Homework 9

1. **Imputing missing entries in data.** In `impute_clusters.json`, you will find a 750 × 2 matrix `U_train` and a 750 × 2-vector `U_test` consisting of raw training and test data. (There is no output data.) We will work with \( x = \phi(u) = u \). In this exercise, you will fit data models and impute missing entries in data, determining how many clusters there are in the data by validating how well each data model does at imputing data.

Using the following implausibility functions, fit the data models using the training data. For each of the data models, give the model parameter \( \theta \). Submit your code used to generate these models.

To validate each data model, you will remove an element from each test data record at random, impute the entry (substitute the missing with values from the following models) according to the data model, and compute the average RMS imputation error for each model. In addition, for each of the data models, plot the training data, in blue, and the test data, in red, on a scatter plot. (There should be one scatter plot for each implausibility function.)

(a) Sum squares implausibility function: \( \ell_\theta(x) = \|x - \theta\|_2^2 \).
(b) Sum absolute implausibility function: \( \ell_\theta(x) = \|x - \theta\|_1 \).
(c) \( k \)-means implausibility function, with \( k = 5 \): \( \ell_\theta(x) = \min_{j=1,...,5} \|x - \theta_j\|_2^2 \).
(d) \( k \)-means implausibility function, with \( k = 10 \): \( \ell_\theta(x) = \min_{j=1,...,10} \|x - \theta_j\|_2^2 \).
(e) \( k \)-means implausibility function, with \( k = 15 \): \( \ell_\theta(x) = \min_{j=1,...,15} \|x - \theta_j\|_2^2 \).
(f) \( k \)-means implausibility function, with \( k = 20 \): \( \ell_\theta(x) = \min_{j=1,...,20} \|x - \theta_j\|_2^2 \).
(g) Based on parts (a)-(f), give a guess for how many clusters are in the data. Provide a short justification for your answer.

**Hint.** For parts (a) and (b), you can use Statistics functions to compute \( \theta \).

**Hint.** For the \( k \)-means implausibility function, you can import Clustering and use `Clustering.kmeans()` The `kmeans(X, K)` function takes a matrix \( X \) where each column is a data point and \( K \), the number of clusters.

**Solution.**

The guess should “officially” be that there are 15 clusters. However, due to randomness it’s possible that the 10 or 20 cluster solution might achieve slightly lower RMSE. They should be very close.

A sample code is provided below and the figures are provided at the end.
using LinearAlgebra
using Statistics
using Random
using Distributions
using Clustering
import PyPlot; const plt = PyPlot

include("readclassjson.jl")

D = readclassjson("impute_clusters.json")
X_train = D["U_train"]
X_test = D["U_test"]

n, d = size(X_train)

function RMSE(Xhat, X)
    return sqrt(sum((Xhat-X).^2)/size(Xhat,1))
end

X_test_data_rm = Array{Union{Missing, Float64}}(missing, size(X_test))

for i=1:size(X_test,1)
    j = rand(1:d)
    X_test_data_rm[i,j] = X_test[i,j]
end

# Part a
theta = Statistics.mean(X_train,dims=1) # average of the columns
X_test_hat = copy(X_test_data_rm)
for i=1:size(X_test_hat,1)
    for j=1:d
        if ismissing(X_test_hat[i,j])
            X_test_hat[i,j] = theta[j] # substitute missing ones with mean
        end
    end
end
println(theta)
println(RMSE(X_test_hat, X_test))

plt.figure()
plt.xlabel("x_1")
plt.ylabel("x_2")
plt.scatter(X_train[:,1], X_train[:,2], color="blue", label="training")
plt.scatter(X_test_hat[:,1], X_test_hat[:,2], color="red", label="test/imputed")
plt.title(string("Square, RMSE=",RMSE(X_test_hat, X_test)))
plt.savefig(string("square_plot.pdf"))

# Part b
theta = Statistics.median(X_train, dims=1)
X_test_hat = copy(X_test_data_rm)
for i=1:size(X_test_hat,1)
    for j=1:d
        if ismissing(X_test_hat[i,j])
            X_test_hat[i,j] = theta[j]
        end
    end
end
# println(theta)
println(RMSE(X_test_hat, X_test))
plt.figure()
plt.xlabel("x_1")
plt.ylabel("x_2")
plt.scatter(X_train[:,1], X_train[:,2], color="blue", label="training")
plt.scatter(X_test_hat[:,1], X_test_hat[:,2], color="red", label="test/imputed")
plt.title(string("Absolute, RMSE=" , RMSE(X_test_hat, X_test)))
plt.savefig(string("absolute_plot.pdf"))

# Parts c-f
for K in [5, 10, 15, 20]
    result = Clustering.kmeans(collect(X_train'), K)
    global theta = collect((result.centers)')
    global X_test_hat = copy(X_test_data_rm)
    for i=1:size(X_test_hat,1)
        idx = 0
        for j=1:d
            if ~ismissing(X_test_hat[i,j])
                idx = argmin(((theta[:,j] .- X_test_hat[i,j]).^2)
            end
        end
        for j=1:d
            if ismissing(X_test_hat[i,j])
                X_test_hat[i,j] = theta[idx, j]
            end
        end
    end
    # println(theta)
3
println(RMSE(X_test_hat, X_test))

plt.figure()
plt.xlabel("x_1")
plt.ylabel("x_2")
plt.scatter(X_train[:,1], X_train[:,2], color="blue", label="training")
plt.scatter(X_test_hat[:,1], X_test_hat[:,2], color="red", label="test/imputed")
plt.title(string("k=", K, ", RMSE=",RMSE(X_test_hat, X_test)))
plt.savefig(string("k_", K, "_plot.pdf"))
end

Square, RMSE=1.1041540156698275

Absolute, RMSE=1.1047891585711844
k=15, RMSE=0.9118597572685866

k=20, RMSE=0.8422433979908285