Homework 5

1. Polynomial fit. You are given a dataset polyfit.json with raw data $u_i \in \mathbb{R}$ represented in a $n$-vector $u$ and raw output $v_i \in \mathbb{R}$ represented in a $n$-vector $v$.

(a) Split the given dataset using 80-20 train-test split. Apply embeddings $\phi(u) = u$ and $\psi(v) = v$ to the raw input and raw output, respectively. Fit a least squares model to the train part of the dataset. Perform out of sample validation for the introduced data split and report root-mean-square error (RMSE) for the training set and test set.

(b) Now, we will introduce polynomial embedding which maps the raw input to the polynomial features of the degree $d - 1$ as shown below.

$$\phi(u) = (1, u, u^2, ..., u^{d-1})$$

Write a Julia function $X = \text{polynomial_features}(u, d)$ which performs a polynomial embedding of degree $d - 1$ described above for $n$-vector $u$ with raw input data and returns a $n \times d$ matrix $X$, where $x_i^T$ row represents the polynomial mapping for a scalar raw data input $u_i$. Embedding for raw output remains the same as in part (a).

(c) Find the optimal parameters $\theta^*$ that minimizes the square loss for the polynomial fits up to degree 20 for the train dataset. Plot the train and test RMSE versus the degree. Select the degree which enables the best fit for the given dataset. Compare and plot the fittings for one example of underfitting, one example of overfitting, and the best fit. Comment the results.

(d) Implement a Julia function $\text{avg_RMSE, std} = \text{cross_validation}(X, y, k)$ for the $n \times d$ matrix $X$ of features, the $n$-vector $y$ of corresponding labels, and the $k$ number of folds. The output should contain the average of root-mean-square errors for the test set and its standard deviation for the $k$ fits.

Plot the average RMSE versus the degree for the test set with standard deviation error bars for $k = 5$. Select the best degree choice for the given dataset.

Hint. Explore the Plots.jl documentation to plot the error bars.

Solution.

(a) Loss: 11.217549062604363

Example code is shown below:

```julia
include("readclassjson.jl")
include("RMSE.jl")
include("find_theta.jl")
using LinearAlgebra
using Random
```
data = readclassjson("polyfit.json")
U_DATA = data["u"]
V_DATA = data["v"]

n = length(V_DATA)
n_train = round(Int, 0.8*n)
X_train, X_test = U_DATA[1:n_train,:), U_DATA[(n_train+1):end,:]
y_train, y_test = V_DATA[1:n_train,:), V_DATA[(n_train+1):end,:]
CASE = X_train
CASE_TEST = X_test
theta = 0
theta, loss = find_theta(CASE,y_train)
y_pred = [ones(n-n_train) X_test]*theta
test_loss = RMSE(y_pred, y_test)
println("Part a answer: $test_loss")

(b) Example code is shown below.

#part b

function polynomial_features(u,d)
    embedding = [ones(length(u)) u]
    for degree = 2:1:d-1
        embedding = [embedding u.^degree]
    end
    return embedding
end

#part c
train_loss_data = []
test_loss_data = []

for degree=2:1:20
    x_train = polynomial_features(X_train, degree)
x_test = polynomial_features(X_test, degree)
theta = x_train \ y_train
y_pred = x_train*theta
train_loss = sqrt(sum((y_pred - y_train).^2)/n_train)
test_loss = sqrt(sum((x_test*theta - y_test).^2)/n_test)
println("degree $degree train_loss: ", train_loss, " test loss: ", test_loss)
push!(train_loss_data, train_loss)
push!(test_loss_data, test_loss)
end

d = 2:1:20
plot(d, [train_loss_data test_loss_data], label = ["train RMSE" "test RSME"]
# savefig("RMSE_degree_plot.png")

# underfitted d = 2
degree2 = CASE[:,1:3]
theta2 = degree2 \ y_train

# best fit d = 6
degree6 = CASE[:,1:7]
theta6 = degree6 \ y_train

# overfitted d = 11
degree9 = CASE[:,1:10]
theta9 = degree9 \ y_train

plot(u_data_og, v_data_og, seriestype=:scatter, label = "training samples")
x = range(0, 1, length=100)

x2 = [ones(length(x)) x x.^2]
underfit = x2 * theta2
plot!(x, underfit, label = "under fit")

x6 = [x2 x.^3 x.^4 x.^5 x.^6]
bestfit = x6 * theta6
plot!(x, bestfit, label = "best fit")

x9 = [x6 x.^7 x.^8 x.^9]
overfit = x9 * theta9
plot!(x[5:end], overfit[5:end], label = "over fit")

# savefig("models.png")

# part d

function cross_validation(U_DATA, V_DATA,k)

    rmse_array_fold = []

    for i = 1:k

        # split data
        n_test_start = 15*(i-1)
        n_test_stop = 15*i
        X_test, y_test = U_DATA[n_test_start+1:n_test_stop,:], V_DATA[n_test_start+1:n_test_stop,:]
        X_train_1, y_train_1 = U_DATA[1:n_test_start,:], V_DATA[1:n_test_start,:]
        X_train_2, y_train_2 = U_DATA[n_test_stop+1:end,:], V_DATA[n_test_stop+1:end,:]

        # training d = 1
        degree1 = CASE[:,1:2]
        theta1 = degree1 \ y_train
        train_1 = X_train_1 * theta1
        train_2 = X_train_2 * theta1
        train_3 = X_test * theta1
        train_4 = train_1 + train_2 + train_3

        # training d = 3
        degree3 = CASE[:,1:5]
        theta3 = degree3 \ y_train
        train_5 = X_train_1 * theta3
        train_6 = X_train_2 * theta3
        train_7 = X_test * theta3
        train_8 = train_5 + train_6 + train_7

        # training d = 5
        degree5 = CASE[:,1:9]
        theta5 = degree5 \ y_train
        train_9 = X_train_1 * theta5
        train_10 = X_train_2 * theta5
        train_11 = X_test * theta5
        train_12 = train_9 + train_10 + train_11

        # training d = 7
        degree7 = CASE[:,1:15]
        theta7 = degree7 \ y_train
        train_13 = X_train_1 * theta7
        train_14 = X_train_2 * theta7
        train_15 = X_test * theta7
        train_16 = train_13 + train_14 + train_15

        # evaluate
        train_17 = train_4
        train_18 = train_8
        train_19 = train_12
        train_20 = train_16

        # savefig("cross_validation.png")

    end
end
if i == k
    X_train = X_train_1
    y_train = y_train_1
elseif i == 1
    X_train = X_train_2
    y_train = y_train_2
else
    X_train = vcat(X_train_1,X_train_2)
    y_train = vcat(y_train_1,y_train_2)
end

# fit model
theta = X_train \ y_train
y_pred = X_test*theta
test_loss = sqrt(sum((y_pred - y_test).^2)/length(y_test))
push!(rmse_array_fold, test_loss)
#print(size(X_test))
end

avg_rmse = sum(rmse_array_fold)/k
std = Statistics.std(rmse_array_fold)

return avg_rmse, std
end

COMBINED_DATA = [u_data_og v_data_og]
SHUFFLED_DATA = COMBINED_DATA[shuffle(1:end), :]
u_data_shuffled = SHUFFLED_DATA[:,1]
v_data_shuffled = SHUFFLED_DATA[:,2]

#println(u_data_shuffled)
u_data = [ones(length(u_data_shuffled)) u_data_shuffled]
avg_rmse_array = []
std_array = []

for degree=2:1:20
    u_data = [u_data u_data_shuffled.^degree]
    avg_rmse,std = cross_validation(u_data, v_data_shuffled, 5)
push!(avg_rmse_array, avg_rmse)
push!(std_array, std)
end

degree = 2:1:20
plot(degree,avg_rmse_array,grid=false,yerror=std_array)
(c) Best fit: degree 5

Plots:
Values:

degree 2 train_loss: 8.426419082431694 test loss: 4.458221430396712
degree 3 train_loss: 8.278628706008677 test loss: 4.956378963371017
degree 4 train_loss: 7.002712612221152 test loss: 1.3258791885467602
degree 5 train_loss: 6.996075524579252 test loss: 1.2956473110093492
degree 6 train_loss: 6.991848919565315 test loss: 1.468389178119094
degree 7 train_loss: 6.737020880883263 test loss: 2.248913117024868
degree 8 train_loss: 6.595974965718213 test loss: 2.397288835393142
degree 9 train_loss: 6.416284159021109 test loss: 2.876295585609932
degree 10 train_loss: 5.839610199650911 test loss: 4.412985722407249
degree 11 train_loss: 5.636786115644632 test loss: 4.43155571425957
degree 12 train_loss: 5.394930990103881 test loss: 5.1814930639771815
degree 13 train_loss: 5.144580247638554 test loss: 5.629434894358495
degree 14 train_loss: 5.0557338102661 test loss: 5.718184368008277
degree 15 train_loss: 4.839288745882897 test loss: 6.956627436643218
degree 16 train_loss: 4.270826112080955 test loss: 8.337732327247368
degree 17 train_loss: 4.055671204992111 test loss: 9.61357172209051
degree 18 train_loss: 3.82273077543463 test loss: 11.05502751948818
degree 19 train_loss: 3.595274368325878 test loss: 12.52919955434321
degree 20 train_loss: 3.584288729578905 test loss: 12.486417727445115

(d) Plot:

![Graph showing example code](image)

Example code is shown below:

```python
```
2. **Sequential outlier removal.** We consider the problem of fitting data corrupted with outliers, using a simple sequential outlier removal method. In `outlier_rem.json`, you will find a $250 \times 10$ matrix $U_{\text{train}}$ and a 250-vector $v_{\text{train}}$ consisting of raw training input and output data, and a $250 \times 10$ matrix $U_{\text{test}}$ and a 250-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 $y = \psi(v) = v$, and you will use a simple linear predictor (without a constant feature) with square loss to fit the model. We will judge model performance using the RMS error on the test set.

A number of the output data entries in the training set have been corrupted (but in a non-obvious way). You do not know which data points have been corrupted, or how many, but you can assume no more than 50. You will explore a simple sequential method to remove the corrupted data points and form a prediction model.

Repeat the following for 50 iterations:

- Create a linear predictor from the training data set.
- Find the data point in the training data set with the largest prediction error.
- Remove the data point from the training data set.

This results in 50 predictors. Plot the test RMS error for them, versus the number of points removed. Give a guess as to how many of the data points were corrupted, with justification.

*Julia hint.* To remove row $i$ from a matrix $X$ and a vector $y$, use $X[\text{setdiff}(1:\text{end}, i), :]$ and $y[\text{setdiff}(1:\text{end}, i)]$, respectively.

**Solution.** A plot of test RMS error for each iteration is shown below. There were actually 25 outliers in this data set, but estimates between 15–30 were accepted, with proper justification. The justification is that the RMS test error flattens out between iterations 15–30; removing the outliers significantly reduces the test RMS error, helping the predictive power of the model, but removing data points that are not outliers hurts the model’s predictive power.
3. **Augmenting features with the average.** You are fitting a regression model $\hat{y} = x^T \theta$ to data, computing the model coefficients $\theta \in \mathbb{R}^d$ using least squares. A friend suggests adding a new feature, which is the average of the original features, \textit{i.e.}, he suggests using the new feature vector $\tilde{x} = (x, \text{avg}(x))$. He explains that by adding this new feature, you might end up with a better model. (Of course, you would test the new model using validation.) Is this a good idea? How probable is it that this idea results in a better model with smaller error?

**Solution.** This is a bad idea. With the new feature the feature matrix becomes

$$X_{\text{new}} = [X \ \text{avg}(x)] .$$

This means that the last column is $1/n$ times the sum of the other columns. In particular, the last column of $X_{\text{new}}$ is a linear combination of the other columns. It follows that the matrix $X_{\text{new}}$ does not have linearly independent columns, and we cannot solve the associated least squares problem. Even if we could solve the associated least squares problem, our results would not be any better. Any linear combination of the columns of $X_{\text{new}}$ is also a linear combination of the columns of $X$, and it follows that we cannot find a linear combination of the columns of $X_{\text{new}}$ that is closer to $y$ than the optimal linear combination of the columns of $X$. 

![Figure 1](image-url) 

**Figure 1** Plot of test RMS error versus iteration.