In [1]:
```
using JuMP, Gurobi
using LinearAlgebra
using Random
using DataFrames, CSV
using Statistics
Random.seed!(15095);
```

In [2]:
```
model = Model(Gurobi.Optimizer);
```

```
Academic license - for non-commercial use only - expires 2022-08-19
```

1.3 Question 3: Stable Regression

a)

$$\min_{\beta} \quad \sum_{i=1}^{n} |y_i - \beta_0 - \beta^T x_i| + \lambda \sum_{i=0}^{p} |\beta_i|$$

We rewrite the above problem as:

$$\begin{gathered}\min_{\beta,z,a} \quad t_i + \lambda \sum_{i=1}^{p} a_i \\ \text{s.t.} \\ \\ y - X\beta \leq t_i, \in i \\ -y + X\beta \leq t_i, \in i \\ \beta_j \leq a_j, \\ -\beta_j \leq a_j\end{gathered}$$

In [3]:
```
function a_regression(X, y, rho; solver_output=0)
    n,p = size(X)

    # Build model
    model = Model(Gurobi.Optimizer)
    set_optimizer_attribute(model, "OutputFlag", solver_output)

    # Insert variables
    @variable(model, beta[i=0:p])
    @variable(model, a[j=0:p]>=0)
    @variable(model, t[k=1:n]>=0)

    #Insert constraints
    @constraint(model,[j=0:p], beta[j]<=a[j])
    @constraint(model,[j=0:p], -beta[j]<=a[j])
    #you can expand the below constraint i
    @constraint(model, [k=1:n], y[k]-beta[0]-dot(beta[1:p],X[k,:]) <= t[k])
    @constraint(model, [k=1:n], -y[k]+beta[0]+dot(beta[1:p],X[k,:]) <= t[k]

    #Objective
    @objective(model,Min, sum(t[i] for i=1:n) + rho*sum(a[j] for j=0:p))

    # Optimize
    optimize!(model)

    # Return estimated betas
    return (value.(beta))
end
```

Out[3]:
```
a_regression (generic function with 1 method)
```

b)

In [4]:
```
trainx = CSV.read("stableX_train_and_valid.csv", DataFrame, header=0);
testx = CSV.read("stableX_test.csv", DataFrame, header=0);
trainy = CSV.read("stabley_train_and_valid.csv", DataFrame, header=0)[:,1];
testy = CSV.read("stabley_test.csv", DataFrame, header=0)[:,1];
```

In [5]:
```
function compute_mse(X, y, beta)
    n,p = size(X)
    return sum((y .- X*beta[1:p] .- beta[0]).^2)/n
end ;
```

In [6]:
```
lambda = [0.01, 0.03, 0.08, 0.1, 0.3, 0.8, 1, 3];
```

In [7]:
```
function robust_regression_valid(X, y, rho_vals; method=a_regression, split
    n,p = size(X)
    split = convert(Int,floor(split_at*n)) #floor takes the integer part

    #To create train and validation data, we will define the indices of eac
    permuted_indices = randperm(n)
    train_indices, valid_indices = permuted_indices[1:split], permuted_indi
    X_train, y_train = X[train_indices,:], y[train_indices]
    X_valid, y_valid = X[valid_indices,:], y[valid_indices]

    #we create an array to hold the results
    errors = zeros(length(rho_vals))

    for (i,rho) in enumerate(rho_vals)
        #get the beta coefficients from the Lasso or Ridge regression
        beta = method(X_train,y_train,rho,solver_output=solver_output)
        #compute the MSE with the optimal beta we just found
        errors[i] = compute_mse(Matrix(X_valid), y_valid, beta)
    end

    #get the best performing rho
    i_best = argmin(errors)
    beta_best = method(X,y,rho_vals[i_best])
    return beta_best, rho_vals[i_best], errors
end;
```

In [8]:
```
op_beta, op_lambda, err = robust_regression_valid(trainx, trainy, lambda; m
print(" Optimal Lambda = ", op_lambda)
```

```
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
 Optimal Lambda = 0.01
```

In [9]:
```
mse_list = zeros(5)

for i=1:5
    op_beta, op_lambda, err = robust_regression_valid(trainx, trainy, lambd
    mse = compute_mse(Matrix(testx),testy,op_beta);
    mse_list[i] = mse;
end

print(mse_list)
print("\nMSE Range: ", minimum(mse_list), " - ", maximum(mse_list))
```

```
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19
```

```
[4.887885853459812, 4.889140607362017, 4.887885853459812, 5.0241228592857
39, 4.889140607362017]
MSE Range: 4.887885853459812 - 5.024122859285739
```

c)

$$\min_{\beta} \max_{z \in Z} \quad \sum_{i=1}^{n} z_i |y_i - \beta_0 - \beta^T x_i| + \lambda \sum_{i=0}^{p} |\beta_i|$$

where $Z = \{z : \sum_{i=1}^{n} z_i = k, z_i = \{0, 1\}\}$

We rewrite the above problem as:

$$\min_{\beta,z,a} \quad \sum_{i=1}^{n} z_i t_i + \lambda \sum_{i=1}^{p} a_i$$
$$\text{s.t.}$$
$$\sum_{i=1}^{n} z_i = k$$
$$\sum_{i=1}^{p} \delta_i = s$$
$$|\beta_i| \leq M\delta_i, \delta_i \in 0, 1, \forall i \in [p], 0 \leq z_i \leq 1, \forall i \in [n]$$
$$y - X\beta \leq t_i, \in i$$
$$-y + X\beta \leq t_i, \in i$$
$$\beta_j \leq a_j,$$
$$-\beta_j \leq a_j$$

δ_i represents which coefficients are nonzero When $z_i = 1$, the point (x_i, y_i) is assigned to the training set, otherwise, it is assigned to the testing set.

