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 \( \alpha \in \{0.5, 1, 2\} \): \( \ell(\hat{y}, y) = p^\text{hub}_\alpha(\hat{y} - y) \), where

\[
p^\text{hub}_\alpha(r) = \begin{cases} r^2 & |r| \leq \alpha \\ \alpha(|r| - \alpha) & |r| > \alpha. \end{cases}
\]

- Log Huber loss, with \( \alpha \in \{0.5, 1, 2\} \): \( \ell(\hat{y}, y) = p^\text{dh}_\alpha(\hat{y} - y) \), where

\[
p^\text{dh}_\alpha(r) = \begin{cases} r^2 & |r| \leq \alpha \\ \alpha^2(1 - 2\log(\alpha) + \log(r^2)) & |r| > \alpha. \end{cases}
\]

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, \( l\text{\_quadratic}(yhat, y) = (yhat - y)^2 \) and similarly, for huber loss, \( l\text{\_huber}(yhat, y) = \text{huber}(yhat - y, \alpha) \), 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 \( \text{regression\_fit}(X, Y, l, r, \lambda) \) function above.
2. **Non quadratic regularizers.**

Using the same data file \texttt{non\_quadratic.json} and provided \texttt{regresion\_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: \( r(\theta) = \lambda \|\theta_{2:k}\|_2^2 \)
- 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.*

3. **Fitting neural networks with Julia.** In \texttt{nn\_regression.json}, you will find a \( 4000 \times 30 \) matrix \( U_{\text{train}} \) and a 4000-vector \( v_{\text{train}} \) consisting of raw training input and output data, and a \( 1000 \times 30 \) matrix \( U_{\text{test}} \) and a 1000-vector \( v_{\text{test}} \) 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 \texttt{nn\_regression.jl} we have also provided you with a function

\[
\text{nn\_regression}(X, Y, \lambda).
\]

This function takes in input/output data \( X \) and \( Y \), and a local regularization hyper-parameter \( \lambda \). It outputs the model parameters \( \theta \) for a neural network with parameters and activation functions defined in the code. You must include the \texttt{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, \( A \in \mathbb{R}^{13 \times 10} \) 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 `lambdas = 10 .^ range(-3,0,length=10)` to compute the \( \lambda \) values.

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