CS189

1. (a) By myself. consulting piazza.

(b) I certify that all solutions are entirely in my words and that I have not looked at another student's solutions. I have credited all external sources in this write up.

Yao Cai

HW10 Yao Cai 25933599

2. (a) when $k = n$, each centroid for each single-sample cluster is the sample itself, and loss for each point is 0, the objective can reach 0.

(b) $\frac{\partial}{\partial u_i}\left(\lambda \|u_i\|_2^2 + \sum_{x_j \in C_i} \|x_j - u_i\|_2^2\right)$

$= 2\lambda u_i - \sum_{x_j \in C_i} 2(x_j - u_i)$

$= 2\lambda u_i - 2\sum_{x_j \in C_i} x_j + 2|C_i| u_i$

$= 0$

$\Rightarrow \hat{u}_i = \frac{1}{|C_i| + \lambda} \sum_{x_j \in C_i} x_j$

(c) Suppose the meet up locations are u_i $(i = 1, \cdots, k)$.

The distance from meet up locations to campus is $\|u_i\|$, in total it is $\sum_{i=1}^{k} \|u_i\|$.

Each student travels from home to meet up locations $\sum_{i=1}^{k} \sum_{x_j \in C_i} \|x_j - u_i\|$.

The objective is $\sum_{i=1}^{k} \left(\|u_i\| + \sum_{x_j \in C_i} \|x_j - u_i\|\right)$

(d) In k-means, the cluster assignment is found by

$$\underset{i}{\operatorname{argmin}} \| x_j - \mu_i \|_2^2$$

$$= \underset{i}{\operatorname{argmin}} \left\| x_j - \frac{1}{|C_i|} \sum_{x \in C_i} x \right\|_2^2$$

$$= \underset{i}{\operatorname{argmin}} \left\langle x_j - \frac{1}{|C_i|} \sum_{x \in C_i} x \,,\, x_j - \frac{1}{|C_i|} \sum_{x \in C_i} x \right\rangle$$

$$= \underset{i}{\operatorname{argmin}} \left(\langle x_j, x_j \rangle + \frac{1}{|C_i|^2} \left\langle \sum_{x \in C_i} x, \sum_{x \in C_i} x \right\rangle - \frac{2}{|C_i|} \left\langle x_j, \sum_{x \in C_i} x \right\rangle \right)$$

$$= \underset{i}{\operatorname{argmin}} \left(\langle x_j, x_j \rangle + \frac{1}{|C_i|^2} \sum_{x \in C_i} \sum_{x' \in C_i} \langle x, x' \rangle - \frac{2}{|C_i|} \sum_{x \in C_i} \langle x_j, x \rangle \right.$$

replace $\langle x_i, x_j \rangle$ with $k(x_i, x_j)$

$$\Rightarrow \underset{i}{\operatorname{argmin}} \left(k(x_j, x_j) + \frac{1}{|C_i|^2} \sum_{x \in C_i} \sum_{x' \in C_i} k(x, x') - \frac{2}{|C_i|} \sum_{x \in C_i} k(x_j, x) \right)$$

3a

3b

3c

CCA is the best, random projection is the worst. Because essentially cca uses the most information(both features and labels), pca only uses features, random projections uses nothing from the data.

3d

apple
banana
eggplant
310
15
10
298
240
80
70
60
50
40
30
20
10
0
apple
anana
gplant

apple
banana
eggplant
83
17
3
6
98
4
9
8
76
80
70
60
50
40
30
20
10
0
apple
anana
gplant

3e

--------code----------

```
class LDA_Model():

        def __init__(self,class_labels):

                ###SCALE AN IDENTITY MATRIX BY THIS TERM AND ADD TO COMPUTED
COVARIANCE MATRIX TO PREVENT IT BEING SINGULAR ###
                self.reg_cov = 0.001
                self.NUM_CLASSES = len(class_labels)



        def train_model(self,X,Y):
                """
                FILL IN CODE TO TRAIN MODEL
                MAKE SURE TO ADD HYPERPARAMTER TO MODEL

                '''

                X = np.array(X)
                Y = np.array(Y)
                self.means = np.zeros((self.NUM_CLASSES, X.shape[1]))
                X_demean = np.zeros_like(X)
```

```
        for i in range(self.NUM_CLASSES):
            self.means[i, :] = np.mean(X[Y==i], axis = 0)
            X_demean[Y==i] = X[Y==i] - self.means[i, :]

        self.covariance = 1/Y.shape[0] * X_demean.T @ X_demean +
self.reg_cov*np.eye(X.shape[1])

    def eval(self,x):
        """
        Fill in code to evaluate model and return a prediction
        Prediction should be an integer specifying a class
        '''
        pred_array = np.zeros(self.NUM_CLASSES)
        x = np.array(x)

        for i in range(self.NUM_CLASSES):
            x_demean = (x-self.means[i, :]).reshape(x.shape[0], -1)
            pred_array[i] = (x_demean.T @ inv(self.covariance) @ x_demean)[0, 0]
        return np.argmin(pred_array)
```

3f

--------code----------

```
class LDA_Model():

        def __init__(self,class_labels):

                ###SCALE AN IDENTITY MATRIX BY THIS TERM AND ADD TO COMPUTED
COVARIANCE MATRIX TO PREVENT IT BEING SINGULAR ###
                self.reg_cov = 0.001
                self.NUM_CLASSES = len(class_labels)



        def train_model(self,X,Y):
                '''
                FILL IN CODE TO TRAIN MODEL
                MAKE SURE TO ADD HYPERPARAMTER TO MODEL

                '''

                X = np.array(X)
                Y = np.array(Y)
                self.means = np.zeros((self.NUM_CLASSES, X.shape[1]))
                X_demean = np.zeros_like(X)
```

```
        for i in range(self.NUM_CLASSES):
            self.means[i, :] = np.mean(X[Y==i], axis = 0)
            X_demean[Y==i] = X[Y==i] - self.means[i, :]

        self.covariance = 1/Y.shape[0] * X_demean.T @ X_demean +
self.reg_cov*np.eye(X.shape[1])

    def eval(self,x):
        """
        Fill in code to evaluate model and return a prediction
        Prediction should be an integer specifying a class
        '''
        pred_array = np.zeros(self.NUM_CLASSES)
        x = np.array(x)

        for i in range(self.NUM_CLASSES):
            x_demean = (x-self.means[i, :]).reshape(x.shape[0], -1)
            pred_array[i] = (x_demean.T @ inv(self.covariance) @ x_demean)[0, 0]
        return np.argmin(pred_array)
```

3g

apple
banana
eggplant
311
14
10
298
239
apple
anana
gplant
80
70
60
50
40
30
20
10
0

apple
banana
eggplant
83
17
3
7
97
4
7
9
77
apple
anana
gplant
80
70
60
50
40
30
20
10
0

3h

apple
banana
eggplant
311
14
10
298
2
240
apple
anana
gplant
80
70
60
50
40
30
20
10
0

apple
banana
eggplant
83
17
3
7
97
4
9
7
77
apple
anana
gplant
80
70
60
50
40
30
20
10
0

3i

The better one should be the curve that curved most towards (0, 1), which means the area under curve (AUC) should be large for the better one.

But in this case, it seems the AUC for the two curves are similar, so it is hard to decide which one is better.

---------code----------

```
def ROC(scores, labels):
    thresholds = sorted(np.unique(scores))
    thresholds = [-float("Inf")] + thresholds + [float("Inf")]
    tps = []
    fps = []

    for thres in thresholds:
        tpr, fpr = compute_tp_fp(thres, scores, labels)
        tps.append(tpr)
        fps.append(fpr)

    return tps, fps
```

3j

The ROC curve will not change, the slight change here are due to randomness.

3k

The best model seems to be qda with dimension k=200, but the accuracy is not good, only 40% pictures are classified to the right class.

apple
banana
nectarine
plum
peach
watermelon
pear
mango
grape
orange
strawberry
pineapple
radish
carrot
potato
tomato
bellpepper
broccoli
cabbage
cauliflower
celery
eggplant
garlic
spinach
ginger
80
70
60
50
40
30
20
10
0

HW10. Yao Cai 25933599.

4.(a) $p(X=x, Z=1; \theta) = p(X=x; Z=1, \theta)\, p(Z=1; \theta)$

$$= p(X=x; u_1, \sigma_1^2) \times 0.5$$

$$= \frac{1}{2\sqrt{2\pi}\sigma_1} \exp\left(-\frac{(x-u_1)^2}{2\sigma_1^2}\right)$$

$$p(X=x, Z=2; \theta) = \frac{1}{2\sqrt{2\pi}\sigma_2} \exp\left(-\frac{(x-u_2)^2}{2\sigma_2^2}\right)$$

$$p(X=x; \theta) = \sum_z p(X=x, Z; \theta)$$

$$= p(X=x, Z=1; \theta) + p(X=x, Z=2; \theta)$$

$$= \sum_{i=1,2} \frac{1}{2\sqrt{2\pi}\sigma_i} \exp\left(-\frac{(x-u_i)^2}{2\sigma_i^2}\right)$$

$$l(X=x; \theta) = \log p(X=x; \theta) = \log\left(\sum_{j=1,2} \frac{1}{2\sqrt{2\pi}\sigma_j} \exp\left(-\frac{(x-u_j)^2}{2\sigma_j^2}\right)\right)$$

(b) Likelihood $= \prod_{i=1}^{n} p(X_i = x_i; \theta)$

$$ll = \log\left(\prod_{i=1}^{n} p(X_i = x_i; \theta)\right)$$

$$= \sum_{i=1}^{n} \log p(X_i = x_i; \theta)$$

$$= \sum_{i=1}^{n} \log\left(\sum_{j=1,2} \frac{1}{2\sqrt{2\pi}\sigma_j} \exp\left(-\frac{(x_i-u_j)^2}{2\sigma_j^2}\right)\right)$$

(c) $l(x_i;\theta) = \log p(x_i;\theta) = \log \sum_{k=1}^{2} p(x_i, k; \theta)$

$$= \log \sum_{k=1}^{2} p(x_i, k; \theta) \frac{1}{q_i(k)} q_i(k)$$

$$= \log E_{q_i}\left[p(x_i, k; \theta) \frac{1}{q_i(k)} \right]$$

$$\geq E_{q_i}\left[\log p(x_i, k; \theta) \frac{1}{q_i(k)} \right]$$

$$= \sum_{k=1}^{2} \log\left(p(x_i, k; \theta) \frac{1}{q_i(k)} \right) q_i(k)$$

$$= \sum_{k=1}^{2} \log p(x_i, k; \theta) \cdot q_i(k) + \sum_{k=1}^{2} q_i(k) \log \frac{1}{q_i(k)}$$

$$= L(x_i; \theta, q_i) + H(q_i) = F_i(\theta; q_i)$$

$$\Rightarrow l(\{x_i\}_{i=1}^{n}; \theta) = \sum_{i=1}^{n} l(x_i;\theta) \geq \sum_{i=1}^{n} F_i(\theta; q_i) = F(\theta; q)$$

(d) $F_i(\theta^t; p(k; x_i, \theta^t)) = \sum_{k=1}^{2} \log\left(p(x_i, k; \theta^t) \frac{1}{p(k; x_i, \theta^t)} \right) p(k; x_i, \theta^t)$

$$= \sum_{k=1}^{2} \left(\log p(x_i; \theta^t) \right) p(k; x_i, \theta^t)$$

$$= \log p(x_i; \theta^t) \sum_{k=1}^{2} p(k; x_i, \theta^t)$$

$$= \log p(x_i; \theta^t) = l(x_i; \theta^t)$$

$$\Rightarrow F(\theta^t; \prod_{i=1}^{n} p(k; x_i, \theta^t)) = l(\{x_i\}_{i=1}^{n}; \theta^t)$$

so $\prod_{i=1}^{n} p(k, x_i, \theta^t) = \underset{q}{\operatorname{argmax}}\, F(\theta^t; q) = q^{t+1}$

(e) $q_i^{t+1}(z_i=1) = p(z=1 \mid X=x_i;\theta^t)$

$$= \frac{p(X=x_i, z=1;\theta^t)}{p(X=x_i;\theta^t)}$$

$$= \frac{\frac{1}{2\sqrt{2\pi}\sigma_1}\exp\left(-\frac{(x_i-u_1)^2}{2\sigma_1^2}\right)}{\frac{1}{2\sqrt{2\pi}\sigma_1}\exp\left(-\frac{(x_i-u_1)^2}{2\sigma_1^2}\right)+\frac{1}{2\sqrt{2\pi}\sigma_2}\exp\left(-\frac{(x_i-u_2)^2}{2\sigma_2^2}\right)}$$

$$= \frac{\frac{1}{\sigma_1}\exp\left(-\frac{(x_i-u_1)^2}{2\sigma_1^2}\right)}{\frac{1}{\sigma_1}\exp\left(-\frac{(x_i-u_1)^2}{2\sigma_1^2}\right)+\frac{1}{\sigma_2}\exp\left(-\frac{(x_i-u_2)^2}{2\sigma_2^2}\right)}$$

obviously,

$$q_i^{t+1}(z_i=2) = 1 - q_i^{t+1}(z_i=1)$$

This is like the softmax step where we assign "soft" clusters for each x_i

Given a point x_i, the probability it belongs to cluster 1 is $p(z=1 \mid X=x_i;\theta^t)$
the probability it belongs to cluster 2 is $p(z=2 \mid X=x_i;\theta^t)$

If x_i is closer to u_1, then it will be very likely that
$p(z=1 \mid X=x_i;\theta^t) > p(z=2 \mid X=x_i;\theta^t)$

(f) $L(\theta; q^{t+1})$

$$= \sum_{i=1}^{n} L(x_i; \theta, q_i^{t+1})$$

$$= \sum_{i=1}^{n} \sum_{k=1}^{2} \log p(x_i, k; \theta) \cdot q_i^{t+1}(k)$$

$$= \sum_{i=1}^{n} \sum_{k=1}^{2} \log\left(\frac{1}{2\sqrt{2\pi}\sigma_k} \exp\left(-\frac{(x_i-u_k)^2}{2\sigma_k^2}\right)\right) \cdot q_i^{k+1}(k)$$

$$= \sum_{i=1}^{n} \sum_{k=1}^{2} \left(-\log 2\sqrt{2\pi} - \log\sigma_k - \frac{(x_i-u_k)^2}{2\sigma_k^2}\right) \cdot q_i^{k+1}(k)$$

$$= \sum_{i=1}^{n} \left(\left(-\log 2\sqrt{2\pi} - \log\sigma_1 - \frac{(x_i-u_1)^2}{2\sigma_1^2}\right) q_i^{k+1} + \left(-\log 2\sqrt{2\pi} - \log\sigma_2 - \frac{(x_i-u_2)^2}{2\sigma_2^2}\right)(1-q_i^{k+1})\right)$$

$$= C - \sum_{i=1}^{n} \left[q_i^{k+1}\left(\frac{(x_i-u_1)^2}{2\sigma_1^2} + \log\sigma_1\right) + (1-q_i^{k+1})\left(\frac{(x_i-u_2)^2}{2\sigma_2^2} + \log\sigma_2\right)\right]$$

(g) $$\frac{\partial L}{\partial u_1} = +\sum_{i=1}^{n} q_i^{k+1} \frac{2(x_i-u_1)}{2\sigma_1^2}$$

$$= -\frac{\sum_{i=1}^{n} q_i^{k+1}(u_1-x_i)}{\sigma_1^2}$$

$$\frac{\partial L}{\partial u_2} = +\sum_{i=1}^{n} (1-q_i^{k+1}) \frac{2(x_i-u_2)}{2\sigma_2^2}$$

$$= -\frac{\sum_{i=1}^{n} (1-q_i^{k+1})(u_2-x_i)}{\sigma_2^2}$$

$$\frac{\partial L}{\partial \sigma_1} = -\sum_{i=1}^{n} q_i^{k+1}\left(\frac{(x_i-u_1)^2}{2\sigma_1^3} \cdot (-2) + \frac{1}{\sigma_1}\right)$$

$$= \frac{\sum_{i=1}^{n} q_i^{k+1}(x_i-u_1)^2}{\sigma_1^3} - \frac{\sum_{i=1}^{n} q_i^{k+1}}{\sigma_1}$$

$$\frac{\partial L}{\partial \sigma_2} = \frac{\sum_{i=1}^{n} (1-q_i^{k+1})(x_i-u_2)^2}{\sigma_2^3} - \frac{\sum_{i=1}^{n} (1-q_i^{k+1})}{\sigma_2}$$

(h) $\frac{\partial L}{\partial u_1} = 0 \Rightarrow u_1^{t+1} = \frac{\sum_{i=1}^{n} q_i^{t+1} x_i}{\sum_{i=1}^{n} q_i^{t+1}}$

$$\frac{\partial L}{\partial u_2} = 0 \Rightarrow u_2^{t+1} = \frac{\sum_{i=1}^{n} (1-q_i^{t+1}) x_i}{\sum_{i=1}^{n} (1-q_i^{t+1})}$$

$$\frac{\partial L}{\partial \sigma_1} = 0 \Rightarrow (\sigma_1^2)^{t+1} = \frac{\sum_{i=1}^{n} q_i^{t+1} (x_i - u_1^{t+1})^2}{\sum_{i=1}^{n} q_i^{t+1}}$$

$$\frac{\partial L}{\partial \sigma_2} = 0 \Rightarrow (\sigma_2^2)^{t+1} = \frac{\sum_{i=1}^{n} (1-q_i^{t+1})(x_i - u_2^{t+1})^2}{\sum_{i=1}^{n} (1-q_i^{t+1})}$$

These are like centroid update in k-means, except that in k-means, we only update "mean", here we use a Guassian model so we can model and update both mean and variance.

Also each point x_i is weighted by the probability that it belongs to the cluster

(i) $l(\{x_i\}_{i=1}^n; u) = \log\left(\prod_{i=1}^n p(X = x_i; u)\right)$

$$= \sum_{i=1}^n \log(p(X = x_i; u))$$

$$= \sum_{i=1}^n \log\left(\frac{1}{2}\frac{e^{-\frac{1}{2}(x-u)^2}}{\sqrt{2\pi}} + \frac{1}{2}\frac{e^{-\frac{1}{2}(x+u)^2}}{\sqrt{2\pi}}\right)$$

(j) $q_i^{t+1}(z_i = 1) = p(k; x_i, u_t)$

$$= \frac{p(k, x_i; u_t)}{p(x_i; u_t)}$$

$$= \frac{p(k, x_i; u_t)}{\sum_{k=1}^2 p(k, x_i; u_t)}$$

$$= \frac{e^{-\frac{1}{2}(x_i - u_t)^2}}{e^{-\frac{1}{2}(x_i - u_t)^2} + e^{-\frac{1}{2}(x_i + u_t)^2}}$$

(k) $L(u; q^{t+1}) = C - \sum_{i=1}^n \left[q_i^{t+1}\left(\frac{(x_i - u)^2}{2}\right) + (1 - q_i^{t+1})\left(\frac{(x_i + u)^2}{2}\right)\right]$

$$\frac{\partial L}{\partial u} = \sum_{i=1}^n + \frac{q_i^{t+1} \cdot 2(x_i - u)}{2} - \frac{(1 - q_i^{t+1})\, 2(x_i + u)}{2}$$

$$= \sum_{i=1}^n q_i^{t+1}(x_i - u) + (q_i^{t+1} - 1)(x_i + u)$$

$$= \sum_{i=1}^n (x_i(2q_i^{t+1} - 1) - u) = 0$$

$$\Rightarrow u_{t+1} = \frac{1}{n}\sum_{i=1}^n x_i(2q_i^{t+1} - 1)$$

$$(l)\ \frac{\partial}{\partial u}\left(\frac{1}{n}l\right) = \frac{\partial}{\partial u}\left(\frac{1}{n}\sum_{i=1}^{n}\log\left(\frac{1}{2}\frac{e^{-\frac{1}{2}(x_i-u)^2}}{\sqrt{2\pi}} + \frac{1}{2}\frac{e^{-\frac{1}{2}(x_i+u)^2}}{\sqrt{2\pi}}\right)\right)$$

$$= \frac{\partial}{\partial u}\left(\frac{1}{n}\sum_{i=1}^{n}\left(-\log 2\sqrt{2\pi} + \log\left(e^{-\frac{1}{2}(x_i-u)^2} + e^{-\frac{1}{2}(x_i+u)^2}\right)\right)\right)$$

$$= \frac{1}{n}\sum_{i=1}^{n}\frac{1}{e^{-\frac{1}{2}(x_i-u)^2} + e^{-\frac{1}{2}(x_i+u)^2}}\left(e^{-\frac{1}{2}(x_i-u)^2}\cdot(x_i-u) + e^{-\frac{1}{2}(x_i+u)^2}\cdot(-x_i-u)\right)$$

$$= \frac{1}{n}\sum_{i=1}^{n}\left(x_i\frac{e^{-\frac{1}{2}(x_i-u)^2} - e^{-\frac{1}{2}(x_i+u)^2}}{e^{-\frac{1}{2}(x_i-u)^2} + e^{-\frac{1}{2}(x_i+u)^2}}\right) - u$$

$$= \frac{1}{n}\sum_{i=1}^{n}\left((2w_i-1)x_i\right) - u$$

$$u_{t+1}^{GA} = u_t^{GA} + \alpha\frac{\partial}{\partial u}\left(\frac{1}{n}l\right)\Big|_{u=u_t^{GA}}$$

$$= u_t^{GA} + \frac{\alpha}{n}\sum_{i=1}^{n}\left((2w_i-1)x_i\right) - u_t^{GA}\cdot\alpha$$

$$= (1-\alpha)u_t^{GA} + \alpha\left[\frac{1}{n}\sum_{i=1}^{n}\left(2w_i(u_t^{GA})-1\right)x_i\right]$$

4m

when alpha=1, the GA becomes EM. When alpha<1, then GA converges slower than EM. When alpha>1, then GA can diverge.

Indeed, in the experiments, GA with alpha=0.05 converges slower than EM.

In both cases with mu=0.5 and mu=3.0, the relative estimation error by EM is small and comparable in both cases.

Results:

```
n_points: 100 , True mean:0.500, GA (final) estimate:0.510, EM
(final) estimate:0.510
n_points: 100 , True mean:3.000, GA (final) estimate:3.085, EM
(final) estimate:3.085
```

4n

When mu is large, clusters are well separated, kmeans work reasonably well, but when mu is small, clusters are not well separated, kmeans doesn't work so well.

Results:

```
True mean:0.500, GA (final) estimate:0.510, EM (final)
estimate:0.510, K-Means (final) estimate:0.591
True mean:3.000, GA (final) estimate:3.085, EM (final)
estimate:3.085, K-Means (final) estimate:3.085
```

5a

The initial guesses are not good since the clusters are close to each other and with elongated shape. EM is able to capture the clusters but “hard” version EM cannot.

5b

when clusters are well separated, initial guessed are good, since when sampling, points from each cluster have similar chances of being drawn. So it is very likely when we draw 3 initial random points from all points, we are drawing one from each cluster.

In this case, all algorithm works pretty well.

True
EM
KQDA
Kmeans
10
0
−10
−20
−10
0