Homework 9

1. **Imputing missing entries in data.** In `impute_clusters.json`, you will find a $750 \times 2$ matrix $U_{\text{train}}$ and a $750 \times 2$-vector $U_{\text{test}}$ 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 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 $k = 5$: $\ell_\theta(x) = \min_{j=1,\ldots,5} \|x - \theta_j\|_2^2$.

(d) $k$-means implausibility function, with $k = 10$: $\ell_\theta(x) = \min_{j=1,\ldots,10} \|x - \theta_j\|_2^2$.

(e) $k$-means implausibility function, with $k = 15$: $\ell_\theta(x) = \min_{j=1,\ldots,15} \|x - \theta_j\|_2^2$.

(f) $k$-means implausibility function, with $k = 20$: $\ell_\theta(x) = \min_{j=1,\ldots,20} \|x - \theta_j\|_2^2$.

(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 cluster solution will achieve slightly lower RMSE. They should be very close.

---

[https://juliastats.org/Clustering.jl/stable/kmeans.html]
In [10]: using LinearAlgebra
using Statistics
using Random
using Distributions
using Clustering

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

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

Out[10]: writeclassjson (generic function with 1 method)

In [11]: function RMSE(Xhat, X)
    return sqrt(sum((Xhat-X).^2)/size(Xhat,1))
end

define function create_data(k, n, d)
    dists = []
    for ki = 1:k
        push!(dists, MvNormal(1*randn(d), I/100))
    end

    X = []
    for i=1:n
        ki = rand(1:k)
        push!(X, rand(dists[ki]))
    end

    return hcat(X...)', dists
end

Out[11]: create_data (generic function with 1 method)
In [25]:

k = 15
n = 1500
d = 2
# Random.seed!(5)
# X, dists = create_data(k,n,d)
# X = collect(X)
# plt.scatter(X[:,1], X[:,2])
# plt.show()
# X_train = X[1:n÷2,:]
# X_test = X[n÷2+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
# D = Dict()
# D["U_train"] = X_train
# D["U_test"] = X_test
# D["U_test_data_rm"] = X_test_data_rm
# writeclassjson(D, "impute_clusters.json")

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

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
In [26]: `X_test_data_rm`

Out[26]: 750×2 Array{Union{Missing, Float64},2}:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.46891</td>
<td>missing</td>
</tr>
<tr>
<td>missing</td>
<td>1.13645</td>
</tr>
<tr>
<td>-0.154458</td>
<td>missing</td>
</tr>
<tr>
<td>missing</td>
<td>-0.730167</td>
</tr>
<tr>
<td>missing</td>
<td>-1.43932</td>
</tr>
<tr>
<td>0.114878</td>
<td>missing</td>
</tr>
<tr>
<td>missing</td>
<td>0.539652</td>
</tr>
<tr>
<td>-0.120205</td>
<td>missing</td>
</tr>
<tr>
<td>missing</td>
<td>0.133664</td>
</tr>
<tr>
<td>-1.93339</td>
<td>missing</td>
</tr>
<tr>
<td>missing</td>
<td>0.207469</td>
</tr>
<tr>
<td>missing</td>
<td>-0.889097</td>
</tr>
<tr>
<td>-0.823596</td>
<td>missing</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>missing</td>
<td>0.516693</td>
</tr>
<tr>
<td>missing</td>
<td>3.2788</td>
</tr>
<tr>
<td>missing</td>
<td>1.25458</td>
</tr>
<tr>
<td>-1.69374</td>
<td>missing</td>
</tr>
<tr>
<td>missing</td>
<td>0.151755</td>
</tr>
<tr>
<td>-0.359042</td>
<td>missing</td>
</tr>
<tr>
<td>1.69274</td>
<td>missing</td>
</tr>
<tr>
<td>missing</td>
<td>-1.48937</td>
</tr>
<tr>
<td>-0.353427</td>
<td>missing</td>
</tr>
<tr>
<td>-0.989329</td>
<td>missing</td>
</tr>
<tr>
<td>missing</td>
<td>0.0536509</td>
</tr>
<tr>
<td>missing</td>
<td>-1.01005</td>
</tr>
</tbody>
</table>
In [27]:

```python
# Part a
theta = Statistics.mean(X_train, dims=1)
X_test_hat = copy(X_test_data_rm)
for i in range(size(X_test_hat, 1)):
    for j in range(dims):
        if ismissing(X_test_hat[i, j]):
            X_test_hat[i, j] = theta[j]
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"))
```

```
[-0.036203204521214435 0.1402996818339936]
1.112312813178296
```
### Part b

\[ \theta = \text{Statistics.median}(X_{\text{train}}, \text{dims}=1) \]

\[ X_{\text{test hat}} = \text{copy}(X_{\text{test data rm}}) \]

\[
\text{for } i=1: \text{size}(X_{\text{test hat}},1) \\
\quad \text{for } j=1:d \\
\quad\quad \text{if ismissing}(X_{\text{test hat}}[i,j]) \\
\quad\quad\quad X_{\text{test hat}}[i,j] = \theta[j] \\
\quad\end{\text{for}} \\
\end{\text{for}} \\
\]

\[ \text{println}(\theta) \]
\[ \text{println}(\text{RMSE}(X_{\text{test hat}}, X_{\text{test}})) \]

```python
plt.figure()
plt.xlabel("x_1")
plt.ylabel("x_2")
plt.scatter(X_{\text{train}}[:,1], X_{\text{train}}[:,2], color="blue", label="training")
plt.scatter(X_{\text{test hat}}[:,1], X_{\text{test hat}}[:,2], color="red", label="test/ imputed")
plt.title(string("Absolute, RMSE=",\text{RMSE}(X_{\text{test hat}}, X_{\text{test}})))
plt.savefig(string("absolute_plot.pdf"))
```

\[ [-0.07803491813728365, 0.19034339232244657] \]

1.1134446231432333
In [29]:

for K in [5, 10, 15, 20]
    result = Clustering.kmeans(collect(X_train'), K)
    theta = collect((result.centers)')

    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

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
In [ ]:

k=15, RMSE=0.9104768147629922

In [ ]:

k=20, RMSE=0.9060537980643261
Square, RMSE=1.112312813178296

Absolute, RMSE=1.1134446231432333

k=5, RMSE=1.022891014047501