Homework 6

1. Fitting non-quadratic losses to data. In non_quadratic.json, you will find a 500×300 matrix U_train and a 500-vector v_train consisting of raw training input and output data, and a 500×300 matrix U_test and a 500-vector v_test consisting of raw test input and output data, respectively. We will work with input and output embeddings \( x = \phi(u) = (1, u) \) and \( y = \psi(v) = v \). Our performance metric is the RMS error on the test data set.

In regression_fit.jl we have also provided you with a function

\[
\text{regression_fit}(X, Y, l, r, \lambda).
\]

This function takes in input/output data \( X \) and \( Y \), a loss function \( l(\hat{y}, y) \), a local regularizer function \( r(\theta) \), and a local regularization hyper-parameter \( \lambda \). It outputs the model parameters \( \theta \) for the RERM linear predictor. You must include the Flux and LinearAlgebra Julia packages in your code in order to utilize this function. You will use this function to fit a linear predictor to the given data using the loss functions listed below.

- Quadratic loss: \( \ell(\hat{y}, y) = (\hat{y} - y)^2 \).
- Absolute loss: \( \ell(\hat{y}, y) = |\hat{y} - y| \).
- Huber loss, with \( \alpha \in \{0.5, 1, 2\} \): \( \ell(\hat{y}, y) = p_{\alpha}^{\text{hub}}(\hat{y} - y) \), where
  \[
  p_{\alpha}^{\text{hub}}(r) = \begin{cases} 
  r^2 & |r| \leq \alpha \\
  \alpha(2|r| - \alpha) & |r| > \alpha.
  \end{cases}
  \]
- Log Huber loss, with \( \alpha \in \{0.5, 1, 2\} \): \( \ell(\hat{y}, y) = p_{\alpha}^{\text{dh}}(\hat{y} - y) \), where
  \[
  p_{\alpha}^{\text{dh}}(r) = \begin{cases} 
  r^2 & |r| \leq \alpha \\
  \alpha^2(1 - 2 \log(\alpha) + \log(r^2)) & |r| > \alpha.
  \end{cases}
  \]

We won’t use regularization so you can use \( r(\theta) = 0 \) and \( \lambda = 1 \) (though your choice of \( \lambda \) does not matter).

Report the training and test RMS errors. Which model performs best? Create a one-sentence conjecture or story about why the particular model was the best one.

Julia hint. You will need to define the loss functions above in Julia. You can do this in a compact (but readable) form by defining the function inline.

For example, for quadratic loss, \( 1_{\text{quadratic}}(\hat{y}, y) = (\hat{y} - y)^2 \) and similarly, for huber loss, \( 1_{\text{huber}}(\hat{y}, y) = \text{huber}(\hat{y} - y, \alpha) \), where, \( \text{huber}(r, \alpha) \) function would be defined separately.
You will also need to do the same for the regularizer (although it is zero). You can do this with $r(\theta) = 0$. Pass the names of the function (say, $l_{\text{quadratic}}$ instead of $l$) in the `regression_fit(X, Y, l, r, lambda)` function above.

**Solution.**

<table>
<thead>
<tr>
<th></th>
<th>Train RMS error</th>
<th>Test RMS error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Square</td>
<td>0.734</td>
<td>1.39</td>
</tr>
<tr>
<td>Absolute</td>
<td>0.949</td>
<td>1.20</td>
</tr>
<tr>
<td>0.5-Huber</td>
<td>0.858</td>
<td>1.26</td>
</tr>
<tr>
<td>1-Huber</td>
<td>0.787</td>
<td>1.31</td>
</tr>
<tr>
<td>2-Huber</td>
<td>0.739</td>
<td>1.39</td>
</tr>
<tr>
<td>0.5-Log Huber</td>
<td>1.09</td>
<td>1.15</td>
</tr>
<tr>
<td>1-Log Huber</td>
<td>0.942</td>
<td>1.22</td>
</tr>
<tr>
<td>2-Log Huber</td>
<td>0.766</td>
<td>1.36</td>
</tr>
</tbody>
</table>

(The actual values can vary based on the number of iterations) The log Huber predictor with $\alpha = 0.5$ performs the best on the test data. A reasonable guess as to why this did well is that there are many outliers in the data (although this is not the only conjecture that was reasonable).

```julia
using LinearAlgebra
using Random
using Flux

include("readclassjson.jl")
include("regression_fit.jl")

Data = readclassjson("non_quadratic.json")
X_train = Data["U_train"]
X_test = Data["U_test"]
y_train = Data["v_train"]
y_test = Data["v_test"]

function huber(r, alpha)
    if abs(r) <= alpha
        return r.^2
    else
        return alpha .* (2 .* abs(r) - alpha)
    end
end

function log_huber(r, alpha)
    if abs(r) <= alpha
        return r.^2
    end
end
```
else
    return (alpha.^2)*(1 - 2*log(alpha) + log(r.^2))
end
end

function RMSE(yhat, y)
    mse = sum((yhat .- y).^2)/length(y)
    return sqrt(mse)
end

# Quadratic Loss
loss_square(yhat, y) = norm(yhat - y,2).^2

# Absolute Loss
loss_abs(yhat, y) = norm(yhat - y,1)

# Huber loss
loss_huber_half(yhat, y) = huber(yhat - y, 0.5)
loss_huber(yhat, y) = huber(yhat - y, 1)
loss_huber2(yhat, y) = huber(yhat - y, 2)

# Log Huber loss
loss_log_huber_half(yhat, y) = log_huber(yhat - y, 0.5)
loss_log_huber(yhat, y) = log_huber(yhat - y, 1)
loss_log_huber2(yhat, y) = log_huber(yhat - y, 2)

# Regression
reg(x) = 0

losses = [loss_square, loss_abs, loss_huber, loss_huber_half, loss_huber2, loss_log_huber, loss_log_huber_half, loss_log_huber2]
names = ["square", "absolute", "huber", "huber_half", "huber2", "log_huber", "log_huber_half", "log_huber2"]

train_rmses = Dict()
test_rmses = Dict()

X_train = [ones(500) X_train]
X_test = [ones(500) X_test]

for (LOSS, NAME) in zip(losses, names)
    theta = regression_fit(X_train, y_train, LOSS, reg, 1, numiters = 100)
    train_rmses[NAME] = RMSE(X_train*theta, y_train)
test_rmses[NAME] = RMSE(X_test*theta, y_test)
end

display(train_rmses)
display(test_rmses)

2. Non quadratic regularizers.

Using the same data file non_quadratic.json and provided regression_fit function, we will investigate the impact of different regularizers.

Using the test RMS error, select the best 2 loss functions from Problem 1. We will evaluate them with the following regularizers:

- Quadratic or ridge regularization: \( r(\theta) = \lambda \|\theta_{2:k}\|^2 \)
- Lasso regularization: \( r(\theta) = \lambda \|\theta_{2:k}\|_1 \)
- No regularization (you’ll use this as a baseline)

You are free to choose the weight \( \lambda \). A good starting choice is 0.1 but you are encouraged to experiment. Keep in mind different weights may work better for different functions.

(a) For the 2 best loss functions in Problem 1, and the regularizers listed above, report the training and test RMS errors. What is the best loss + regularizer combination? Don’t forget to try a few different values of \( \lambda \) (you only have to report the one you choose).

(b) Provide a (brief) comment on whether your results in this problem agree with your results from Problem 1.

(c) Some loss functions, such as quadratic loss, are convex. Others are not (such as log-Huber). Convex functions are advantageous because they can be reliably optimized. Is your best loss + regularizer convex? If not, what is the best convex loss + regularizer you found?

*Julia hint.* Refer question 1.

*Solution.*

The two best loss functions from Problem 1 should be 0.5-log-Huber, with a test RMS error of 1.15, and absolute, with a test RMS error of 1.20.

In this problem, results may vary depending on the value of \( \lambda \). One sample of results with \( \lambda = 0.05 \) is shown. It is OK to have different results as long as \( \lambda \) is reported.

<table>
<thead>
<tr>
<th></th>
<th>Train RMS error</th>
<th>Test RMS error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5-Log Huber, no regularizer</td>
<td>1.09</td>
<td>1.15</td>
</tr>
<tr>
<td>0.5-Log Huber, quadratic, ( \lambda = 0.05 )</td>
<td>1.16</td>
<td>1.15</td>
</tr>
<tr>
<td>0.5-Log Huber, lasso, ( \lambda = 0.05 )</td>
<td>1.21</td>
<td>1.16</td>
</tr>
<tr>
<td>Absolute, no regularizer</td>
<td>0.95</td>
<td>1.20</td>
</tr>
<tr>
<td>Absolute, quadratic, ( \lambda = 0.05 )</td>
<td>1.16</td>
<td>1.14</td>
</tr>
<tr>
<td>Absolute, lasso, ( \lambda = 0.05 )</td>
<td>1.21</td>
<td>1.16</td>
</tr>
</tbody>
</table>
(a) The best loss + regularizer we found with $\lambda = 0.05$ was absolute loss $l(\hat{y}, y) = |\hat{y} - y|$ with a square regularizer $\|\theta_{2,k}\|^2_2$. Depending on the choice of $\lambda$, it is also possible to have 0.5-log-Huber loss perform better (for example, $\lambda = 0.1$ produces this result).

(b) A reasonable answer is that in Problem 1 the good performance of log-Huber suggests outliers in the data. Adding the regularizer reduces the impact of outliers. Since the regularizer improves our performance, the results here agree with Problem 1.

It’s possible to have other answers.

(c) The best convex loss + regularizer we found was absolute loss with a square regularizer. It is OK to have a different answer as long as you don’t confuse nonconvex and convex functions.

```julia
using LinearAlgebra
using Random
using Flux

include("readclassjson.jl")
include("regression_fit.jl")

Data = readclassjson("non_quadratic.json")
X_train = Data["U_train"]
X_test = Data["U_test"]
y_train = Data["v_train"]
y_test = Data["v_test"]

function RMSE(yhat, y)
    mse = sum((yhat .- y).^2)/length(y)
    return sqrt(mse)
end

loss_abs(yhat, y) = norm(yhat - y, 2)

function log_huber(r, alpha)
    if abs(r) <= alpha
        return r.^2
    else
        return (alpha.^2)*(1 - 2*log(alpha) + log(r.^2))
    end
end

loss_log_huber_half(yhat, y) = log_huber(yhat - y, 0.5)
```
function square_reg(theta)
    return norm(theta[2:end], 2).^2
end

function l1_reg(theta)
    return norm(theta[2:end], 1)
end

none_reg(x) = 0

########################################################################

regs = [square_reg, l1_reg, none_reg,]
regrnames = ["quadratic", "lasso", "none"]

losses = [loss_abs, loss_log_huber_half,]
lossnames = ["absolute", "log huber half"]

train_rmses = Dict()
test_rmses = Dict()

X_train = [ones(500) X_train]
X_test = [ones(500) X_test]

for (LOSS, LNAME) in zip(losses, lossnames)
    for (REG, RNAME) in zip(regs, regrnames)
        theta = regression_fit(X_train, y_train, LOSS, REG, 0.05)
        train_rmses[(LNAME, RNAME)] = RMSE(X_train*theta, y_train)
        test_rmses[(LNAME, RNAME)] = RMSE(X_test*theta, y_test)
    end
end

display(train_rmses)
display(test_rmses)

3. Fitting neural networks with Julia. In nn_regression.json, you will find a 4000 × 30 matrix $U_{train}$ and a 4000-vector $v_{train}$ consisting of raw training input and output data, and a 1000 × 30 matrix $U_{test}$ and a 1000-vector $v_{test}$ consisting of raw test input and output data, respectively. We will work with input and output embeddings $x = \phi(u) = u$ and $y = \psi(v) = v$.

In nn_regression.jl we have also provided you with a function

$$nn\_regression(X, Y, \lambda).$$
This function takes in input/output data X and Y, and a local regularization hyperparameter lambda. It outputs the model parameters theta for a neural network with parameters and activation functions defined in the code. You must include the Flux Julia package in your code in order to utilize this function.

(a) Use linear regression (without regularization) to fit a linear predictor to the training data. Report the training and test RMS errors.

(b) Inspect the nn_regression function in nn_regression.jl. What is the form of the neural network model in the code (i.e., the form of \( \hat{y} \))? Specify the number of layers; for all layers, specify the activation function and model parameters (including dimensions). In total, how many scalar model parameters are there? (For example, \( A \in \mathbb{R}^{13\times10} \) has \( 13 \times 10 = 130 \) scalar entries.)

(c) Using nn_regression.jl, fit a neural network to the training data for regularization parameters taking 10 values logarithmically spaced between \( 10^{-3} \) and \( 10^{0} \). Report the best test RMS error and the corresponding regularization parameter.

Remark. It is normal for training this neural network to take several minutes.

Hint. The code in nn_regression.jl randomly initializes the model parameters; in order to generate reproducible results on your end, we suggest you use the seed function from the Random package with a seed of your choice, e.g., Random.seed(0).

Hint: Use \( \text{lambdas} = 10.0 \cdot \text{range}(-3,0,\text{length}=10) \) to compute the \( \lambda \) values.

Hint: Call predictall(model,U) to compute a vector of \( \hat{y} \).

Solution.

(a) The test RMS error is 5.82.

(b) The form of the model is

\[
\hat{y} = g^4(g^3(g^2(g^1(x))))
\]

where \( g^1 : \mathbb{R}^{30} \to \mathbb{R}^{10}, g^2 : \mathbb{R}^{10} \to \mathbb{R}^{10}, g^3 : \mathbb{R}^{10} \to \mathbb{R}^{10}, \) and \( g^4 : \mathbb{R}^{10} \to \mathbb{R} \). \( g^1, g^2, \) and \( g^3 \) all have ReLu activation functions, while \( g^4 \) has an identity activation function. There are thus \( 31 \times 10 + 11 \times 10 + 11 \times 10 + 11 \times 1 = 541 \) scalar parameters. (including weights and bias)

(c) The best test RMS error was 1.05 for \( \lambda = 0.002 \).

println("Importing packages....")

using LinearAlgebra
# using Statistics
using Random
using Flux

include("readclassjson.jl")
include("writeclassjson.jl")
include("regression_fit.jl")
include("nn_regression.jl")

import PyCall
import PyPlot; const plt = PyPlot; plt.plt.style.use("seaborn")

Random.seed!(0)

# println("Making data....")

# n = 5000
# ntrain=4000
# d = 30

# X = randn(n,d)
# y = [ 0.2 * sum(X[i, 10:15].^2) + log(sum(abs.(X[i,1:30]))) + x[i,2]*x[i,1].^2 + 1.5*sin(x[i,4]) + 0.2*sqrt(abs(x[i,3])) + 0.3*x[i,8] + 0.42*x[i,7] + 2*rand() for i=1:n]

# global X_train = X[1:ntrain, :]
# X_test = X[ntrain+1:n, :]

# global y_train = y[1:ntrain]
# y_test = y[ntrain+1:n]

# # outlier_idxs_train = shuffle(1:500)[1:50]
# # y_train[outlier_idxs_train] = -y_train[outlier_idxs_train]
# # outlier_idxs_test = shuffle(501:1000)[1:100]
# # y_test[outlier_idxs_test] = -y_test[outlier_idxs_test]

Data = Dict()
Data["U_train"] = X_train
Data["v_train"] = y_train
Data["U_test"] = X_test
Data["v_test"] = y_test
writeclassjson(Data, "neural_net.json")

println("Loading data....")
Data = readclassjson("neural_net.json")
X_train = Data["U_train"]
X_test = Data["U_test"]
y_train = Data["v_train"]
y_test = Data["v_test"]

########################################################################
function RMSE(yhat, y)
    mse = sum((yhat .- y).^2)/length(y)
    return sqrt(mse)
end

nnz(a, tol) = sum(abs.(a) .<= tol)

function ridgeregression(X,Y,lambda)
    n,d = size(X)
    m = size(Y,2)
    A = [X; sqrt(lambda*n)*I(d)]
    B = [Y; zeros(d,m)]
    theta = A\B
end

println("linear regression...")
theta = X_train \ y_train
predictall(theta,X) = vcat([theta' * x for x in eachrow(X)]...)
println(RMSE(predictall(theta,X_test), y_test))

println("nn regression....")
lambdas = 10 .^ range(-3,0,length=10)

train_errors = []
test_errors = []
for in lambdas
    model = nnregression(X_train, y_train, )
    predictall(model,X) = vcat([model(x) for x in eachrow(X)]...)
    train_error = RMSE(predictall(model,X_train), y_train)
    test_error = RMSE(predictall(model,X_test), y_test)
    push!(train_errors, train_error)
    push!(test_errors, test_error)
    println(" ", test_error)
end
println(test_errors)

# plt.figure()
# plt.plot(lambdas, train_errors, color="blue", label="train")
# plt.legend()
# plt.tight_layout()
# plt.savefig("train.pdf", bbox_inches="tight")

# plt.figure()
# plt.plot(lambdas, test_errors, color="red", label="test")
# plt.legend()
# plt.tight_layout()
# plt.savefig("test.pdf", bbox_inches="tight")