Ameya Hampihallikar
CMPSC 448
HW 5

(1)

a) Centering or mean removal

$$x_i \rightarrow (x_i - \mu) \qquad \mu = \frac{1}{n}\sum_{1}^{n} x_i \qquad x_i \in \mathbb{R}^d, \ \mu \in \mathbb{R}^d$$

Centering the data will remove the mean from each feature. This allows each feature to be centered around 0 which will eliminate bias.

Scaling

$$x_i \longrightarrow a + \frac{x - \min(x)}{\max(x) - \min(x)} (b - a)$$

Scaling a feature to range $[a, b]$ allows us to keep the feature value between a minimum and maximum range. Scaling will help keep the robustness to a small standard deviation of features as well as preserving zero entries in sparse data.

Standardization

$$x_i \rightarrow \text{diag}(\sigma_1, \sigma_2 \ldots \sigma_d)(x_i - \mu)$$

Standardization will help remove mean and scale the data to unit variance. This helps to remove bias. Also, standardization is good when data follows gaussian distribution.

Normalization

$$x_i \rightarrow \frac{x - \min(x)}{\max(x) - \min(x)}$$

Normalization will help shift and rescale the data such that the values lie between 0 and 1. This process ensures data is transformed such that they are either dimensionless and/or have similar distribution.

(b)

$$X = \begin{bmatrix} 1 & 2 \\ -1 & 1 \\ 0 & 1 \\ 2 & 4 \\ 3 & 1 \end{bmatrix} \quad n=5 \quad d=2 \quad a=0 \quad b=1$$

Centering

$$\mu_1 = 1 \qquad \mu_2 = 9/5 = 1.8$$

$$\mu = [1,\ 1.8]$$

$$X = \begin{bmatrix} 0 & 0.2 \\ -2 & -0.8 \\ -1 & -0.8 \\ 1 & 2.2 \\ 2 & -0.8 \end{bmatrix}$$

Scaling $a=0 \quad b=1$

$\min(x_1) = -1 \quad \max(x_1) = 3$
$\min(x_2) = 1 \quad \max(x_2) = 4$

$$X = a + \frac{x - \min(x)}{\max(x) - \min(x)}(b-a) = \frac{x - \min(x)}{\max(x) - \min(x)}$$

$$X = \begin{bmatrix} 0.5 & 0.33 \\ 0 & 0 \\ 0.25 & 0 \\ 0.75 & 1 \\ 1 & 0 \end{bmatrix}$$

Standardization

$\mu = [1, 1.8] \quad \sigma = [1.414, 1.166]$

$$X = \frac{X - \mu}{\sigma}$$

$$X = \begin{bmatrix} 0 & 0.172 \\ -1.414 & -0.686 \\ -0.707 & -0.686 \\ 0.707 & 1.887 \\ 1.414 & -0.686 \end{bmatrix}$$

Normalization

$\min(x_1) = -1 \quad \max(x_1) = 3$
$\min(x_2) = 1 \quad \max(x_2) = 4.$

$$X = \frac{X - \min(x)}{\max(x) - \min(x)}$$

$$X = \begin{bmatrix} 0.5 & 0.33 \\ 0 & 0 \\ 0.25 & 0 \\ 0.75 & 1 \\ 1 & 0 \end{bmatrix}$$

PROBLEM 2

Part 1

```
[40]
# n = 1
pca = PCA(n_components=1)
pca.fit(X)
print('For n_components = 1')
print('Explained Variance= ', pca.explained_variance_)
print('Explained Variance Ratio= ', pca.explained_variance_ratio_)
print()

# n = 2
pca = PCA(n_components=2)
pca.fit(X)
print('For n_components = 2')
print('Explained Variance= ', pca.explained_variance_)
print('Explained Variance Ratio= ', pca.explained_variance_ratio_)
print()

#  n = 3
pca = PCA(n_components=3)
pca.fit(X)
print('For n_components = 3')
print('Explained Variance= ', pca.explained_variance_)
print('Explained Variance Ratio= ', pca.explained_variance_ratio_)
print()


#  n = 4
pca = PCA(n_components=4)
pca.fit(X)
print('For n_components = 4')
print('Explained Variance= ', pca.explained_variance_)
print('Explained Variance Ratio= ', pca.explained_variance_ratio_)
print()
```

```
For n_components = 1
Explained Variance=  [4.22824171]
Explained Variance Ratio=  [0.92461872]

For n_components = 2
Explained Variance=  [4.22824171 0.24267075]
Explained Variance Ratio=  [0.92461872 0.05306648]

For n_components = 3
Explained Variance=  [4.22824171 0.24267075 0.0782095 ]
Explained Variance Ratio=  [0.92461872 0.05306648 0.01710261]

For n_components = 4
Explained Variance=  [4.22824171 0.24267075 0.0782095  0.02383509]
Explained Variance Ratio=  [0.92461872 0.05306648 0.01710261 0.00521218]
```

For the first principal component variance in the data is explained by 92.461872%

fraction of variation explained in data decreases exponentially to close to zero as k increases.

Part 2

[41]

```
# centering the data
centered = np.array(X)
centered = centered - centered.mean(axis = 0)
print(centered[0])

print("After centering")
print()

# n = 1
pca_centered = PCA(n_components=1)
pca_centered.fit(centered)
print('For n_components = 1')
print('Explained Variance= ', pca_centered.explained_variance_)
print('Explained Variance Ratio= ', pca_centered.explained_variance_ratio_)
print()

# n = 2
pca_centered = PCA(n_components=2)
pca_centered.fit(centered)
print('For n_components = 2')
print('Explained Variance= ', pca_centered.explained_variance_)
print('Explained Variance Ratio= ', pca_centered.explained_variance_ratio_)
print()

#  n = 3
pca_centered = PCA(n_components=3)
pca_centered.fit(centered)
print('For n_components = 3')
print('Explained Variance= ', pca_centered.explained_variance_)
print('Explained Variance Ratio= ', pca_centered.explained_variance_ratio_)
print()


#  n = 4
pca_centered = PCA(n_components=4)
pca_centered.fit(centered)
print('For n_components = 4')
print('Explained Variance= ', pca_centered.explained_variance_)
print('Explained Variance Ratio= ', pca_centered.explained_variance_ratio_)
print()
```

```
[-0.74333333  0.44266667 -2.358      -0.99933333]
After centering

For n_components = 1
Explained Variance=  [4.22824171]
Explained Variance Ratio=  [0.92461872]

For n_components = 2
Explained Variance=  [4.22824171 0.24267075]
Explained Variance Ratio=  [0.92461872 0.05306648]

For n_components = 3
Explained Variance=  [4.22824171 0.24267075 0.0782095 ]
Explained Variance Ratio=  [0.92461872 0.05306648 0.01710261]

For n_components = 4
Explained Variance=  [4.22824171 0.24267075 0.0782095  0.02383509]
Explained Variance Ratio=  [0.92461872 0.05306648 0.01710261 0.00521218]
```

Perform scaling on data

[50]

```
min_max_scaler = MinMaxScaler()
scaled = min_max_scaler.fit_transform(X)


print("After normalization")
print()

# n = 1
pca_scaled = PCA(n_components=1)
pca_scaled.fit(scaled)
print('For n_components = 1')
print('Explained Variance= ', pca_scaled.explained_variance_)
print('Explained Variance Ratio= ', pca_scaled.explained_variance_ratio_)
print()

# n = 2
pca_scaled = PCA(n_components=2)
pca_scaled.fit(scaled)
print('For n_components = 2')
print('Explained Variance= ', pca_scaled.explained_variance_)
print('Explained Variance Ratio= ', pca_scaled.explained_variance_ratio_)
print()

#  n = 3
pca_scaled = PCA(n_components=3)
pca_scaled.fit(scaled)
print('For n_components = 3')
print('Explained Variance= ', pca_scaled.explained_variance_)
print('Explained Variance Ratio= ', pca_scaled.explained_variance_ratio_)
print()


#  n = 4
pca_scaled = PCA(n_components=4)
pca_scaled.fit(scaled)
print('For n_components = 4')
print('Explained Variance= ', pca_scaled.explained_variance_)
print('Explained Variance Ratio= ', pca_scaled.explained_variance_ratio_)
print()
```

```
After normalization

For n_components = 1
Explained Variance=  [0.23245325]
Explained Variance Ratio=  [0.84136038]

For n_components = 2
Explained Variance=  [0.23245325 0.0324682 ]
Explained Variance Ratio=  [0.84136038 0.11751808]

For n_components = 3
Explained Variance=  [0.23245325 0.0324682  0.00959685]
Explained Variance Ratio=  [0.84136038 0.11751808 0.03473561]

For n_components = 4
Explained Variance=  [0.23245325 0.0324682  0.00959685 0.00176432]
Explained Variance Ratio=  [0.84136038 0.11751808 0.03473561 0.00638592]
```

Perform standardization on data

[42]

```
standard = np.array(X)
standard = (standard - standard.mean(axis = 0))/ standard.std(axis = 0)
print(standard[0])

print("After standardization")
print()

# n = 1
pca_standard = PCA(n_components=1)
pca_standard.fit(standard)
print('For n_components = 1')
print('Explained Variance= ', pca_standard.explained_variance_)
print('Explained Variance Ratio= ', pca_standard.explained_variance_ratio_)
print()

# n = 2
pca_standard = PCA(n_components=2)
pca_standard.fit(standard)
print('For n_components = 2')
print('Explained Variance= ', pca_standard.explained_variance_)
print('Explained Variance Ratio= ', pca_standard.explained_variance_ratio_)
print()

#  n = 3
pca_standard = PCA(n_components=3)
pca_standard.fit(standard)
print('For n_components = 3')
print('Explained Variance= ', pca_standard.explained_variance_)
print('Explained Variance Ratio= ', pca_standard.explained_variance_ratio_)
print()


#  n = 4
pca_standard = PCA(n_components=4)
pca_standard.fit(standard)
print('For n_components = 4')
print('Explained Variance= ', pca_standard.explained_variance_)
print('Explained Variance Ratio= ', pca_standard.explained_variance_ratio_)
print()
```

```
[-0.90068117  1.01900435 -1.34022653 -1.3154443 ]
After standardization

For n_components = 1
Explained Variance=  [2.93808505]
Explained Variance Ratio=  [0.72962445]

For n_components = 2
Explained Variance=  [2.93808505 0.9201649 ]
Explained Variance Ratio=  [0.72962445 0.22850762]

For n_components = 3
Explained Variance=  [2.93808505 0.9201649  0.14774182]
Explained Variance Ratio=  [0.72962445 0.22850762 0.03668922]

For n_components = 4
Explained Variance=  [2.93808505 0.9201649  0.14774182 0.02085386]
Explained Variance Ratio=  [0.72962445 0.22850762 0.03668922 0.00517871]
```

Perform normalization of data

[55]

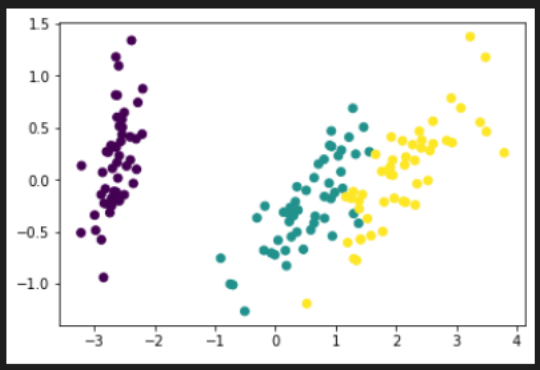
```
normalized = np.array(X)
normalized = (normalized - normalized.min(axis = 0))/(normalized.max(axis = 0) - normalized.min(axis = 0))
print(normalized[0])
print()

print("After normalization")
print()

# n = 1
pca_normalized = PCA(n_components=1)
pca_normalized.fit(normalized)
print('For n_components = 1')
print('Explained Variance= ', pca_normalized.explained_variance_)
print('Explained Variance Ratio= ', pca_normalized.explained_variance_ratio_)
print()

# n = 2
pca_normalized = PCA(n_components=2)
pca_normalized.fit(normalized)
print('For n_components = 2')
print('Explained Variance= ', pca_normalized.explained_variance_)
print('Explained Variance Ratio= ', pca_normalized.explained_variance_ratio_)
print()

#  n = 3
pca_normalized = PCA(n_components=3)
pca_normalized.fit(normalized)
print('For n_components = 3')
print('Explained Variance= ', pca_normalized.explained_variance_)
print('Explained Variance Ratio= ', pca_normalized.explained_variance_ratio_)
print()


#  n = 4
pca_normalized = PCA(n_components=4)
pca_normalized.fit(normalized)
print('For n_components = 4')
print('Explained Variance= ', pca_normalized.explained_variance_)
print('Explained Variance Ratio= ', pca_normalized.explained_variance_ratio_)
print()
```

```
[0.22222222 0.625      0.06779661 0.04166667]

After normalization

For n_components = 1
Explained Variance=  [0.23245325]
Explained Variance Ratio=  [0.84136038]

For n_components = 2
Explained Variance=  [0.23245325 0.0324682 ]
Explained Variance Ratio=  [0.84136038 0.11751808]

For n_components = 3
Explained Variance=  [0.23245325 0.0324682  0.00959685]
Explained Variance Ratio=  [0.84136038 0.11751808 0.03473561]

For n_components = 4
Explained Variance=  [0.23245325 0.0324682  0.00959685 0.00176432]
Explained Variance Ratio=  [0.84136038 0.11751808 0.03473561 0.00638592]
```

From the above images, we can see that the best explained variance ratio is given by standardization. There is a gradual decrease in the fraction of variance for normalization and scaling and there is no change for centering.

Part 3

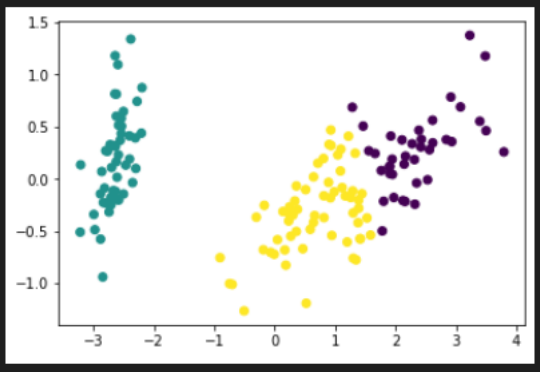
```
[52]
pca = PCA(n_components=2)
new_data = pca.fit(X).transform(X)
new_x = new_data[:,0]
new_y = new_data[:,1]
plt.scatter(new_x, new_y, c = y)
plt.show()
```



Three clusters are shown.

Part 4

```
[51]
k_means_pp = KMeans(n_clusters=3, init='k-means++')
k_means_pp.fit(new_data)
kmeans_x = new_data[:,0]
kmeans_y = new_data[:,1]
plt.scatter(kmeans_x, kmeans_y, c = k_means_pp.labels_)
plt.show()
```




```
[-]
```

k-means ++ offers better clustering compared to PCA as there is less overlap in the clusters as seen between the 2 images.

(3) First find gradient by fixing $V^{(t)}$ and taking gradient step for $U^{(t)}$ & vice versa

(1) Objective function

$$\min_{U,V} \left[f(U,V) = \sum_{(i,j)\in\Omega} (R_{i,j} - u_i^T v_j)^2 + \alpha \sum_{i=1}^{n} \|\mu_i\|_2^2 + \beta \sum_{j=1}^{m} \|v_j\|_2^2 \right]$$

$$\frac{\partial f}{\partial u_i} = \sum_{j\in|m|\, i,j\in\Omega} 2(R_{i,j} - \mu_i^T v_j)(-v_j) + 2\alpha\mu_i = 0$$

$$\Rightarrow \sum_{j\in|m|,\, i,j\in\Omega} 2(R_{i,j} - \mu_i^T v_j)(v_j) = 2\alpha\mu_i$$

$$\Rightarrow \sum_{j\in|m|\, i,j\in\Omega} R_{i,j} v_j = \left(\alpha I + \sum_{j\in|m|\, i,j\in\Omega} v_j v_j^T\right) \mu_i$$

$$\Rightarrow \mu_i = \left(\alpha I + \sum_{j\in|m|\, i,j\in\Omega} v_j v_j^T\right)^{-1} \sum_{j\in|m|\, i,j\in\Omega} R_{i,j} v_j$$

$$\frac{\partial f}{\partial v_j} = \sum_{j\in|m|\, i,j\in\Omega} 2(R_{ij} - \mu_i^T v_j)(-\mu_i) + 2\beta v_j = 0$$

$$\Rightarrow \sum (R_{i,j}\, \mu_i) = 2\beta v_j + \sum_{j\in|m|\, i,j\in\Omega} \mu_i \mu_i^T v_j$$

$$\Rightarrow v_j = \left(\sum_{i\in|u|\, i,j\in\Omega} \mu_i \mu_i^T + \beta I\right)^{-1} \sum_{i=|u|\, (i,j)\in\Omega} R_{i,j}\, \mu_i$$

By using Alternating Minimization algorithm we find that $U^{(0)}$ and $V^{(0)}$ are initialized to zero the update rule will give zero values.

(2) $\alpha = 0$

through the gradient calculation we find that $\sum_{i \in [m],\, i,j \in \Omega} v_j v_j^T$ must be invertible and have full rank of k.

Also, when $\beta = 0$

we can say $\sum_{j \in [m],\, i,j \in \Omega} \mu_i \mu_i^T$ has to be invertible and number of elements has to be equal to the rank.

Hence, we will require at least k movies in the sum so that they can be rated by i^{th} user. Thus for all users this should be true, Therefore we will have k movies and k users. Each movie must be rated by k users.

(3) Gradient for μ_i

$$\frac{df}{d\mu_i} = \sum_{j \in [m],\, i,j \in \Omega} 2(R_{i,j} - \mu_i^T v_j)(-v_j) + 2\alpha\mu_i$$

Update rule for $\mu_i^{(t)}$ at t^{th} iteration of SGD

$$\mu_i^{(t)} = \mu_i - \eta_t (2(\mu_i^T v_j - R_{ij})(v_j) + 2\alpha\mu_i)$$

Gradient for v_j

$$\frac{df}{dv_j} = \sum_{i \in [m],\, i,j \in \Omega} 2(R_{i,j} - \mu_i^T v_j)(-\mu_i) + 2\beta v_j)$$

Update rule for $v_j^{(t)}$ at t^{th} iteration of SGD

$$v_j^{(t)} = v_j - \eta_t (2(\mu_i^T v_j - R_{i,j})\mu_i + 2\beta v_j)$$

4

1) 1st policy π_1 | 2nd policy π_2

$s_0 \to a_1$	$s_0 \to a_2$
$s_1 \to a_0$	$s_0 \to a_0$
$s_2 \to a_0$	$s_0 \to a_0$
$s_3 \to a_0$	$s_0 \to a_0$

$$V_\pi(s) = \sum_a \pi(a|s) \sum_{s'r} p(s', r|s, a)[r + \gamma V_\pi(s')]$$

$\gamma = 0 \quad p = q = 1/2$

$V_\pi(s_0) = 0$
$V_\pi(s_1) = 0$
$V_\pi(s_2) = 1$
$V_\pi(s_3) = 10$

2) Required equation $\Rightarrow$ Bellman's optimality.

$$V_*(s) = \max_a q_{\pi a}(s, a)$$

$$= \max_a \sum_{s,r} p(s', r|s, a)[r + \gamma V_*(s')]$$

$$V_*(s_0) = \max_{a_1, a_2} \sum_{s,r} (0 + \gamma V_*(s_1) + 0 + \gamma V_*(s_2))$$

$$= \gamma \max_{a_1, a_2} \{V_*(s_1), V_*(s_2)\}$$

$$V_*(s_1) = (1-p)(0 + \gamma V_*(s_1)) + p(0 + \gamma V_*(s_3))$$
$$= \gamma[(1-p) V_*(s_1) + p V_*(s_3)].$$

$V^*(s_2) = (1-q)(1+\gamma V^*(s_0)) + q(1+V^*(s_3))$

$V^*(s_3) = 10 + \gamma V^*(s_0)$

③ We know that

$$\pi^*(s_0) = \underset{a_1, a_2}{\operatorname{argmax}} \gamma \sum_{s'} P(s'|s,a) V^*(s')$$

If $\gamma = 0$

π^* is not unique. Hence there is no guarentee $\pi^*(s_0) = a_2$

If $\gamma = 1$

$V^*(s_2)$ has to be greater than $V^*(s_1)$ such that $\pi^*(s_0) = a_0$ and $q \in [0,1]$

When $p=0$ means that $V^*(s_1) = \gamma V^*(s_1) = 0$ $V^*(s_2) = (1-p)(1+\gamma V^*(s_0)) + q(1+V^*(s_3)) \geq 1$

Therefore $V^*(s_2) > V^*(s_1)$ and $\pi^*(s_1) = a_2$ for $\gamma \in [0,1)$ and $q \in [0,1]$

④ $p = q = 0.25$ $\gamma = 0.9$

π^* $s_0 \rightarrow a_1$ $V^*(s_0) = 14.185$

$s_1 \rightarrow a_0$ $V^*(s_1) = 15.761$

$s_2 \rightarrow a_0$ $V^*(s_2) = 15.69$

$s_3 \rightarrow a_0$ $V^*(s_3) = 22.76$

(5) From SARSA Algorithm, a' and s' is chosen before updating Q function.

On the other hand Q learning will update Q function before choosing the action for next iteration.

Hence they will not be the same.