Homework 5

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, `l_quadratic(yhat, y) = (yhat - y).^2` and similarly,

for huber loss, `l_huber(yhat, y) = huber(yhat - y, alpha)`.

where, `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_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 the `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: \( r(\theta) = \lambda \|\theta_{2:k}\|_2^2 \)
- 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>Loss Function</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 $\|\theta_{\lambda}\|_2^2$. 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.

```plaintext
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,]
regnames = [ "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, regnames)
        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. How often does your predictor over-estimate? In this problem, you will identify how often linear predictors with tilted absolute losses over-estimate.

In residual_props.json, you will find a 500 × 10 matrix U_train and a 500-vector v_train consisting of raw training input and output data, and a 500×10 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 $y = \psi(v) = v$, and use no regularization ($r(\theta) = 0$). You will also use regression_fit.jl from the previous problem.
Recall that the tilted absolute penalty is

$$p_\tau(u) = \begin{cases} -\tau u & u < 0 \\ (1 - \tau)u & u \geq 0, \end{cases}$$

where $\tau \in [0, 1]$. Fit a linear predictor to the given data using the tilted absolute penalty, i.e., $\ell(\hat{y}, y) = p_\tau(\hat{y} - y)$, for $\tau \in \{0.15, 0.5, 0.85\}$. For both the training set and the test set, report how frequently this predictor over-estimates ($\%$ of $\hat{y} > y$), and plot the empirical CDFs (Cumulative Distribution Function) of the residuals.

**Hint.** A predictor $\hat{y}$ over-estimates $y$ when $\hat{y} > y$. To generate an empirical CDF plot of the residuals $r$ of length $d$, you may plot `collect(1:d)/d` versus `sort(r)`.

**Solution.**

For $\tau = 0.15$, the predictor overestimates 14.6% of the time on the training set and 14.8% on the test set. For $\tau = 0.5$, the predictor overestimates 50.2% of the time on the training set and 52.8% on the test set. For $\tau = 0.85$, the predictor overestimates 86% of the time on the training set and 81.8% on the test set.

![Figure 1 CDFs for Problem 3.](image)

```plaintext
using LinearAlgebra
using Random
using Flux

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

Data = readclassjson("residual_props.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

function p_tlt(r, tau)
    if r < 0
        return -tau * r
    else
        return (1-tau) * r
    end
end

loss_tlt_15(yhat, y) = p_tlt(yhat - y, 0.15)
loss_tlt_50(yhat, y) = p_tlt(yhat - y, 0.5)
loss_tlt_85(yhat, y) = p_tlt(yhat - y, 0.85)

none_reg(x) = 0

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

losses = [loss_tlt_15, loss_tlt_50, loss_tlt_85]
names = ["tlt_15", "tlt_50", "tlt_85"]

train_rmses = Dict()
test_rmses = Dict()
train_res = Dict()
test_res = 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, none_reg, 1)

    train_rmses[NAME] = RMSE(X_train*theta, y_train)
    test_rmses[NAME] = RMSE(X_test*theta, y_test)
    train_res[NAME] = X_train*theta - y_train
test_res[NAME] = X_test*theta - y_test
end
display(train_rmses)
display(test_rmses)
t_15_train = mean(train_res["tlt_15"] .> 0)
t_15_test = mean(test_res["tlt_15"] .> 0)
t_50_train = mean(train_res["tlt_50"] .> 0)
t_50_test = mean(test_res["tlt_50"] .> 0)
t_85_train = mean(train_res["tlt_85"] .> 0)
t_85_test = mean(test_res["tlt_85"] .> 0)
println("% Overestimates for = 0.15:")
println("\t On training data = ", t_15_train)
println("\t On testing data = ", t_15_test)
println("% Overestimates for = 0.50:")
println("\t On training data = ", t_50_train)
println("\t On testing data = ", t_50_test)
println("% Overestimates for = 0.85:")
println("\t On training data = ", t_85_train)
println("\t On testing data = ", t_85_test)
d = length(train_res["tlt_15"]) y = collect(1:d)/d
fig, ax = plt.subplots(2,1)
taus = [0.15, 0.50, 0.85]
ax[1].set_title("Train")
for (i,resname) in enumerate(["tlt_15", "tlt_50", "tlt_85"]) res = train_res[resname]
ax[1].plot(sort(res), y, label=string(" = ", taus[i]))
end
ax[1].legend()
ax[2].set_title("Test")
for (i,resname) in enumerate(["tlt_15", "tlt_50", "tlt_85"]) res = test_res[resname]
ax[2].plot(sort(res), y, label=string(" = ", taus[i]))
end
ax[2].legend()
plt.savefig("residual_cdfs.pdf", bbox_inches="tight")