We reformulate this by introducing the dual variable θ for the first constraint and u_i for the second set of constraints to arrive at:

$$\min_{\theta,u} \quad k\theta + \sum_{i=1}^{n} u_i$$
$$\text{s.t.}$$
$$\theta + u_i \geq |y_i - \beta^T x_i|$$
$$u_i \geq 0, \forall i \in [n]$$

Then, I substitute this minimization problem back into the outer minimization we arrive at the following problem:

$$\min_{\beta,\theta,u} \quad k\theta + \sum_{i=1}^{n} u_i + \lambda \sum_{i=1}^{p} a_i$$
$$\text{s.t.}$$

$$\theta + u_i \geq y_i - \beta_0 - \beta^T x_i$$
$$\theta + u_i \geq -(y_i - \beta_0 - \beta^T x_i)$$
$$u_i \geq 0, \forall i \in [n]$$
$$\beta_j \leq a_j,$$
$$-\beta_j \leq a_j$$

d)

In [10]:
```
function d_regression(X,y,rho;split_at=0.7,solver_output=0)

    n,p = size(X)

    # Build model
    model = Model(Gurobi.Optimizer)
    set_optimizer_attribute(model, "OutputFlag", solver_output)

    # Insert variables
    @variable(model,beta[i=0:p])
    @variable(model,theta)
    @variable(model,u[k=1:n]>=0)
    @variable(model,a[j=0:p]>=0)

    #Insert constraints
    @constraint(model,[i=1:n], theta + u[i] >= y[i] - beta[0] - dot(beta[1:
    @constraint(model,[i=1:n], theta + u[i] >= -(y[i] - beta[0] - dot(beta[
    @constraint(model,[j=0:p], beta[j]<=a[j])
    @constraint(model,[j=0:p], -beta[j]<=a[j])

    k = convert(Int,floor(split_at*n))

    #Objective
    @objective(model,Min, k*theta + sum(u[i] for i=1:n) + rho*sum(a[i] for

    # Optimize
    optimize!(model)

    # Return estimated betas
    return (value.(beta), value.(u))

end
```

Out[10]:
```
d_regression (generic function with 1 method)
```

In [11]:
```
#d_regression(trainx,trainy,0.01)
```

In [16]:
```
err = zeros(8)
for (i,lambda) in enumerate(lambda)
    #get the beta coefficients, and dual var from the Lasso or Ridge regres
    (beta, u) = d_regression(trainx,trainy,lambda,solver_output=0)
    val_ind = (u .== 0)
    xval = trainx[val_ind,:]
    yval = trainy[val_ind,:]
    #compute the MSE with the optimal beta and "most difficult" validation
    print("\n For lambda = ", lambda, ", MSE =  ", compute_mse(Matrix(xval)
    err[i] = compute_mse(Matrix(xval), yval, beta)
end

#get the best performing lambda
i_best = argmin(err);
op_beta = a_regression(trainx,trainy,lambda[i_best]);
print("\n \n \n Best Beta vector = \n", a_regression(trainx,trainy,lambda[i
print("Best Lambda = ", lambda[i_best], "\n")
print("Best MSE = ", err[i_best])
print("\nNew MSE on test data = ", compute_mse(Matrix(testx), testy, op_bet
```

```
Academic license - for non-commercial use only - expires 2022-08-19

 For lambda = 0.01, MSE =  0.13400726366428356
Academic license - for non-commercial use only - expires 2022-08-19

 For lambda = 0.03, MSE =  0.13404428909160396
Academic license - for non-commercial use only - expires 2022-08-19

 For lambda = 0.08, MSE =  0.13445657553870197
Academic license - for non-commercial use only - expires 2022-08-19

 For lambda = 0.1, MSE =  0.1345157845773268
Academic license - for non-commercial use only - expires 2022-08-19

 For lambda = 0.3, MSE =  0.13485956481506053
Academic license - for non-commercial use only - expires 2022-08-19

 For lambda = 0.8, MSE =  0.13684273955762288
Academic license - for non-commercial use only - expires 2022-08-19

 For lambda = 1.0, MSE =  0.13714135531087593
Academic license - for non-commercial use only - expires 2022-08-19

 For lambda = 3.0, MSE =  0.1405566814167734
Academic license - for non-commercial use only - expires 2022-08-19
Academic license - for non-commercial use only - expires 2022-08-19



 Best Beta vector = 
1-dimensional DenseAxisArray{Float64,1,...} with index sets:
    Dimension 1, 0:7
And data, a 8-element Vector{Float64}:
   2.4636518957528164
  -0.45787550200138105
```

```
  10.054386044591087
  18.88974340764618
   9.054858218924496
 -18.06448759078263
 -10.02468984890087
   5.778924433654609


Best Lambda = 0.01
Best MSE = 0.13400726366428356
New MSE on test data = 4.889140607362017
```

The lower bound MSE for part b and part d are nearly identical.

In [ ]: