THE CHINESE UNIVERSITY OF HONG KONG, SHENZHEN

DDA 2020

MACHINE LEARNING

Assignment4 Report

Name: Xiang Fei

Student ID: 120090414

Contents

DDA2020 Assignment 4

Q1.

proof: consider random variable X with n possible values $x_1, x_2, \cdots x_n$.

$$E[X] = \sum_{i=1}^{n} P(X=x_i) \cdot x_i$$

$$E[f(x)] = \sum_{i=1}^{n} P(X=x_i) f(x_i)$$

∴ we just need to show:

$$f\left(\sum_{i=1}^{n} P(X=x_i) \cdot x_i\right) \le \sum_{i=1}^{n} P(X=x_i) f(x_i)$$

use mathematical induction:

base case: $i=1$, $f(x_1) \le f(x_1)$

$i=2$, $f(P(X=x_1)x_1 + (1-P(X=x_1))x_2) \le P(X=x_1)f(x_1) + (1-P(X=x_1))f(x_2)$

the above statement is the definition of convex functions.

Inductive hypothesis:

For $n=k$: $f(P(x_1)x_1 + P(x_2)x_2 + \cdots + P(x_k)x_k) \le P(x_1)f(x_1) + \cdots + P(x_k)f(x_k)$

consider $n=k+1$: we have: $\sum_{i=1}^{k} \frac{P(x_i)}{1-P(x_{k+1})} = 1$

$$f\left(\sum_{i=1}^{k+1} P(x_i)x_i\right) = f\left((1-P(x_{k+1}))\sum_{i=1}^{k} \frac{P(x_i)}{1-P(x_{k+1})} x_i + P(x_{k+1})x_{k+1}\right)$$

$$\le (1-P(x_{k+1}))\, f\left(\sum_{i=1}^{k} \frac{P(x_i)}{1-P(x_{k+1})} x_i\right) + P(x_{k+1}) f(x_{k+1})$$

$$\le (1-P(x_{k+1}))\left(\sum_{i=1}^{k} \frac{P(x_i)}{1-P(x_{k+1})} f(x_i)\right) + P(x_{k+1}) f(x_{k+1}) = \sum_{i=1}^{k+1} P(x_i) f(x_i)$$

∴ Q.E.D.

Q2.

(1) Bernoulli distribution: $P(X^{(n)}|Z^{(n)}=K) = \prod_{j=1}^{d} \mu_{kj}^{x_j^{(n)}} (1-\mu_{kj})^{1-x_j^{(n)}}$

M-step:

$$\mu^{new} = \arg\max_{\mu} \sum_{n=1}^{N} E_{q_n(z^{(n)})}\left[\ln P(z^{(n)}, x^{(n)}; \mu)\right], \text{ s.t. } \sum_{k=1}^{K} \pi_k = 1$$

$$\mu^{new} = \arg\max_{\mu} \sum_{n=1}^{N}\sum_{k=1}^{K} r_k^{(n)}\left[\ln(P(z^{(n)}=k|\mu_{jk})) + \ln\left[P(x^{(n)}|z^{(n)}=k;\mu)\right)\right]$$

$$= \arg\max_{\mu} \sum_{n=1}^{N}\sum_{k=1}^{K} r_k^{(n)}\left[\ln(\pi_k) + \sum_{j=1}^{d} \ln \mu_{kj}^{x_j^{(n)}} (1-\mu_{kj})^{1-x_j^{(n)}}\right]$$

∴ we just need:

$$\max \sum_{n=1}^{N}\sum_{k=1}^{K} r_k^{(n)} \ln(\pi_k) + \sum_{n=1}^{N}\sum_{k=1}^{K} r_k^{(n)} \sum_{j=1}^{d} x_j^{(n)} \ln \mu_{kj} + \sum_{n=1}^{N}\sum_{k=1}^{K} r_k^{(n)} \sum_{j=1}^{d} (1-x_j^{(n)}) \ln(1-\mu_{kj})$$

$$\frac{\partial J}{\partial \mu_{kj}} = \frac{\sum_{n=1}^{N} r_k^{(n)} x_j^{(n)}}{\mu_{kj}} - \frac{\sum_{n=1}^{N} r_k^{(n)} (1-x_j^{(n)})}{1-\mu_{kj}} = 0$$

$$\Rightarrow \mu_{kj} = \frac{\sum_{n=1}^{N} r_k^{(n)} x_j^{(n)}}{\sum_{n=1}^{N} r_k^{(n)}}, \text{ the same as } \mu_{kj} = \frac{\sum_i r_{ik} x_{ij}}{\sum_i r_{ik}}$$

Q.E.D.

(2) Beta distribution prior: $P(\mu_{jk}) = \frac{\mu_{jk}^{\alpha-1}(1-\mu_{jk})^{\beta-1}}{B(\alpha,\beta)}$

For MAP estimation, we try to maximum the posterior.

$$\mu^{new} = \arg\max_{\mu} \sum_{n=1}^{N}\sum_{k=1}^{K} r_k^{(n)}\left[\ln(P(z^{(n)}=k|\mu)) + \ln\left[P(x^{(n)}|z^{(n)}=k;\mu)\right)\right] + \ln(P(\mu_{jk}))$$

∴ we just need:

$$\max \sum_{n=1}^{N}\sum_{k=1}^{K} r_k^{(n)} \ln(\pi_k) + \sum_{n=1}^{N}\sum_{k=1}^{K} r_k^{(n)} \sum_{j=1}^{d} x_j^{(n)} \ln \mu_{kj} + \sum_{n=1}^{N}\sum_{k=1}^{K} r_k^{(n)} \sum_{j=1}^{d} (1-x_j^{(n)}) \ln(1-\mu_{kj})$$
$$+ \sum_{n=1}^{N}\sum_{k=1}^{K} (\alpha-1)\ln \mu_{kj} + \sum_{n=1}^{N}\sum_{k=1}^{K} (\beta-1)\ln(1-\mu_{kj})$$

$$\therefore \frac{\partial J}{\partial \mu_{kj}} = \frac{\sum_{n=1}^{N} r_k^{(n)} x_j^{(n)} + \alpha - 1}{\mu_{kj}} - \frac{\sum_{n=1}^{N} r_k^{(n)} (1-x_j^{(n)}) + \beta - 1}{1-\mu_{kj}} = 0$$

$$\Rightarrow \mu_{kj} = \frac{\sum_{n=1}^{N} r_k^{(n)} x_j^{(n)} + \alpha - 1}{\sum_{n=1}^{N} r_k^{(n)} + \alpha + \beta - 2}, \text{ the same as } \mu_{kj} = \frac{\sum_i r_{ik} x_{ij} + \alpha - 1}{\sum_i r_{ik} + \alpha + \beta - 2}$$

∴ Q.E.D.

Q3.

proof:

$$J_W(\Sigma)=\frac{1}{2}\sum_{k=1}^{K}\sum_{i:\Sigma_i=k}\sum_{i':\Sigma_{i'}=k}(X_i-X_{i'})^2=\frac{1}{2}\sum_{k=1}^{K}\sum_{i:\Sigma_i=k}(n_k S^2+n_k(\bar{X}_k-X_i)^2)$$

$$=\frac{1}{2}\sum_{k=1}^{K}n_k^2\cdot\frac{1}{n_k}\sum_{i:\Sigma_i=k}(X_i-\bar{X}_k)^2+\frac{1}{2}\sum_{k=1}^{K}n_k\left(n_k\frac{1}{n_k}\sum_{i:\Sigma_i=k}(X_i-\bar{X}_k)^2+n_k(\bar{X}_k-\bar{X}_k)^2\right)$$

$$=\frac{1}{2}\sum_{k=1}^{K}n_k\sum_{i:\Sigma_i=k}(X_i-\bar{X}_k)^2+\frac{1}{2}\sum_{k=1}^{K}n_k\sum_{i:\Sigma_i=k}(X_i-\bar{X}_k)^2$$

$$=\sum_{k=1}^{K}n_k\sum_{i:\Sigma_i=k}(X_i-\bar{X}_k)^2$$

$\therefore$ Q.E.D.

Q4.

$$AUC=\frac{1}{m^+m^-}\sum_{i=1}^{m^+}\sum_{j=1}^{m^-}u(e_{ij})$$

we just need to prove:

$$\sum_{i=1}^{m^+}\sum_{j=1}^{m^-}u(e_{ij})=\sum_{i=1}^{m^+}rank_i-(m^+)(m^++1)/2$$

don't consider the same rank case, for ith positive sample, its rank is $rank_i$, so there are $rank_i-1$ samples have smaller predictor, including $i-1$ positive samples and $rank_i-i$ negative samples

when $1\le j\le rank_i-i$, $e_{ij}=1$

when $rank_i-i<j\le n$, $e_{ij}=0$

$$\therefore\sum_{i=1}^{m^+}\sum_{j=1}^{m^-}u(e_{ij})=\sum_{i=1}^{m^+}(rank_i-i)=\sum_{i=1}^{m^+}rank_i-\frac{(1+m^+)\cdot m^+}{2}$$

if some of the data have the same g value we take the average rank of them, since the sum of the rank don't change, so the form is still $\sum_{i=1}^{m^+}(rank_i-i)$

suppose negative samples has smaller predictors is N^-, has equal value is $N\bar{e}$. $\therefore rank_i-i=N^-+N\bar{e}/2$

it equals to: $\sum_{i}^{m^+}\sum_{j}^{m^-} I(g(x_j^-) < x_i^+)) + \frac{1}{2} I(g(x_j^-) = x_i^+))$

$\therefore$ Q.E.D.

Question 5

get the mean of X: here A is X in the question:

```
m = np.sum(A,axis=1)/10
```

then do the normalization for A(in fact, get x(n)-mu):

```
for i in range(5):
    for j in range(10):
        A[i][j] = A[i][j]-m[i]
```

```
print(A)
```

```
[[ 2.4  0.4  1.4 -0.6  1.4  2.4 -1.6 -1.6 -1.6 -2.6]
 [-0.8  1.2  1.2  0.2 -0.8  2.2  2.2 -2.8 -3.8  1.2]
 [ 0.8 -3.2  0.8  2.8  0.8 -1.2 -3.2  1.8  0.8 -0.2]
 [-3.2 -3.2  2.8  1.8 -1.2  0.8  2.8  2.8 -2.2 -1.2]
 [-0.7 -0.7 -0.7  0.3  2.3 -0.7  3.3 -0.7 -1.7 -0.7]]
```

Calculate the empirical covariance matrix:

```
sigma = np.dot(A,A.T)/10
print(sigma)
```

```
[[ 3.04  0.82 -0.02 -0.82 -0.12]
 [ 0.82  3.76 -2.16  1.04  1.04]
 [-0.02 -2.16  3.56  0.76 -0.84]
 [-0.82  1.04  0.76  5.56  1.16]
 [-0.12  1.04 -0.84  1.16  2.21]]
```

Then, compute eigenvectors and eigenvalues, and choose two of them.

```
eigen_vals, eigen_vecs = np.linalg.eig(sigma)
index=np.argsort(eigen_vals)
n_index=index[-1:-3:-1]
n_eigenVec=eigen_vecs[:, n_index]
n_eigenVals=eigen_vals[n_index]
```

```
print(n_eigenVec)
```

```
[[-0.02625442  0.29809153]
 [ 0.57873744  0.39493632]
 [-0.32862419 -0.58025264]
 [ 0.65356276 -0.64618454]
 [ 0.35949345  0.03031732]]
```

```
print(n_eigenVals)
```

```
[6.76978985 5.93067614]
```

Finally, compute the projection:

```
projection=np.dot(n_eigenVec.T,A)
```

```
print(projection)
```

```
[[-3.13194616 -0.60746566  1.97315969  0.4956134  -0.72008587  1.87576557
   5.38313097 -0.591651   -4.46907149 -0.20744945]
 [ 1.98183693  4.49653708 -1.40348923 -2.87861204  0.48232827  1.74241301
   0.53945236 -4.45776178 -1.07184006  0.56913546]]
```

2. Programming Question

2.1 Question restatement

In the programming problem, we need to implement 3 clustering algorithms from scratch, including K-means, accelerated K-means with triangle-inequality, GMM-EM. And then, we need to use an internal evaluation metrics: Silhouette coefficient and an external evaluation metrics: Rand Index to evaluate the performance of the above algorithms. What's more, we are also required to analyze the sensitivity to clustering initialization of each algorithm. Last, we need to record the corresponding iteration numbers and the required runtime for each algorithm.

2.2 K-means

First, I write a function to compute the Euclidean distance between two points. Then, I write a function to initialize centroids with random samples. Here I use random.uniform(0,len(data)) to generate k index and then can have k initial points as the original centroids. And then is a function to implement the main processing logic of K-means algorithm. We iteratively assignment and refitting until the assignment will not change the clustering. In assignment part, we find the closest centroid for each point, and in refitting part, we update the centroids in the given assignment.

The final centroids is like the following:

```
[[14.81910448 14.53716418  0.88052239  5.59101493  3.29935821  2.70658507
   5.21753731]
 [11.98865854 13.28439024  0.85273659  5.22742683  2.88008537  4.58392683
   5.0742439 ]
 [18.72180328 16.29737705  0.88508689  6.20893443  3.72267213  3.60359016
   6.06609836]]
```

The final clustering is like the following:

```
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 0. 1. 1. 1. 0.
 1. 1. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 2. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 0. 2. 2.
 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.
 2. 2. 2. 2. 1. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.
 2. 2. 1. 2. 1. 2. 2. 2. 2. 2. 2. 2. 1. 1. 1. 1. 2. 1. 1. 1. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

2.3 Accelerated K-means with triangle-inequality

We can accelerate the K-means algorithms by reducing the computational distances according to Elkan and based on the triangle-inequality. Let x be a data point, and b and c be two centers. d(x, b) denotes the distance between x and b. We have two lemma:

- Lemma 1: if d(b, c) ≥ 2d(x, b), then d(x, c) > d(x, b)

- Lemma 2: d(x, c) > max{0, d(x, b) − d(d, c)} Based on these two lemmas, we can implement the accelerated K-means algorithm.

I first write a function to get the distances between different centers (d(c,c')). Then, I use random.sample to get the initial centroids. Then is the first step of accelerated K-means, I named it as initialize_acc_k_means. In this part, we first pick the initial centers. Then set the lower bound l(x,c)=0 for each point x and center c. Assign each x to its closest initial center c(x) = $argmin_c$(d(x)), using Lemma1 here to avoid redundant distance calculations. Each time d(x,c) is computed, set l(x,c)=d(x,c). Assign upper bound u(x)=min_cd(x,c).

Then, I write a function called assignment to update the clustering. For all centers c, compute sc = 0.5*min_(c!=c')(d(c,c')). Then, identify all points x such that u(x)<=s(c(x)). If u(x)<=s(c(x)), no need to re-assignment. For all remaining points x and center c, if no need to re-assignment, compute d(x,c(x)), and change the need flag to be true. Otherwise, d(x,c(x))=u(x). Then, check whether d(x,c(x))>l(x,c) or d(x,c(x))>0.5*d(c(x),c), if it is true, compute d(x,c). If d(x,c)<d(x,c(x)), then change d(x,c(x)) and the clustering for x.

And then, I write a function called update the centroid. m(c) is the mean of the points assigned to center c. For each point x and center c, assign l(x,c)=max{l(x,c)-d(c,m(c)),0}. And for each point x and center c, assign l(x,c)=max{l(x,c)-d(c,m(c)),0}. For each point x, assign u(x) = u(x)-d(m(c(x)),c(x)).

repeat the above two process until convergence(there is no different between current centroids and the previous centroids).

The final centroids is like the following:

```
[[11.96441558 13.27480519  0.8522      5.22928571  2.87292208  4.75974026
   5.08851948]
 [14.64847222 14.46041667  0.87916667  5.56377778  3.27790278  2.64893333
   5.19231944]
 [18.72180328 16.29737705  0.88508689  6.20893443  3.72267213  3.60359016
   6.06609836]]
```

The final clustering is like the following:

```
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 0. 1. 1. 1. 1.
 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 2. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 1. 1. 1. 1. 1. 0. 2. 2.
 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.
 2. 2. 2. 2. 1. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.
 2. 2. 1. 2. 1. 2. 2. 2. 2. 2. 2. 2. 1. 1. 1. 1. 2. 1. 1. 1. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
```

2.4 GMM-EM

For GMM-EM algorithm, we use two part to implement it: Expectation and Maximization.

For Expectation part, we use bayes theorem to get the gamma. Here p(z=k)=pi_k, and p(x|z=k) follows multivariate normal distribution.

For Maximization part, we compute mu_k, cov_k and pi_k using the given optimized formula.

We repeat the two steps until convergence(here we compute log likelihood to check the convergence).

The final clustering is like the following:

```
[2 2 2 2 2 2 2 2 0 2 2 2 2 2 2 2 1 2 2 1 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 2 2 2 2 2 1 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

2.5 Evaluation

2.5.1 Silhouette Coefficient

Given a clustering, we define

- a: The mean distance between a point and all other points in the same cluster.
- b: The mean distance between a point and all other points in the next nearest cluster

Silhouette coefficient s for a single sample is formulated as: s = (b-a)/max(a,b)

For all samples, compute the average Silhouette coefficient s.

The result for the above three clustering algorithms:

```
Silhouette Coefficient for kmeans: 0.46813908008596955
Silhouette Coefficient for acc_kmeans: 0.4719337319126895
Silhouette Coefficient for gmm-em: 0.41971966340999006
```

2.5.2 Rand index

Given a set of n samples S = {o1, o2, . . . , on}, there are two clusterings/partitions of S to compare, including:

- X = {X1, X2, . . . , Xr} with r clusters
- Y = {Y1, Y2, . . . , Ys} with s clusters

We can calculate the following values:

- a: The number of pairs of elements in S that are in the same subset in X and in the same subset in Y
- b: The number of pairs of elements in S that are in the different subset in X and in the different subset in Y
- c: The number of pairs of elements in S that are in the same subset in X and in the different subset in Y
- d: The number of pairs of elements in S that are in the different subset in X and in the same subset in Y

The rand index (RI) can be computed as follows:

RI=(a+b)/(a+b+c+d)

Note that RI ∈ [0, 1], and higher score corresponds higher similarity

The result for the above three clustering algorithms:

```
rand index for kmeans: 0.8713602187286398
rand index for acc_kmeans: 0.8743677375256322
rand index for gmm-em: 0.9242196400091137
```

2.6 Sensitivity

In this part, I run one clustering algorithm with random initialization multiple times, and calculate the variance of evaluation scores of these clustering results.

Results:

```
sc_var for kmeans: 3.5638471652921954e-06
ri_var for kmeans: 2.238679405280084e-06

sc_var for acc_kmeans: 3.6926331177279848e-06
ri_var for acc_kmeans: 4.040757042879567e-06

sc_var for gmm-em: 0.0012744445565184633
ri_var for gmm-em: 0.009305063773790617
```

2.7 Runtime Analysis

In this part, I run one clustering algorithm with random initialization multiple times, and calculate the average number of iterations and average time.

Results:

```
avg_iteration for kmeans: 9.55
avg_time for kmeans: 0.030767643451690675

avg_iteration for acc_kmeans: 10.25
avg_time for acc_kmeans: 0.09971770048141479

avg_iteration for gmm-em: 38.5
avg_time for gmm-em: 7.151297473907471
```