.ace_max_mean,
            ))
        end
    end

    return results
end
```

run_churn_scenarios

In [75]:
```
path = "Bank Customer Churn Prediction.csv"

results_churn = run_churn_scenarios(
    path;
    severities = [:mild, :strong],
    thetas = [0.5, 0.75, 0.9],
    train_ratio = 0.7,
    n_splits = 8,
    n_pert_per_split = 100,
    lambda_l2 = 0.01,
    seed = 2025,
    max_samples_per_scenario = 2000,   # <-- this is the cap
)

println.(results_churn)
```

```
(severity = :mild, theta = 0.5, method = :dro, auc_avg_mean = 0.6809890560473452, auc_min_mean = 0.664903204398336, ace_avg_mean = 0.06460177985489618, ace_max_mean = 0.08777094974627037)
(severity = :mild, theta = 0.5, method = :logistic, auc_avg_mean = 0.7517845869875566, auc_min_mean = 0.7278152044312919, ace_avg_mean = 0.04193727450301925, ace_max_mean = 0.0660311427357998)
(severity = :mild, theta = 0.75, method = :dro, auc_avg_mean = 0.7300406577070826, auc_min_mean = 0.7116409109201876, ace_avg_mean = 0.053757912590425276, ace_max_mean = 0.07300932894970448)
(severity = :mild, theta = 0.75, method = :logistic, auc_avg_mean = 0.7333472257349112, auc_min_mean = 0.7081167983791491, ace_avg_mean = 0.038713677706732764, ace_max_mean = 0.06103026321394643)
(severity = :mild, theta = 0.9, method = :dro, auc_avg_mean = 0.7527691053883292, auc_min_mean = 0.7287964524184268, ace_avg_mean = 0.060683069530205413, ace_max_mean = 0.08226691094058226)
(severity = :mild, theta = 0.9, method = :logistic, auc_avg_mean = 0.7435817374996072, auc_min_mean = 0.7174550020938562, ace_avg_mean = 0.04241667169189757, ace_max_mean = 0.06407911341649271)
(severity = :strong, theta = 0.5, method = :dro, auc_avg_mean = 0.641215088869021, auc_min_mean = 0.616288977723691, ace_avg_mean = 0.07629882130515442, ace_max_mean = 0.09647496631886289)
(severity = :strong, theta = 0.5, method = :logistic, auc_avg_mean = 0.709687278929923, auc_min_mean = 0.6693135594425055, ace_avg_mean = 0.04870220426575063, ace_max_mean = 0.07183562847552016)
(severity = :strong, theta = 0.75, method = :dro, auc_avg_mean = 0.6876614974775004, auc_min_mean = 0.6615166921501202, ace_avg_mean = 0.05839039397970805, ace_max_mean = 0.07868046922392939)
(severity = :strong, theta = 0.75, method = :logistic, auc_avg_mean = 0.6926513102271572, auc_min_mean = 0.6490075944510951, ace_avg_mean = 0.048570312242927335, ace_max_mean = 0.07465620969977059)
(severity = :strong, theta = 0.9, method = :dro, auc_avg_mean = 0.7065200817874847, auc_min_mean = 0.6754661556606002, ace_avg_mean = 0.044212101836086294, ace_max_mean = 0.06705531415980445)
(severity = :strong, theta = 0.9, method = :logistic, auc_avg_mean = 0.7054237888163174, auc_min_mean = 0.6677568197490297, ace_avg_mean = 0.04425064308727014, ace_max_mean = 0.06873497159443204)
```

```
12-element Vector{Nothing}:
 nothing
 nothing
 nothing
 nothing
 nothing
 nothing
 nothing
 nothing
 nothing
 nothing
 nothing
 nothing
```

In [76]:
```
using DataFrames

"""
    summarize_differences(results_churn)

Convert the vector of NamedTuples returned by `run_churn_scenarios`
into a DataFrame with one row per (severity, theta), and columns:

- auc_dro, auc_log, delta_auc = auc_dro - auc_log
- ace_dro, ace_log, delta_ace = ace_dro - ace_log
"""
function summarize_differences(results_churn)
    df = DataFrame(results_churn)

    # pivot: two rows per config -> one row per (severity, theta)
    configs = unique(df[:, [:severity, :theta]])
    out = DataFrame(
        severity = String[],
        theta     = Float64[],
        auc_dro   = Float64[],
        auc_log   = Float64[],
        delta_auc = Float64[],
        ace_dro   = Float64[],
        ace_log   = Float64[],
        delta_ace = Float64[],
    )

    for row in eachrow(configs)
        sev = row.severity
        θ   = row.theta

        sub = df[(df.severity .== sev) .& (df.theta .== θ), :]

        dro_row = sub[sub.method .== :dro, :]
        log_row = sub[sub.method .== :logistic, :]

        auc_dro = dro_row.auc_avg_mean[1]
        auc_log = log_row.auc_avg_mean[1]
        ace_dro = dro_row.ace_avg_mean[1]
        ace_log = log_row.ace_avg_mean[1]

        push!(out, (
            string(sev),
            θ,
            auc_dro,
            auc_log,
```

```
                auc_dro - auc_log,
                ace_dro,
                ace_log,
                ace_dro - ace_log,
            ))
    end

    return out
end

df_diff = summarize_differences(results_churn)
println(df_diff)
```

```
6×8 DataFrame
 Row │ severity  theta    auc_dro   auc_log   delta_auc     ace_dro    ace_
log    delta_ace
     │ String    Float64  Float64   Float64   Float64       Float64    Floa
t64    Float64
─────┼──────────────────────────────────────────────────────────────────────
───────────────────
   1 │ mild         0.5   0.680989  0.751785  -0.0707955    0.0646018  0.04
19373   0.0226645
   2 │ mild         0.75  0.730041  0.733347  -0.00330657   0.0537579  0.03
87137   0.0150442
   3 │ mild         0.9   0.752769  0.743582   0.00918737   0.0606831  0.04
24167   0.0182664
   4 │ strong       0.5   0.641215  0.709687  -0.0684722    0.0762988  0.04
87022   0.0275966
   5 │ strong       0.75  0.687661  0.692651  -0.00498981   0.0583904  0.04
85703   0.00982008
   6 │ strong       0.9   0.70652   0.705424   0.00109629   0.0442121  0.04
42506  -3.85413e-5
```

In [95]:

```
############################################################
# Churn: scenario-based DRO + perturb parameters (U / V / R)
############################################################

"""
    build_churn_scenario_params(scenario, theta;
                                n_x, m, num_g, w = 1.0)

Build (dro_params, pert_params) for the Bank Churn dataset
under one of three subgroup metric scenarios:

- :U (Uniform):   A_g = B_g = C_g = 1         for all groups.
- :V (Vulnerable): (A,B,C) = (0.5,0.5,0.5) for group 1,
                   (1,1,1)               for group 2,
                   (2,2,2)               for group 3.
- :R (Reversed):   same pattern as :V but swap group 1 and 3.

Here we keep the data-generating side (γ_x, δ) simple:

- γ_x[j] = 1     for all continuous features (already standardized).
- δ[ℓ]  = 1     for all categorical features.
- ε = -log(theta).

The perturbation object only carries (mode, w); the *metric*
(A_g · γ_x, B_g · δ, C_g) controls how large the actual shifts are
inside `perturb_testset`.
"""
```

```julia
function build_churn_scenario_params(
    scenario::Symbol,
    theta::Float64;
    n_x::Int,
    m::Int,
    num_g::Int,
    w::Float64 = 1.0,
)
    @assert num_g == 3 "This helper assumes 3 groups (France / Germany /

    # Base feature scales
    gamma_x = ones(Float64, n_x)
    delta   = ones(Float64, m)

    # Scenario-specific A_g, B_g, C_g
    A_group = zeros(Float64, num_g)
    B_group = zeros(Float64, num_g)
    C_group = zeros(Float64, num_g)

    if scenario == :U
        # All groups equally easy to move
        A_group .= 1.0
        B_group .= 1.0
        C_group .= 1.0

    elseif scenario == :V
        # Group 1 most vulnerable, group 3 most robust
        A_group .= [0.5, 1.0, 2.0]
        B_group .= [0.5, 1.0, 2.0]
        C_group .= [0.5, 1.0, 2.0]

    elseif scenario == :R
        # Reverse: group 3 most vulnerable, group 1 most robust
        A_group .= [2.0, 1.0, 0.5]
        B_group .= [2.0, 1.0, 0.5]
        C_group .= [2.0, 1.0, 0.5]

    else
        error("Unknown scenario = $scenario. Use :U, :V or :R.")
    end

    # DRO radius
    epsilon = -log(theta)

    dro_params = DROParams(
        delta,
        A_group,
        B_group,
        C_group,
        gamma_x,
        epsilon,
    )

    # Perturbation knob w: global noise budget
    pert_params = PerturbParams(scenario, w)

    return dro_params, pert_params
end
```

build_churn_scenario_params

In [100…

```
############################################################
# Churn experiment driver (U / V / R, 1000-sample subset)
############################################################

using Random
using Statistics
using Ipopt

"""
    run_churn_scenarios_uvr(
        path;
        scenarios = [:U, :V, :R],
        thetas = [0.5, 0.75, 0.9],
        train_ratio = 0.7,
        n_splits = 5,
        n_pert_per_split = 100,
        lambda_l2 = 1e-2,
        max_samples = 1000,
        w = 1.0,
        seed = 2025,
        optimizer = optimizer_with_attributes(Ipopt.Optimizer, "print_lev
    )

Apply the same U / V / R subgroup-metric pipeline to the
Bank Customer Churn dataset, using only a random subset of
at most `max_samples` points for speed.

For each scenario ∈ {U,V,R} and θ ∈ {0.5,0.75,0.9}:

1. Build (dro_params, pert_params) via `build_churn_scenario_params`.
2. Run `run_experiment` with method = :dro.
3. Run `run_experiment` with method = :logistic (same perturbation law).
4. Aggregate per-split summaries into means over splits.

Returns a vector of NamedTuples with fields:
    (scenario, theta, method, auc_avg_mean, auc_min_mean,
     ace_avg_mean, ace_max_mean)
"""
function run_churn_scenarios_uvr(
    path::AbstractString;
    scenarios = [:U, :V, :R],
    thetas = [0.5, 0.75, 0.9],
    train_ratio::Float64 = 0.7,
    n_splits::Int = 5,
    n_pert_per_split::Int = 100,
    lambda_l2::Float64 = 1e-2,
    max_samples::Int = 1000,
    w::Float64 = 1.0,
    seed::Int = 2025,
    optimizer = optimizer_with_attributes(Ipopt.Optimizer, "print_level"
)
    rng = MersenneTwister(seed)

    # 1) Load and subsample dataset
    X_full, Z_full, g_full, y_full, meta = load_churn_dataset(path; stand
    N_total = size(X_full, 1)
    N_use   = min(max_samples, N_total)

    idx_sub = randperm(rng, N_total)[1:N_use]
```

```
    X = X_full[idx_sub, :]
    Z = Z_full[idx_sub, :]
    g = g_full[idx_sub]
    y = y_full[idx_sub]

    n_x  = size(X, 2)
    m    = size(Z, 2)
    num_g = maximum(g)

    logit_params = LogitParams(lambda_l2)

    results = NamedTuple[]

    for scen in scenarios
        println(scen)
        for θ in thetas
            println(θ)
            # 2) Build DRO + perturbation parameters for this scenario
            dro_params, pert_params = build_churn_scenario_params(
                scen, θ;
                n_x = n_x,
                m = m,
                num_g = num_g,
                w = w,
            )

            # 3) DRO model
            res_dro = run_experiment(
                X, Z, g, y;
                train_ratio = train_ratio,
                method = :dro,
                dro_params = dro_params,
                logit_params = logit_params,
                pert_params = pert_params,
                n_splits = n_splits,
                n_pert_per_split = n_pert_per_split,
                rng = rng,
                optimizer = optimizer,
            )

            # 4) Logistic baseline
            res_logit = run_experiment(
                X, Z, g, y;
                train_ratio = train_ratio,
                method = :logistic,
                dro_params = dro_params,
                logit_params = logit_params,
                pert_params = pert_params,
                n_splits = n_splits,
                n_pert_per_split = n_pert_per_split,
                rng = rng,
                optimizer = optimizer,
            )

            # 5) Average the per-split summaries
            for (method, res) in zip((:dro, :logistic), (res_dro, res_log
                auc_avg_mean = mean(r.auc_avg for r in res)
                auc_min_mean = mean(r.auc_min for r in res)
                ace_avg_mean = mean(r.ace_avg for r in res)
                ace_max_mean = mean(r.ace_max for r in res)
```

```
                push!(results, (
                    scenario     = scen,
                    theta        = θ,
                    method       = method,
                    auc_avg_mean = auc_avg_mean,
                    auc_min_mean = auc_min_mean,
                    ace_avg_mean = ace_avg_mean,
                    ace_max_mean = ace_max_mean,
                ))
            end
        end
    end

    return results
end
```

```
run_churn_scenarios_uvr
```

In [101…

```
path = "Bank Customer Churn Prediction.csv"

results_churn_uvr = run_churn_scenarios_uvr(
    path;
    scenarios = [:U, :V, :R],
    thetas = [0.5, 0.75, 0.9],
    train_ratio = 0.7,
    n_splits = 5,
    n_pert_per_split = 100,
    lambda_l2 = 1e-2,
    max_samples = 1000,
    w = 1.0,
    seed = 2025,
)
```

```
U
0.5
0.75
0.9
V
0.5
0.75
0.9
R
0.5
0.75
0.9
```

```
18-element Vector{NamedTuple}:
 (scenario = :U, theta = 0.5, method = :dro, auc_avg_mean = 0.6006728884961647, auc_min_mean = 0.5372019670609568, ace_avg_mean = 0.08502026522647321, ace_max_mean = 0.11759724874090113)
 (scenario = :U, theta = 0.5, method = :logistic, auc_avg_mean = 0.6218250865737012, auc_min_mean = 0.5501126897822066, ace_avg_mean = 0.09438246192760315, ace_max_mean = 0.131438654906637)
 (scenario = :U, theta = 0.75, method = :dro, auc_avg_mean = 0.5819279142389265, auc_min_mean = 0.5203608841970363, ace_avg_mean = 0.06150629296684945, ace_max_mean = 0.0980818336640321)
 (scenario = :U, theta = 0.75, method = :logistic, auc_avg_mean = 0.6400344725704978, auc_min_mean = 0.5679586614900688, ace_avg_mean = 0.08565027883003404, ace_max_mean = 0.12288719969875483)
 (scenario = :U, theta = 0.9, method = :dro, auc_avg_mean = 0.608438816389667, auc_min_mean = 0.5313931089558744, ace_avg_mean = 0.0634777225748133 6, ace_max_mean = 0.1044129528383747)
 (scenario = :U, theta = 0.9, method = :logistic, auc_avg_mean = 0.62891127677067, auc_min_mean = 0.567954691822777, ace_avg_mean = 0.08911254339202858, ace_max_mean = 0.1321977006832086)
 (scenario = :V, theta = 0.5, method = :dro, auc_avg_mean = 0.5885027594093508, auc_min_mean = 0.5187536229290397, ace_avg_mean = 0.07336313120985193, ace_max_mean = 0.10997329719691713)
 (scenario = :V, theta = 0.5, method = :logistic, auc_avg_mean = 0.6245864297487123, auc_min_mean = 0.5573429388070347, ace_avg_mean = 0.09567139063447101, ace_max_mean = 0.13412318329783363)
 (scenario = :V, theta = 0.75, method = :dro, auc_avg_mean = 0.5924007011684548, auc_min_mean = 0.5263368631427828, ace_avg_mean = 0.07969990426576971, ace_max_mean = 0.1167856060440502)
 (scenario = :V, theta = 0.75, method = :logistic, auc_avg_mean = 0.6287110445119499, auc_min_mean = 0.5583531649786219, ace_avg_mean = 0.09500999803440499, ace_max_mean = 0.13367615562858454)
 (scenario = :V, theta = 0.9, method = :dro, auc_avg_mean = 0.572953241281928, auc_min_mean = 0.5074456172918075, ace_avg_mean = 0.0649898930305766 6, ace_max_mean = 0.10801462505590768)
 (scenario = :V, theta = 0.9, method = :logistic, auc_avg_mean = 0.6141023333814848, auc_min_mean = 0.5388785482171677, ace_avg_mean = 0.09847893095235506, ace_max_mean = 0.14231754452958617)
 (scenario = :R, theta = 0.5, method = :dro, auc_avg_mean = 0.5894690644829353, auc_min_mean = 0.5249913056751174, ace_avg_mean = 0.07413790833028004, ace_max_mean = 0.11579876071421318)
 (scenario = :R, theta = 0.5, method = :logistic, auc_avg_mean = 0.6323692667771349, auc_min_mean = 0.5556010820839283, ace_avg_mean = 0.08957496391661526, ace_max_mean = 0.13482479317835475)
 (scenario = :R, theta = 0.75, method = :dro, auc_avg_mean = 0.6062527185965222, auc_min_mean = 0.5313759589766944, ace_avg_mean = 0.07797536104686764, ace_max_mean = 0.10949653105602628)
 (scenario = :R, theta = 0.75, method = :logistic, auc_avg_mean = 0.6338214173352161, auc_min_mean = 0.5647782646969348, ace_avg_mean = 0.09131558342989252, ace_max_mean = 0.13443504653581545)
 (scenario = :R, theta = 0.9, method = :dro, auc_avg_mean = 0.5964059067748367, auc_min_mean = 0.5253347857068538, ace_avg_mean = 0.05887231063912492 4, ace_max_mean = 0.09415360015661693)
 (scenario = :R, theta = 0.9, method = :logistic, auc_avg_mean = 0.6264880687261054, auc_min_mean = 0.556192388761948, ace_avg_mean = 0.09235123171819284, ace_max_mean = 0.13546810519900426)
```

In [ ]: