Robust Logistic Regression

December 3, 2020

[370]:
```
using Pkg
Pkg.add(PackageSpec(path="https://github.com/diegozea/ROC.jl"))
```

```
   Cloning git-repo `https://github.com/diegozea/ROC.jl`
  Updating git-repo
`https://github.com/diegozea/ROC.jl`.0 %46.3 %>  ]  93.1 %
[1mFetching: [========================================>]
100.0 %.0 % Resolving package versions…
 Installed Infinity  v0.2.3
  Updating `~/.julia/environments/v1.0/Project.toml`
  [e4f92426] + ROC v0.1.0 #master
(https://github.com/diegozea/ROC.jl)
  Updating `~/.julia/environments/v1.0/Manifest.toml`
  [a303e19e] + Infinity v0.2.3
  [e4f92426] + ROC v0.1.0 #master
(https://github.com/diegozea/ROC.jl)
```

[391]:
```
using JuMP, Gurobi, CSV, LinearAlgebra, DataFrames, Random, Distributions,␣
 ↪Statistics,MLBase, CPUTime,ScikitLearn ,MLDataUtils
@sk_import metrics: roc_auc_score
gurobi_env = Gurobi.Env()
```

```
Academic license - for non-commercial use only
```

[391]:
```
Gurobi.Env(Ptr{Nothing} @0x00007fa9e453c000)
```

[429]:
```
function one_hot_encode(X, names)
    X2 = deepcopy(X)
    select!(X2, Not(Symbol.(names)))
    for i in names
        vales = unique(X[i])
        for j in 1:length(vales)-1
           X2[Symbol(string(i)*"_"*string(vales[j]))] = (X[i].==vales[j])*1
        end
    end
    return X2
end
```

```
function normalize(X, names)
    X2 = deepcopy(X)
    select!(X2, Not(Symbol.(names)))
    for j in names
        X2[j] = (X[:,j] .- mean(X[:,j])) / std(X[:,j])
    end
    return X2
end

function clean(X)
    n,p = size(X)
    X2 = deepcopy(X)
    i = 0
    while i < n
        i += 1
        if "?" in X[i,:]
            X = X[1:end .!= i, :]
            i -= 1
        end
        n,_ = size(X)
    end
    return X
end

function toNum(df, names)
    n,p = size(df)
    for name in names
        if !(isa(df[1,name], Int64) || isa(df[1,name], Float64))
            temp = zeros(n)
            for i=1:n
                temp[i] = parse(Float64,df[i,name])
            end
            df[!,name] = temp
        end
    end
    return df
end

function preprocess(df, categorical_vars, numerical_vars)
    df = clean(df)
    df = toNum(df,numerical_vars)
    df = normalize(df,numerical_vars)
    df = one_hot_encode(df[:,1:end], categorical_vars)
    df[df[:,end].==0,end] .= -1
    return df
end
```

[429]: preprocess (generic function with 1 method)

[700]:
```
### Utils Functions ###
function compute_ f(w_k, y, X,  )
    n, p = size(X)
    temp = zeros(p)
    for i in 1:n
         t = min(exp(-y[i]*(transpose(w_k)*Array(X[i,:
 ↪]))+ *transpose(w_k)*w_k),100000)
         Δ = (1/(1+t))*t*(-y[i]*Array(X[i,:]) .+ 2* *w_k)
#         Δ = (-1/(1+exp(y[i]*dot(w_k,X[i,:])+ *transpose(w_k)*w_k)))*(y[i].
 ↪*X[i,:] .+ 2* *w_k)
        temp = temp + Δ
#          if i >= n-5
#              println("Δ ", Δ)
#              println(exp(-y[i]*(transpose(w_k)*Array(X[i,:
 ↪]))+ *transpose(w_k)*w_k))
# #              for i in 1:5
# #                  println("y ",y[i])
# #                  println("X ",X[i,:])
# #                  println("w ",w_k[:])
# #                  #println(log(1+exp(-y[i]*dot(X[i,:], w_k))))
# #                  println()
# #              end
#          end
#          if i == n
#              println("temp",temp)
#          end
    end
#     for i in 1:5
#          println("y ",y[i])
#          println("X ",X[i,:])
#          println("w ",w_k[:])
#          #println(log(1+exp(-y[i]*dot(X[i,:], w_k))))
#          println()
#     end
#     println("temp",temp)
     f_k = temp
    #println(" f_k", f_k)
    return  f_k
end
```

[700]: compute_ f (generic function with 1 method)

[701]:
```
### Cutting Planes Implementation ###
function LR_cutting_planes(y, X,  ,  )
    errors = []
```

```
    n, p = size(X)
    # Initialization values and step 0
    w_0 = [0 for i in 1:p]
    #w_0 = [rand(Uniform(-0.5, 0.5)) for i in 1:p]
    f_0 = sum(log(1+exp(-y[i]*dot(X[i,:], w_0)+ *transpose(w_0)*w_0)) for i=1:n)
     f_0 = compute_ f(w_0, y, X,  )

    # Outer minimization problem
    outer_min_model = Model(solver=GurobiSolver(OutputFlag=0, gurobi_env))
    @variable(outer_min_model, t >= 0)
    @variable(outer_min_model, w[1:p])
    #@constraint(outer_min_model, [j=1:p], -1 <= w[j] <= 1)
    @constraint(outer_min_model, t >= f_0 + (dot( f_0, w)-dot( f_0, w_0)))
    @constraint(outer_min_model, [j=1:p], 10 >= w[j])
    @constraint(outer_min_model, [j=1:p], w[j] >= -10)
    @objective(outer_min_model, Min, t)
    k = 1 # Number of constraints in the final problem
    solve(outer_min_model)

    # New steps k
    t_k = getvalue(t)
    w_k = getvalue(w)
    f_k = sum(min(log(1+exp(-y[i]*dot(X[i,:], w_k)+ *transpose(w_k)*w_k)),100)␣
↪for i=1:n)
     f_k = compute_ f(w_k, y, X,  )

    while abs(f_k - t_k) >=   # error

        push!(errors, f_k - t_k)
        @constraint(outer_min_model,t >= f_k +(dot( f_k, w)-dot( f_k, w_k)))
        k += 1
        solve(outer_min_model)
        # Updating all the values
        t_k = getvalue(t)
        w_k = getvalue(w)
        f_k = sum(min(log(1+exp(-y[i]*dot(X[i,:],␣
↪w_k)+ *transpose(w_k)*w_k)),1000) for i=1:n)

         f_k = compute_ f(w_k, y, X,  )
         if k%500 == 0
             println("Number of constraints: ", k, "\t Error = ", abs(t_k -␣
↪f_k))
#              println("f",f_k)
#               println("w",w_k)
#               println(" f_k", f_k)
         end
        if k > 20000
```

```
            break
        end
    end
    push!(errors, f_k - t_k)
    return t_k, f_k, w_k, errors
end
```

[701]: LR_cutting_planes (generic function with 1 method)

[702]:
```
function robust_LG_valid(X, y,  , lambda_vals; method=LR_cutting_planes,␣
↪split_at=0.8)
    n,p = size(X)
    split = convert(Int,floor(split_at*n))
    permuted_indices = randperm(n)
    train_indices, valid_indices = permuted_indices[1:split],␣
↪permuted_indices[split+1:end]
    X_train, y_train = X[train_indices,:], y[train_indices]
    X_valid, y_valid = X[valid_indices,:], y[valid_indices]

    accuracies = zeros(length(rho_vals))
    for (i, ) in enumerate(lambda_vals)
        println(i)
        t, f, w, e = method(y_train, X_train,  ,  )
        pred = 1 ./ (1 .+ exp.(-(Matrix(X_valid)*w).+ *transpose(w)*w)) .> 0.5
        accuracies[i] = 1-sum(pred .!= (y_valid .== 1))/length(y_valid)
    end
    IJulia.clear_output()

    i_best = argmax(accuracies)
    t, f, w_best, e = method(y, X,  ,rho_vals[i_best])
    return w_best, lambda_vals[i_best], accuracies
end
```

[702]: robust_LG_valid (generic function with 1 method)

[703]:
```
function test(df, categorical_vars, numerical_vars, test_split,␣
↪validation_split,  , lambda_vals, seed)
    Random.seed!(seed)
    start = time()
    n,_ = size(df)
    permuted_indices = randperm(n)
    test_split = convert(Int,floor(validation_split*n))
    train_indices, test_indices = permuted_indices[1:test_split],␣
↪permuted_indices[test_split+1:end]
    train = df[train_indices,:]
    test = df[test_indices,:]
    train_X = train[:,1:end-1]
```

```
    train_y = train[:,end]
    test_X = test[:,1:end-1]
    test_y = test[:,end]

    IJulia.clear_output()
    println("Enter cross-validation")
    w,  , errors  = robust_LG_valid(train_X, train_y,  , lambda_vals;␣
 ↪method=LR_cutting_planes, split_at=validation_split)
    elapsed = time() - start
    pred_prob = 1 ./ (1 .+ exp.(-(Matrix(test_X)*w).+ *transpose(w)*w))
    pred = pred_prob.> 0.5
    accuracy = 1-sum(pred .!= (test_y .== 1))/length(test_y)
    auc = roc_auc_score(test_y ,pred_prob)
    IJulia.clear_output()

    return auc, accuracy,  , elapsed, w
end
```

[703]: test (generic function with 3 methods)

[682]:
```
df = CSV.read("Data/caesarian.csv";header=true)
size(df)
```

[682]: (80, 6)

[683]:
```
lambda_vals = [0.00001,0.0001,0.0005,0.001,0.005,0.01,0.05,0.1]
categorical_vars = Symbol.(["Delivery number" ;"Delivery time";"Blood of␣
 ↪Pressure";"Heart Problem"])
numerical_vars = Symbol.(["Age"])
df = preprocess(df, categorical_vars, numerical_vars)
n,p=size(df)
```

[683]: (80, 10)

[684]:
```
seed = 1
auc, acc, lambda, elapsed, w = test(df,categorical_vars,numerical_vars,0.75,0.
 ↪75,0.0001, lambda_vals, 1)
print("AUC: ",auc," Accuracy: ",acc, "  : ", lambda, " Time: ", elapsed)
```

```
AUC: 0.7575757575757576 Accuracy: 0.6  : 1.0e-5 Time: 2.2049567699432373
```

[685]:
```
w
```

[685]:
```
9-element Array{Float64,1}:
 -1.2491864081369384
 -0.3937010893346649
  0.8600389649568139
```

```
-0.3205883639048049
-0.8178133645505835
 0.6018132119061007
 1.3387212023583488
 0.416461190330931
 1.500084987129225
```

[542]:
```
seed = 2
auc, acc, lambda, elapsed = test(df,categorical_vars,numerical_vars,0.75,0.75,0.
↪0001, lambda_vals, seed)
print("AUC: ",auc," Accuracy: ",acc, "  : ", lambda, " Time: ", elapsed)
```

```
AUC: 0.7637362637362638 Accuracy: 0.7  : 1.0e-5 Time: 1.3622229099273682
```

[543]:
```
seed = 3
auc, acc, lambda, elapsed = test(df,categorical_vars,numerical_vars,0.75,0.75,0.
↪0001, lambda_vals, seed)
print("AUC: ",auc," Accuracy: ",acc, "  : ", lambda, " Time: ", elapsed)
```

```
AUC: 0.43499999999999994 Accuracy: 0.44999999999999996  : 0.05 Time:
1.4523191452026367
```

[544]:
```
seed = 4
auc, acc, lambda, elapsed = test(df,categorical_vars,numerical_vars,0.75,0.75,0.
↪0001, lambda_vals, seed)
print("AUC: ",auc," Accuracy: ",acc, "  : ", lambda, " Time: ", elapsed)
```

```
AUC: 0.609375 Accuracy: 0.5  : 1.0e-5 Time: 1.463482141494751
```

[692]:
```
df = CSV.read("Data/monks-1.test";header=false)[:,2:end]
select!(df, Not(Symbol.("Column9")))
df[!, 1], df[!, end] = df[!, end], df[!, 1]
size(df)
```

[692]:
```
(432, 7)
```

[693]:
```
lambda_vals = [0.0001,0.0005,0.001,0.005,0.01,0.05,0.1,0.5]
categorical_vars = propertynames(df)
numerical_vars = []
df = preprocess(df, categorical_vars, numerical_vars)
n,p=size(df)
```

[693]:
```
(432, 12)
```

[694]:
```
seed = 1
auc, acc, lambda, elapsed,w = test(df,categorical_vars,numerical_vars,0.75,0.
↪75,0.0001, lambda_vals, seed)
```

```
print("AUC: ",auc," Accuracy: ",acc, "  : ", lambda, " Time: ", elapsed)
```

```
AUC: 0.7124137931034483 Accuracy: 0.8055555555555556  : 0.0001 Time:
14.171710014343262
```

[695]:
```
w
```

[695]:
```
11-element Array{Float64,1}:
 -0.19921007152857848
 -0.3393916056922839
 -0.23309065923756547
 -0.3933468941032062
 -0.10828160207035857
  0.12567416466113626
 -0.12158299942954598
 -0.13455729769572078
 20.0
 -0.11027509123696333
 -0.17074361022605233
```

[613]:
```
seed = 2
auc, acc, lambda, elapsed = test(df,categorical_vars,numerical_vars,0.75,0.75,0.
↪0001, lambda_vals, seed)
print("AUC: ",auc," Accuracy: ",acc, "  : ", lambda, " Time: ", elapsed)
```

```
AUC: 0.7596153846153846 Accuracy: 0.7777777777777778  : 0.1 Time:
11.854342937469482
```

[614]:
```
w
```

[614]:
```
11-element Array{Float64,1}:
 -0.19967330500412425
 -0.34063384525125345
 -0.23428409783236667
 -0.3926567265808357
 -0.1083289833942036
  0.1260091038918896
 -0.1214634688072341
 -0.13265833242863104
 10.0
 -0.11065600270723822
 -0.17023159739032653
```

[549]:
```
seed = 3
auc, acc, lambda, elapsed = test(df,categorical_vars,numerical_vars,0.75,0.75,0.
↪0001, lambda_vals, seed)
print("AUC: ",auc," Accuracy: ",acc, "  : ", lambda, " Time: ", elapsed)
```

```
AUC: 0.6411663807890222 Accuracy: 0.7222222222222222  : 0.0001 Time:
11.339162111282349
```

[704]:
```
df = CSV.read("Data/credit-screening/crx.data";header=false)
n,p=size(df)
```

[704]:
```
(690, 16)
```

[705]:
```
categorical_vars = propertynames(df[1,vcat(1,4:7,9:10,12:13,16)])
numerical_vars = propertynames(df[1,vcat(2:3,8,11,14:15)])
lambda_vals = [0.0001,0.0005,0.001,0.005,0.01]
df = preprocess(df, categorical_vars, numerical_vars)
n,p=size(df)
```

[705]:
```
(653, 38)
```

[706]:
```
seed = 1
auc, acc, lambda, elapsed,w = test(df,categorical_vars,numerical_vars,0.75,0.
↪75,0.0001, lambda_vals, seed)
print("AUC: ",auc," Accuracy: ",acc, "  : ", lambda, " Time: ", elapsed)
```

```
AUC: 0.9172988249293471 Accuracy: 0.8780487804878049  : 0.0001 Time:
332.22172808647156
```

[ ]:

[564]:
```
seed = 2
auc, acc, lambda, elapsed,w = test(df,categorical_vars,numerical_vars,0.75,0.
↪75,0.0001, lambda_vals, seed)
print("AUC: ",auc," Accuracy: ",acc, "  : ", lambda, " Time: ", elapsed)
```

```
AUC: 0.958049371497804 Accuracy: 0.8841463414634146  : 0.005 Time:
467.2927370071411
```

[565]:
```
seed = 3
auc, acc, lambda, elapsed,w = test(df,categorical_vars,numerical_vars,0.75,0.
↪75,0.0001, lambda_vals, seed)
print("AUC: ",auc," Accuracy: ",acc, "  : ", lambda, " Time: ", elapsed)
```

```
AUC: 0.917572463768116 Accuracy: 0.8536585365853658  : 0.01 Time:
631.3079349994659
```

[675]:
```
df = CSV.read("Data/framingham.csv";header=true)
size(df)
```

[675]:
```
(3658, 16)
```

[676]:
```
categorical_vars = Symbol.(["education"])
numerical_vars = Symbol.(["age" ;"cigsPerDay";"totChol";"sysBP";"diaBP";"BMI";
 ↪"heartRate";"glucose"])
lambda_vals = [0.00001,0.0001,0.0005,0.001,0.005,0.01,0.05,0.1,0.2]
df = preprocess(df, categorical_vars, numerical_vars)
n,p=size(df)
```

[676]:
```
(3658, 18)
```

[678]:
```
seed = 1
auc, acc, lambda,elapsed,w = test(df,categorical_vars,numerical_vars, 0.75, 0.
 ↪75,0.0001, lambda_vals,seed)
print("AUC: ",auc," Accuracy: ",acc, "  : ", lambda, " Time: ", elapsed)
```

```
AUC: 0.8971686633756418 Accuracy: 0.8087431693989071  : 0.001 Time:
291.6356430053711
```

[ ]:

[578]:
```
seed = 2
auc, acc, lambda,elapsed = test(df,categorical_vars,numerical_vars, 0.75, 0.
 ↪75,0.0001, lambda_vals,seed)
print("AUC: ",auc," Accuracy: ",acc, "  : ", lambda, " Time: ", elapsed)
```

```
AUC: 0.8835259112323331 Accuracy: 0.8185792349726776  : 1.0e-5 Time:
288.1769390106201
```

[579]:
```
seed = 3
auc, acc, lambda,elapsed = test(df,categorical_vars,numerical_vars, 0.75, 0.
 ↪75,0.0001, lambda_vals,seed)
print("AUC: ",auc," Accuracy: ",acc, "  : ", lambda, " Time: ", elapsed)
```

```
AUC: 0.9008132903545748 Accuracy: 0.8360655737704918  : 1.0e-5 Time:
292.8686800003052
```

[696]:
```
df = CSV.read("Data/default of credit card clients.csv";header=true)
n,p = size(df)
```

[696]:
```
(30000, 25)
```

[598]:
```
propertynames(df[1,vcat(4:5)])
```

[598]:
```
2-element Array{Symbol,1}:
 :EDUCATION
 :MARRIAGE
```

[697]:
```
categorical_vars = propertynames(df[1,vcat(4:5)])
numerical_vars = propertynames(df[1,vcat([2,6],7:24)])
lambda_vals = [0.0001,0.001,0.01]
df = preprocess(df, categorical_vars, numerical_vars)
n,p=size(df)
```

[697]:
```
(30000, 32)
```

[698]:
```
seed = 1
auc, acc, lambda,elapsed = test(df,categorical_vars,numerical_vars, 0.75, 0.
↪75,0.001, lambda_vals,seed)
print("AUC: ",auc," Accuracy: ",acc, "  : ", lambda, " Time: ", elapsed)
```

```
Enter cross-validation
1
Number of constraints: 500       Error = 6234.1767934946165
w[-0.000631795, 10.3451, -20.0, 6.25596, 0.290806, -1.49541, 0.473608, 3.12466,
3.80476, 0.340599, -6.60304, -20.0, -20.0, 20.0, 20.0, -20.0, 20.0, 0.432675,
-9.31761, -20.0, -5.50421, -6.81292, -20.0, -20.0, -20.0, -20.0, -20.0, -20.0,
-20.0, -20.0, -20.0]
 f_k[1.64637e6, 609.756, -42.1734, 501.664, -48.031, -176.793, -158.085,
9.98766, -12.0211, -147.042, -284.085, -150.372, -65.343, 31.1591, 36.586,
-9.79421, 14.6661, 300.466, 101.046, -66.9913, -77.6335, -49.8635, -79.8858,
99.5099, 174.891, 14.7967, -3.27591, -2.77123, -3.27656, 218.28, 196.331]
Number of constraints: 1000      Error = 2885.396220048346
w[0.000142393, 6.73046, -1.07427, -7.97781, 3.0754, -0.822048, -1.88209,
1.10586, -0.672262, -5.78973, 4.95259, 7.80473, -20.0, 20.0, -4.44127, -20.0,
20.0, -1.51711, -20.0, -20.0, 1.93366, -2.6981, 3.43646, -20.0, -20.0, -20.0,
-20.0, -20.0, -20.0, -20.0, -20.0]
 f_k[4.90704e6, 545.433, 51.901, -105.673, 238.019, -69.5214, -58.648, -56.1077,
-116.172, -187.362, 91.3183, 94.6194, -13.5828, 13.5618, -96.4227, -40.7967,
59.9468, -55.3983, -58.3479, -74.1126, 296.59, 236.679, 286.79, 124.755,
52.6529, 96.9088, 2.69035, -2.66334, -0.919348, 188.862, 142.888]
Number of constraints: 1500      Error = 1857.3325591978273
w[3.08535e-5, 7.23235, 5.99148, -1.24141, 2.47553, 0.594211, 2.42951, -1.81272,
-1.06197, -0.672965, -0.962367, 3.82849, -20.0, 20.0, 9.8599, -20.0, 4.86377,
-0.469119, -20.0, -20.0, 2.55519, 0.218338, 0.269094, -20.0, -20.0, -20.0,
-20.0, -20.0, -20.0, -20.0, -20.0]
 f_k[2.31441e6, 263.476, 103.82, 27.3623, 141.521, 83.4762, 78.5545, -74.3489,
-93.4487, -128.168, -148.773, 85.6653, 24.9589, 57.6813, 25.6919, -76.6227,
-12.9196, -37.8148, -30.452, -35.0406, 20.0933, 216.023, 5.52494, 44.9731,
27.4421, 42.6631, 1.31916, -1.10035, -0.757956, 117.254, 64.0799]
Number of constraints: 2000      Error = 4245.184471340538
w[0.000109771, 6.0757, 13.595, -3.61503, 2.26843, -2.18283, -3.54975, 3.50255,
0.36961, 1.03106, -2.53112, -20.0, 20.0, 8.13606, -20.0, 20.0, -5.68132,
-6.55541, 3.63617, -20.0, -3.64692, 4.03021, 1.85278, -20.0, -20.0, -20.0,
-20.0, -20.0, -20.0, -20.0, -20.0]
```

```
 f_k[5.0264e6, 530.517, 244.658, 56.0847, 151.958, -84.368, -77.1329, 114.713,
102.753, -28.2902, -98.0965, 107.499, 156.777, 281.51, 87.7057, 130.44, 101.277,
27.9108, 434.148, -52.9864, 19.614, 491.794, 339.217, 115.55, 93.4292, 65.9512,
2.3931, -1.55008, -1.33918, 202.299, 156.406]
Number of constraints: 2500      Error = 2265.030762232425
w[0.000231498, 6.66393, -0.243097, -9.11364, -0.523121, -0.162036, -0.641187,
-2.32722, -4.59803, 3.35287, -0.849902, 6.46273, -20.0, 20.0, -20.0, 20.0,
-0.816883, 3.65857, -20.0, -20.0, -5.90664, 2.71778, -0.741181, -20.0, -20.0,
-20.0, -20.0, -20.0, -20.0, -20.0, -20.0]
 f_k[4.05161e6, 425.999, 38.7324, -132.473, -98.9666, -90.3502, -158.32,
-252.633, -269.916, -197.09, -199.661, -2.22483, -35.3654, -7.18867, -49.1304,
-23.9616, 5.02106, 186.169, -53.3318, -59.3075, -53.9424, 237.684, -36.9933,
85.8141, 75.9601, 39.3433, 1.807, -2.25382, -1.41262, 108.676, 154.097]
Number of constraints: 3000      Error = 1414.2292307365024
w[0.000655265, 2.00081, -1.2538, -5.05369, 2.9493, -2.69088, 0.128103, 3.64816,
-4.03011, 3.52184, -0.995579, 5.30197, -20.0, 2.33286, 6.03024, 1.80127,
2.12981, 1.9369, -4.27596, -1.92301, -3.37057, 0.360516, -13.8677, -20.0, -20.0,
-20.0, -20.0, -20.0, -20.0, -20.0, -20.0]
 f_k[2.29595e6, 88.4314, 10.5231, -65.3173, 105.554, -30.569, 77.6204, 146.557,
26.6749, 95.7146, 50.9971, -57.3124, -70.2836, -52.9497, -36.7918, -41.8017,
-38.8365, -24.959, -15.5867, -4.3441, -34.0073, -21.333, -35.5544, 4.04285,
-2.9781, 38.4317, -1.08959, -0.885363, -1.87808, 75.8964, 37.8837]


        InterruptException:



        Stacktrace:

         [1] iterate(::DataFrameRow{DataFrame,DataFrames.Index}, ::Int64) at /
↪Users/iai/builds/InterpretableAI/SysImgBuilder/.julia/packages/DataFrames/
↪uPgZV/src/dataframerow/dataframerow.jl:175

         [2] dot(::DataFrameRow{DataFrame,DataFrames.Index}, ::Array{Float64,1})␣
↪at /Users/julia/buildbot/worker/package_macos64/build/usr/share/julia/stdlib/
↪v1.0/LinearAlgebra/src/generic.jl:667

         [3] (::getfield(Main,␣
↪Symbol("##169#173")){Array{Int64,1},DataFrame,Float64})(::Int64) at ./none:0

         [4] iterate at ./generator.jl:47 [inlined]

         [5] mapfoldl_impl(::typeof(identity), ::typeof(Base.add_sum), ::
↪NamedTuple{(:init,),Tuple{Float64}}, ::Base.
↪Generator{UnitRange{Int64},getfield(Main,␣
↪Symbol("##169#173")){Array{Int64,1},DataFrame,Float64}}, ::Int64) at ./reduce.
↪jl:45
```

```
        [6] mapfoldl_impl(::Function, ::Function, ::NamedTuple{(),Tuple{}}, ::
↪Base.Generator{UnitRange{Int64},getfield(Main,␣
↪Symbol("##169#173")){Array{Int64,1},DataFrame,Float64}}) at ./reduce.jl:59

        [7] #mapfoldl#170 at ./reduce.jl:70 [inlined]

        [8] mapfoldl at ./reduce.jl:70 [inlined]

        [9] #mapreduce#174 at ./reduce.jl:203 [inlined]

        [10] mapreduce at ./reduce.jl:203 [inlined]

        [11] sum at ./reduce.jl:397 [inlined]

        [12] sum at ./reduce.jl:414 [inlined]

        [13] LR_cutting_planes(::Array{Int64,1}, ::DataFrame, ::Float64, ::
↪Float64) at ./In[691]:38

        [14] #robust_LG_valid#165(::typeof(LR_cutting_planes), ::Float64, ::
↪Function, ::DataFrame, ::Array{Int64,1}, ::Float64, ::Array{Float64,1}) at ./
↪In[680]:12

        [15] (::getfield(Main, Symbol("#kw##robust_LG_valid")))(::NamedTuple{(:
↪method, :split_at),Tuple{typeof(LR_cutting_planes),Float64}}, ::
↪typeof(robust_LG_valid), ::DataFrame, ::Array{Int64,1}, ::Float64, ::
↪Array{Float64,1}) at ./none:0

        [16] test(::DataFrame, ::Array{Symbol,1}, ::Array{Symbol,1}, ::Float64,␣
↪::Float64, ::Float64, ::Array{Float64,1}, ::Int64) at ./In[681]:17

        [17] top-level scope at In[698]:2
```

[ ]:

[ ]: