Agreement: This assignment represents my own work. I did not work on this assignment with others. All coding was done by myself.

Q1

(a) $K_1(x, z)$ is a kernel.

Prove:

$$K_1(x,z) = aK(x,z) = a < \Phi(x), \Phi(z) >_{\mathcal{H}_k} = < \sqrt{a}\Phi(x), \sqrt{a}\Phi(z) >_{\mathcal{H}_k}$$

Where $a > 0$.

Therefore, K_1 is an inner product. Therefore, K_1 is a valid kernel.

(b) $K_2(x, z)$ is not necessarily a kernel.

Counterexample:

When $a = -1$ and $K(x,z) = < x, z >$ (linear kernel),

$$K_2(x,z) = aK(x,z) = - < x, z >$$

Assuming there is a dataset of two 1D samples: $x_1 = 1$ and $x_2 = -1$, the Gram Matrix of K_2 can be obtained as below:

$$\boldsymbol{K} = \begin{pmatrix} k_2(x_1,x_1) & k_2(x_1,x_2) \\ k_2(x_2,x_1) & k_2(x_2,x_2) \end{pmatrix} = \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix}$$

To calculate its eigenvalues,

$$\det(\boldsymbol{K} - \lambda I) = \det\left(\begin{bmatrix} -1-\lambda & 1 \\ 1 & -1-\lambda \end{bmatrix}\right) = 0$$

$$(-1-\lambda)^2 - 1 = \lambda^2 + 2\lambda = 0 \Longrightarrow \lambda = 0, -2$$

$\lambda = -2$ means the Gram Matrix of K_2 is not positive semidefinite. Therefore, in this case, $K_2(x, z)$ is not a valid kernel.

(c) $K_3(x, z)$ is a kernel.

Prove:

From the lecture, we know the definition of Polynomial kernels is:

$$k(x, z) = (< x, z > + c)^d, \quad d \geq 2$$

Therefore, both $K_{31}(x, z) =< x, z >^3$ and $K_{32}(x, z) = (< x, z > -1)^2$ are valid Polynomial kernels. Also from the lecture, we know that if $k_1(x, z)$ and $k_2(x, z)$ are kernels, $k(x, z) = \alpha k_1(x, z) + \beta k_2(x, z)$ for $\alpha, \beta \geq 0$ is also a kernel. In this case, $K_3(x, z) = K_{31}(x, z) + K_{32}(x, z)$ with $\alpha = \beta = 1$. Therefore, $K_3(x, z)$ is a valid kernel.

(d) $K_4(x, z)$ is a kernel.

Prove:

From the lecture, we know the definition of Polynomial kernels is:

$$k(x, z) = (< x, z > + c)^d, \quad d \geq 2$$

Therefore, $K_{41}(x, z) =< x, z >^2$ is a valid Polynomial kernel. Also from the lecture, we know that $k(x, z) = g(x)g(z)$ for $g: \mathcal{X} \to \mathbb{R}$ is a kernel. In this case, assuming $g(x) = \exp(-\|x\|^2)$,

$$K_{42}(x, z) = \exp(-\|x\|^2) \exp(-\|z\|^2) = g(x)g(z)$$

So, $K_{42}(x, z)$ is a kernel. Therefore, $K_4(x, z) = K_{41}(x, z) + K_{42}(x, z)$ is also a kernel (This fact is mentioned in (c)).

Q2

(a) The VC dimension is d.

Prove: show that there exists a configuration that d points can be shattered.

In a d-dimensional space, there are at most d linearly independent points that can be found. Because linearly independence, each ω_i can classify x_i into $\{-1,1\}$ by changing the sign. In other word, a linear classifier without a constant term can always identify a weight vector ω that correctly classifies these d linearly independent points for any binary labeling. Therefore, in this configuration, d points can be shattered.

Prove: show that there exists no configuration that $d+1$ points can be shattered.

In a d-dimensional space, any $d+1$ points must be linearly dependent, which means there is at least one point can be represented as a linear combination of the others, which is shown as below:

$$x_{d+1} = \sum_{i=1}^{d} \alpha_i x_i$$

Where α is the relationship between x_{d+1} and other points. Therefore, $f(x_{d+1}) = sgn(\omega^T x_{d+1}) = sgn\left(\omega^T \sum_{i=1}^{d} \alpha_i x_i\right)$.

Because of this linearly dependence, $f(x_{d+1})$ will always be determined by other points, which means it cannot be classified independently.

Therefore, there is no configuration for $d+1$ points that can be shattered.

(b)

(1)

Let $V = S$, which is a d-dimensional subspace in $\mathbb{R}^n$; and then define the linear mappings $f_j : V \to \mathbb{R}$ where $f_j(x) = e_j^T x$ for $j = 1, \ldots, n$, where e_j is the unit vector of $\mathbb{R}^n$.

Because V is a d-dimensional subspace in $\mathbb{R}^n$ with $d \geq 1$, V contains vectors that have non-zero components in at least one dimension. Therefore, for each j, there exists at least one vector x in V such that $e_j^T x \neq 0$. This shows that each f_j is not always zero.

Then, the set of solutions to $f_j(x) = 0$ forms hyperplanes of V, which divide V into cells. The maximum number of cells is $C(n, d)$.

Because each cell in V corresponds to an intersection with an orthant in $\mathbb{R}^n$, the number of orthants that V intersects is at most the number of cells.

Therefore, the orthant intersection problem is reduced to cell counting problem, and the maximum number of orthants intersected by V is $C(n, d)$.

(2)

Firstly, we can build a $d \times n$ matrix M, where each column is a point. Because these points are in general position, any columns in M is linearly

independent, which means the rank of M is equal to the dimension of its column space, which is d.

The computation of patterns is equivalent to finding every unique way that a weight vector is dot product with these points to produce different sign combinations. This can be transformed into considering how the d-dimensional column space (constructed subspace) of matrix M intersects with the orthants in $\mathbb{R}^n$.

Because the constructed subspace is built by these d-dimensional points, it is nontrivial, which means it will at least intersect with one orthant.

We know from (1) that a d-dimensional subspace in $\mathbb{R}^n$ intersects with at most $C(n,d)$ orthants. Therefore, the number of distinct patterns is also at most $C(n,d)$.

(4)

When the number of points, m is equal to d, the number of patterns is calculated as below:

$$C(m,d) = C(d,d) = 2\sum_{k=0}^{d-1}\binom{d-1}{k} = 2 \times 2^{d-1} = 2^d = 2^m$$

Therefore, d points can be shattered.

When $m > d$, $C(m,d) = 2\sum_{k=0}^{d-1}\binom{m-1}{k} < 2^m$. Therefore, it cannot be shattered if the number of points is larger than d.

Therefore, the VC dimension of $\mathcal{F}_d$ is d.

(5)

When n is much larger than d, the dominant term of $C(n,d)$ is $\binom{n-1}{d-1}$,

$$\binom{n-1}{d-1} = \frac{(n-1)(n-2)\dots(n-d+1)}{(d-1)!} \approx \frac{n^{d-1}}{(d-1)!}$$

Because n is much larger than d, and d is constant, the Big-O growth rate is $\boldsymbol{O(n^{d-1})}$.

Efficacy: The complexity of the linear models increases polynomially with respect to n, and the growth rate is determined by $d-1$. Therefore, the linear models might not be expressive enough if the dataset has very complex structures (large d) and a large number of data points. They are more suitable for small dataset that has straightforward relationships.

(c)

The dimension of the vector $(\boldsymbol{k(x_1,x), k(x_2,x), \dots, k(x_n,x)})$ is $\boldsymbol{n}$, because it has n components, each of which is the result of the non-linear kernel function k applied to a pair of inputs.

We know from (b) that the VC dimension is d, which means when $n > d$, there is no configuration that can be shattered. However, by using a kernel corresponding to RKHS, the dimension of inputs, d increases to n, so the number of distinct patterns $\mathrm{C(n,d)}$ increases to $\mathrm{C(n,n)}$. Therefore, the possibility of finding a hyperplane that can linearly separate the dataset is increased.

Q3

(1)

According to the representer theorem, the optimal predictive function, f_θ^* of this problem can be expressed by a linear combination of the kernel functions applied to the training points, which is shown below:

$$f_\theta^*(x) = \sum_{i=1}^{n} \alpha_i k(x, x_i)$$

Where $k(x, x_i)$ is the kernel function corresponding to the reproducing kernel Hilbert space, and α_i is the coefficient.

Because f_θ is in the form of $\theta^T x$, the kernel corresponding to RKHS should be able to represent the functions as a similar form as $\theta^T x$, which means $k(x, x_i)$ is linear kernel. Therefore, $k(x, x_i) = x^T x_i$. Therefore, $f_\theta^*(x)$ is shown below:

$$\boldsymbol{f_\theta^*(x) = \sum_{i=1}^{n} \alpha_i x^T x_i}$$

(2)

The primal optimization problem is:

$$\min_{\omega,\zeta} \frac{1}{2}||\omega||^2 + C\sum_{i=1}^{n} g(\zeta_i)$$

With the constrain:

$$\zeta_i - y_i(\omega^T x_i) \leq 0$$

Therefore, the dual formulation can be constructed as below:

$$\mathcal{L}(\omega, \zeta, \alpha) = \frac{1}{2}||\omega||^2 + C\sum_{i=1}^{n} g(\zeta_i) + \sum_{i=1}^{n} \alpha_i(\zeta_i - y_i(\omega^T x_i))$$

Therefore,

$$\frac{\partial \mathcal{L}}{\partial \omega} = \omega + \sum_{i=1}^{n} \alpha_i(-y_i x_i) = \omega - \sum_{i=1}^{n} \alpha_i y_i x_i = 0$$

$$\omega = \sum_{i=1}^{n} \alpha_i y_i x_i$$

$$\frac{\partial \mathcal{L}}{\partial \zeta_i} = C\frac{-e^{-\zeta_i}}{1 + e^{-\zeta_i}} + \alpha_i = 0$$

$$\zeta_i = ln\frac{C - \alpha_i}{\alpha_i}$$

Substitute ω and ζ_i, the dual formulation can be derived as below:

$$\mathcal{L} = \frac{1}{2}\left\|\sum_{i=1}^{n} \alpha_i y_i x_i\right\|^2 + \sum_{i=1}^{n} Cln\frac{C}{C - \alpha_i} + \sum_{i=1}^{n} \alpha_i \left(ln\frac{C - \alpha_i}{\alpha_i} - y_i \left(\sum_{j=1}^{n} \alpha_j y_j x_j^T \, x_i \right) \right)$$

Simplifying:

$$\mathcal{L} = -\frac{1}{2}\left\|\sum_{i=1}^{n} \alpha_i y_i x_i\right\|^2 + \sum_{i=1}^{n} ClnC - \sum_{i=1}^{n} \alpha_i ln\alpha_i - \sum_{i=1}^{n} (C - \alpha_i)\ln(C - \alpha_i)$$

HW6_Q4

November 26, 2023

[1]:
```
!pip install torch
```

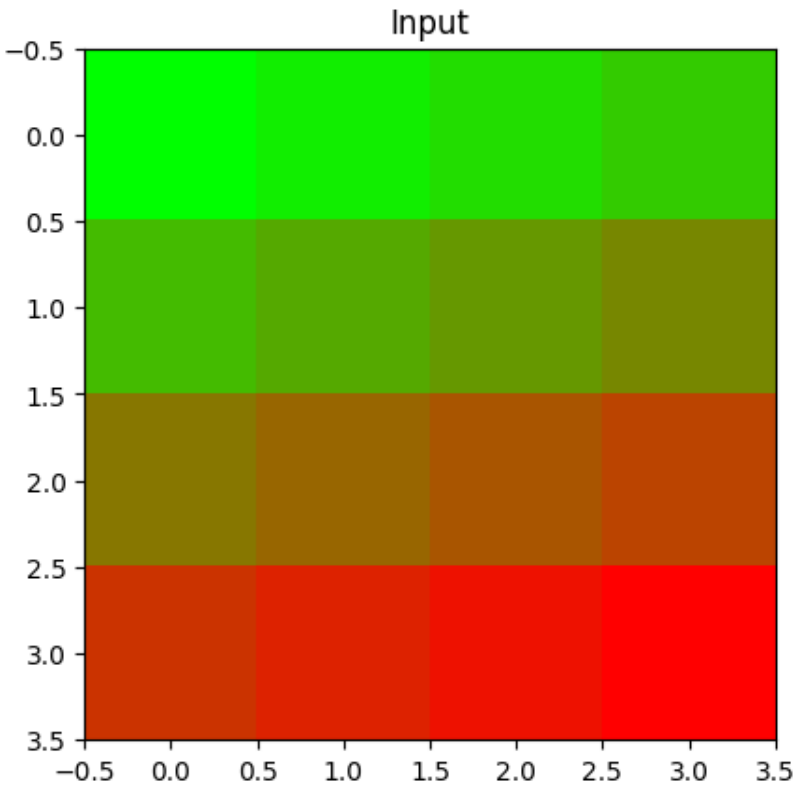
```
Requirement already satisfied: torch in
c:\users\administrator\appdata\local\programs\python\python310\lib\site-packages
(2.1.0+cu118)
Requirement already satisfied: filelock in
c:\users\administrator\appdata\local\programs\python\python310\lib\site-packages
(from torch) (3.9.0)
Requirement already satisfied: typing-extensions in
c:\users\administrator\appdata\local\programs\python\python310\lib\site-packages
(from torch) (4.8.0)
Requirement already satisfied: sympy in
c:\users\administrator\appdata\local\programs\python\python310\lib\site-packages
(from torch) (1.12)
Requirement already satisfied: networkx in
c:\users\administrator\appdata\local\programs\python\python310\lib\site-packages
(from torch) (3.0)
Requirement already satisfied: jinja2 in
c:\users\administrator\appdata\local\programs\python\python310\lib\site-packages
(from torch) (3.1.2)
Requirement already satisfied: fsspec in
c:\users\administrator\appdata\local\programs\python\python310\lib\site-packages
(from torch) (2023.4.0)
Requirement already satisfied: MarkupSafe>=2.0 in
c:\users\administrator\appdata\local\programs\python\python310\lib\site-packages
(from jinja2->torch) (2.1.3)
Requirement already satisfied: mpmath>=0.19 in
c:\users\administrator\appdata\local\programs\python\python310\lib\site-packages
(from sympy->torch) (1.3.0)
```

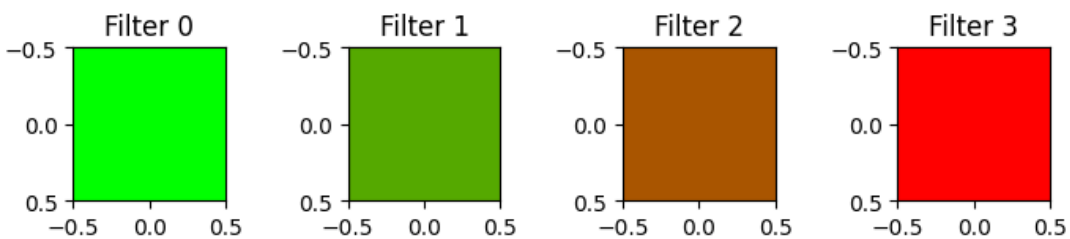
[2]:
```
import torch
import torch.nn.functional as F

torch.manual_seed(42)
height, width, channel = 4, 4, 3
num_kernels = 4

input = torch.stack([
```

```
    torch.linspace(0, 1, height*width),
    torch.linspace(1, 0, height*width),
    torch.linspace(0, 0, height*width),
]).view(channel, height, width)

kernels = torch.stack([
    torch.linspace(0, 1, num_kernels),
    torch.linspace(1, 0, num_kernels),
    torch.linspace(0, 0, num_kernels),
], dim=1).view(num_kernels, channel, 1, 1)
```

1 a.1

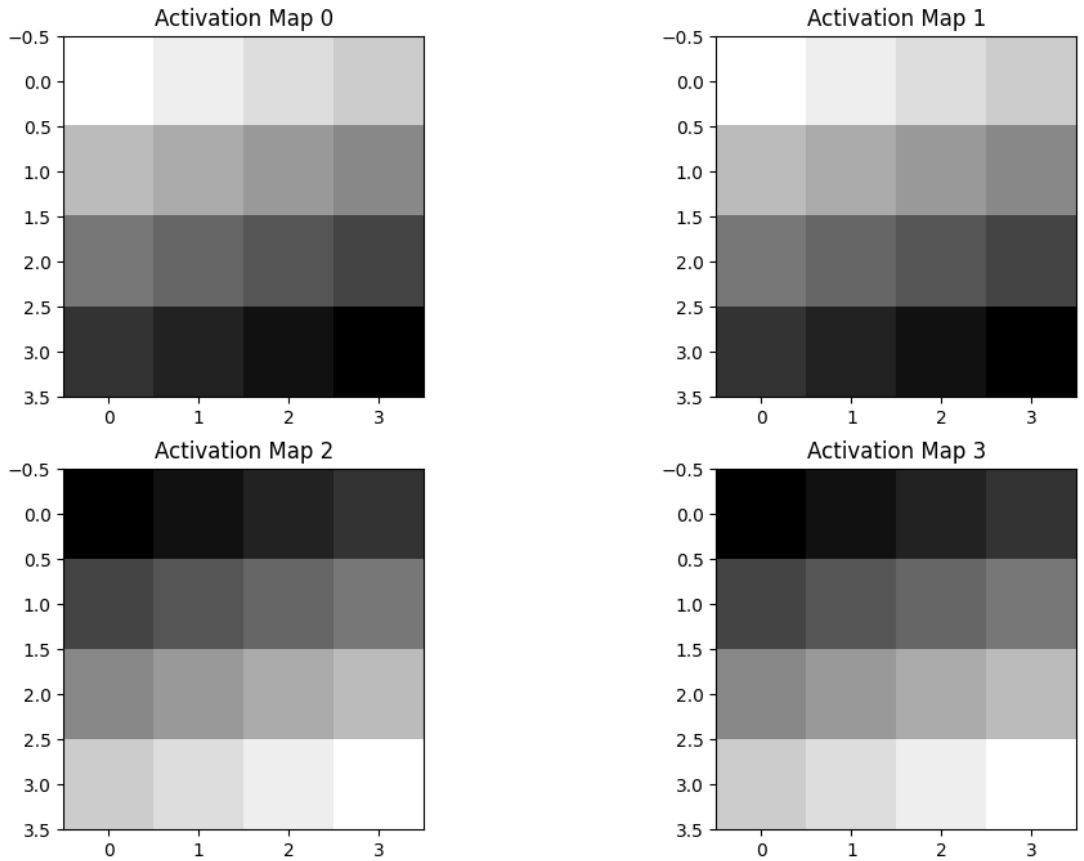
[3]:
```
import matplotlib.pyplot as plt

# visualize input
permuted_input = torch.permute(input, (1, 2, 0))
# print(permuted_input.shape)
plt.title("Input")
plt.imshow(permuted_input)

permuted_kernels = torch.permute(kernels, (0, 2, 3, 1))
# print(permuted_kernels.shape)
plt.figure(figsize=(8,4))
plt.subplots_adjust(wspace = 0.8, hspace = 0.4)
for i in range(num_kernels):
    plt.subplot(141 + i)
    plt.title("Filter " + str(i))
    plt.imshow(permuted_kernels[i])
```





2 a.2

[4]:
```
conv_output = F.conv2d(input, kernels)
# print(conv_output.shape)
plt.figure(figsize=(12,8))
for i in range(num_kernels):
    plt.subplot(221 + i)
    plt.title("Activation Map " + str(i))
    plt.imshow(conv_output[i], cmap='gray')
```



Rule 2 (The similarity between an object and itself is the maximum similarity possible) is violated.

Because for Activation Map 1 and 2, the similarity between the filter and itself is not the maximum value.

3 a.3

Because cos( ) is less than or equal to 1, CosSim meets Rule 1.

Because (x,x) = 0, cos((x,x)) = 1. Therefore, CosSim meets Rule 2.

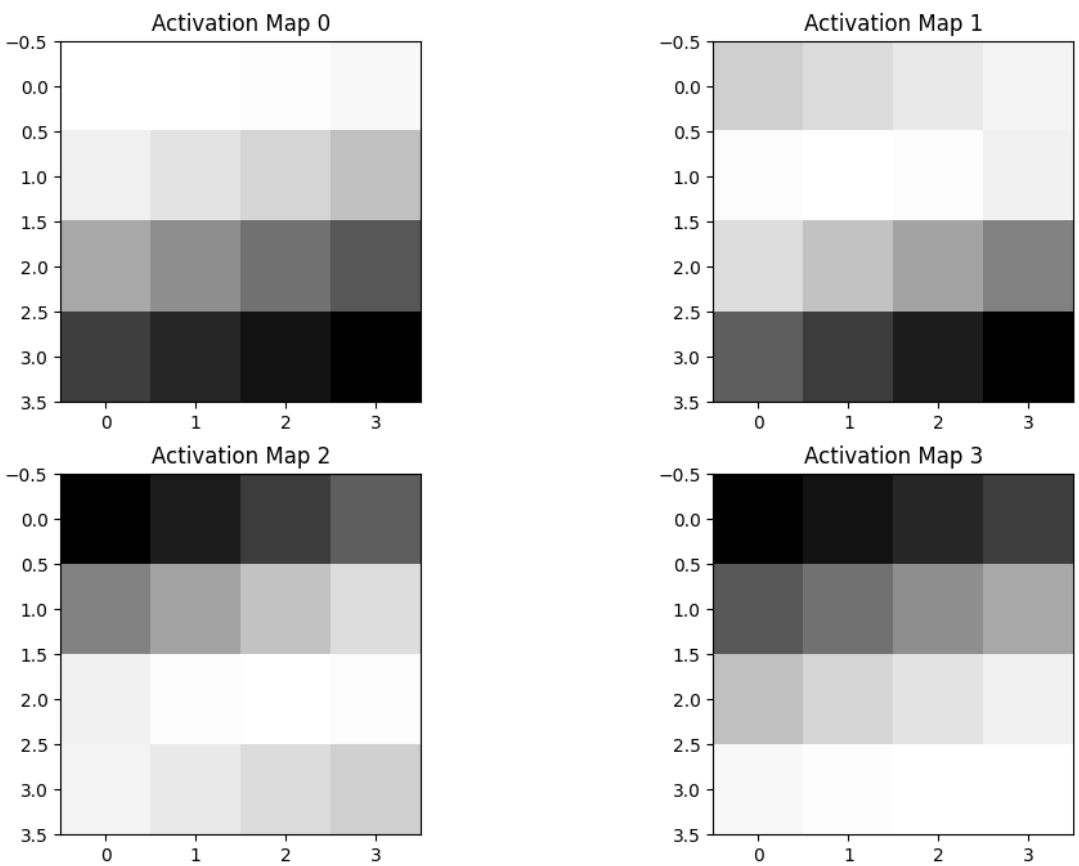
Because has a range of [0, 2pi), when x != y, cos((x,y)) < 1. Therefore, CosSim meets Rule 3.

Because is a distance metric, (x,y) is equal to (y,x). Therefore, CosSim(x, y) is equal to CosSim(y, x), which means CosSim meets Rule 4.

Therefore, CosSim is a similarity metric.

4 a.4

[5]:
```
kernels_norm = kernels.view(kernels.size(0), -1)
kernels_norm = kernels_norm / torch.norm(kernels_norm, dim=1, keepdim=True)
kernels_norm = kernels_norm.view_as(kernels)
conv_output = F.conv2d(input, kernels_norm)
ones_kernel = torch.ones((1, channel, 1, 1), dtype=torch.float32)
input_norm = torch.sqrt(F.conv2d(input**2, ones_kernel))
conv_output = conv_output / input_norm
# print(conv_output.shape)

plt.figure(figsize=(12,8))
for i in range(num_kernels):
    plt.subplot(221 + i)
    plt.title("Activation Map " + str(i))
    plt.imshow(conv_output[i], cmap='gray')
```



The activation maps reflect cosine similarity function.

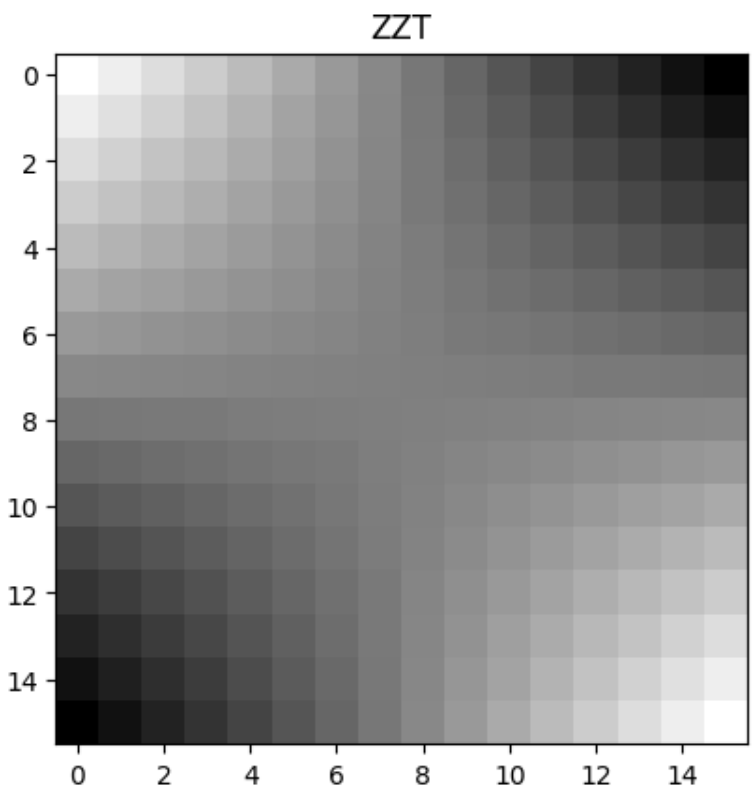
For normal convolution, the input and kernels are directly convolved, which is not similarity function,

While for cosine similarity convolution, both the input and kernels are normalized to achieve cosine similarity.

5 b.1

[6]:
```
import numpy as np

Z = input.view(channel, height * width)
Z = torch.transpose(Z, 0, 1)
ZZT = np.dot(Z, torch.transpose(Z, 0, 1))
plt.title("ZZT")
plt.imshow(ZZT, cmap='gray')
# print(ZZT)
```

[6]: <matplotlib.image.AxesImage at 0x332815a0>



Based on the visualization above, the entries of ZZT cannot be produced by a similarity function.

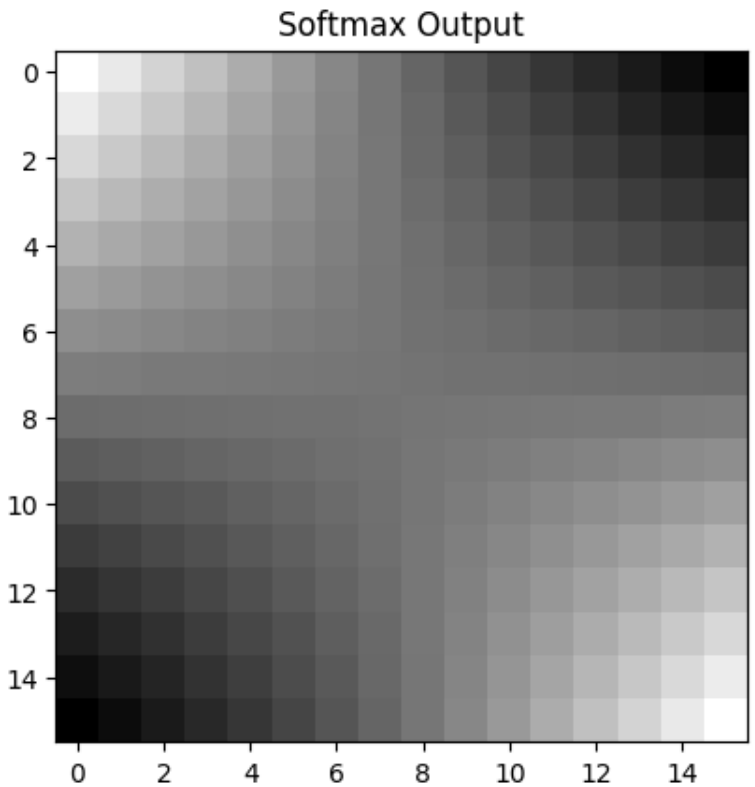
The diagonal elements represent the dot product of a vector with itself.

From the visualization above, we can see the diagonal elements are not always the maximum value. This is violated with Rule 2.

6 b.2

[7]:
```
softmax_out = F.softmax(ZZT/torch.sqrt(torch.tensor(channel)), dim=1)
# print(softmax_out)
plt.title("Softmax Output")
plt.imshow(softmax_out, cmap='gray')
```

[7]: <matplotlib.image.AxesImage at 0x3645bfd0>



Based on this visualization, the entries in this matrix cannot be produced by a similarity function.

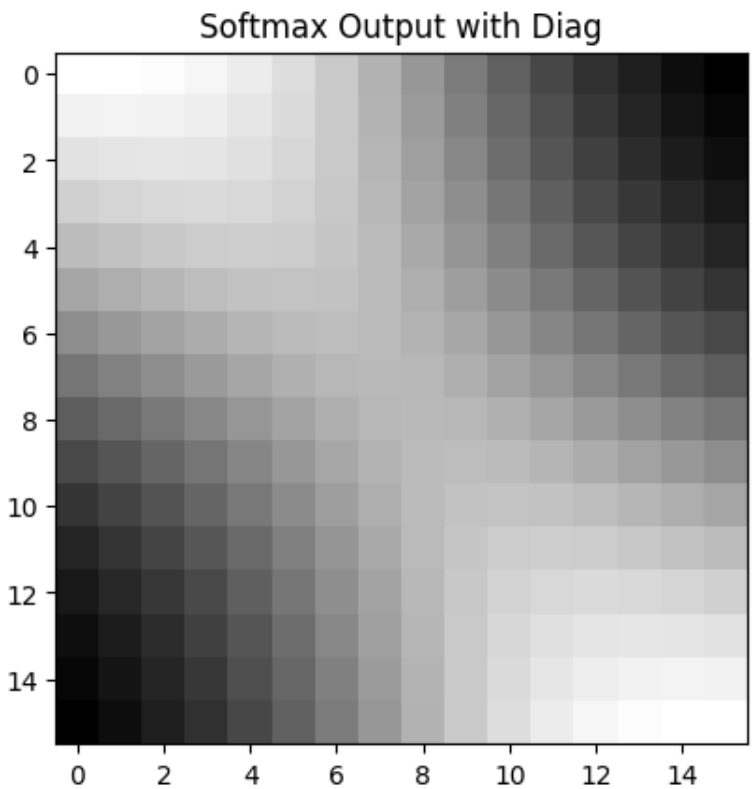
The diagonal elements represent the dot product of a vector with itself.

Same as b.1, we can see the diagonal elements are not always the maximum value. This is violated with Rule 2.

7 b.3

[8]:
```
norms = torch.norm(Z, dim=1)
diag_Z = torch.diag(1.0 / norms)
softmax_out_diag = F.softmax(np.dot(np.dot(diag_Z, ZZT), diag_Z)/torch.
 ↪sqrt(torch.tensor(channel)), dim=1)
# print(softmax_out_diag)
plt.title("Softmax Output with Diag")
plt.imshow(softmax_out_diag, cmap='gray')
```

[8]:
```
<matplotlib.image.AxesImage at 0x367db910>
```



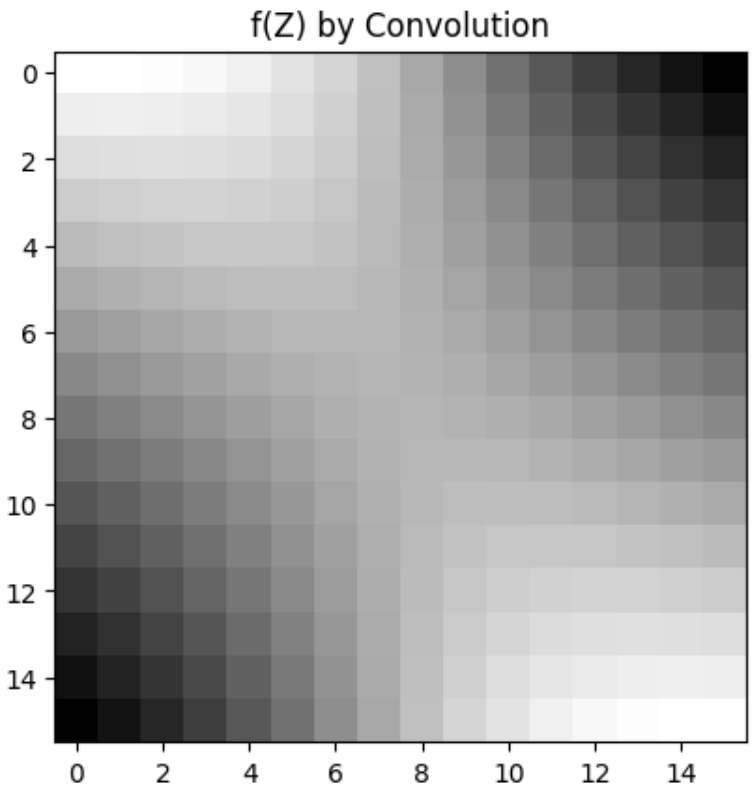
Based on this visualization, the entries in this matrix can be produced by a similarity function.

This visualization shows that the diagonal elements are always the maximum value, and it is symmetric.

8 b.4

[9]:
```
num_kernels = height * width
conv_kernels = (Z / norms[:, None]).view(num_kernels, channel, 1, 1)
conv_output = F.conv2d(input, conv_kernels)
conv_output = conv_output.view(num_kernels, num_kernels)
conv_output = torch.transpose(conv_output, 0, 1)
# print(conv_output)
plt.title("f(Z) by Convolution")
plt.imshow(conv_output, cmap='gray')
```

[9]:
```
<matplotlib.image.AxesImage at 0x36864fa0>
```



[ ]:

HW6_Q5

November 26, 2023

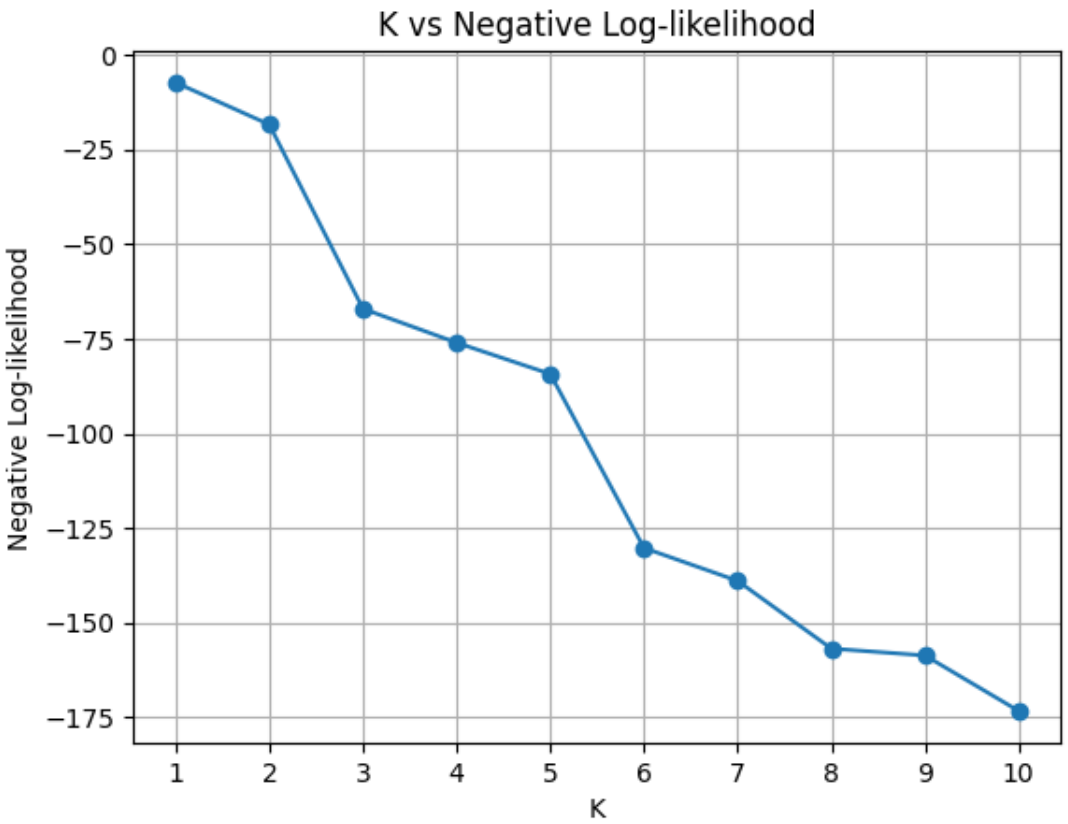
[1]:
```
from sklearn.mixture import GaussianMixture
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
import numpy as np
from scipy.stats import multivariate_normal

seed = 42

x = pd.read_csv("mall_customers.csv")
scaler = MinMaxScaler()
x = scaler.fit_transform(x)
```

[2]:
```
def GMM(data, K, max_iter=100, tol=1e-4):
    np.random.seed(seed)
    N, d = data.shape
    weight = np.full(K, 1/K)
    mean = data[np.random.choice(N, K, replace=False)]
    cov = np.array([0.01 * np.eye(d) for _ in range(K)])
    ll = 0

    for t in range(max_iter):
        # Expectation step
        r = np.zeros((N, K))
        for k in range(K):
            r[:, k] = weight[k] * multivariate_normal.pdf(data, mean=mean[k],␣
↪cov=cov[k])
        r /= r.sum(axis=1, keepdims=True)

        # Maximization step
        for k in range(K):
            sum_rk = r[:, k].sum()
            mean[k] = (data * r[:, k, np.newaxis]).sum(axis=0) / sum_rk
            diff = data - mean[k]
            cov[k] = (r[:, k, np.newaxis] * diff).T @ diff / sum_rk
            cov[k] += 1e-6 * np.eye(d)
            weight[k] = sum_rk / N
```

```
        # Compute log-likelihood
        last_11 = ll
        ll = np.sum(np.log(np.sum([w * multivariate_normal.pdf(data,␣
↪mean=mean[k], cov=cov[k]) for k, w in enumerate(weight)], axis=0)))

        if np.abs(ll - last_11) < tol:
            break

    return weight, mean, cov, ll
```

1 a.1

[3]:
```
neg_log_likelihoods = []
for i in range(1, 11):
    _, _, _, ll = GMM(x, i)
    neg_log_likelihoods.append(-ll)
# print(neg_log_likelihoods)

plt.plot(range(1, 11), neg_log_likelihoods, marker='o')
plt.xticks(range(1, 11))
plt.title('K vs Negative Log-likelihood')
plt.xlabel('K')
plt.ylabel('Negative Log-likelihood')
plt.grid(True)
plt.show()

proper_k = 6
```

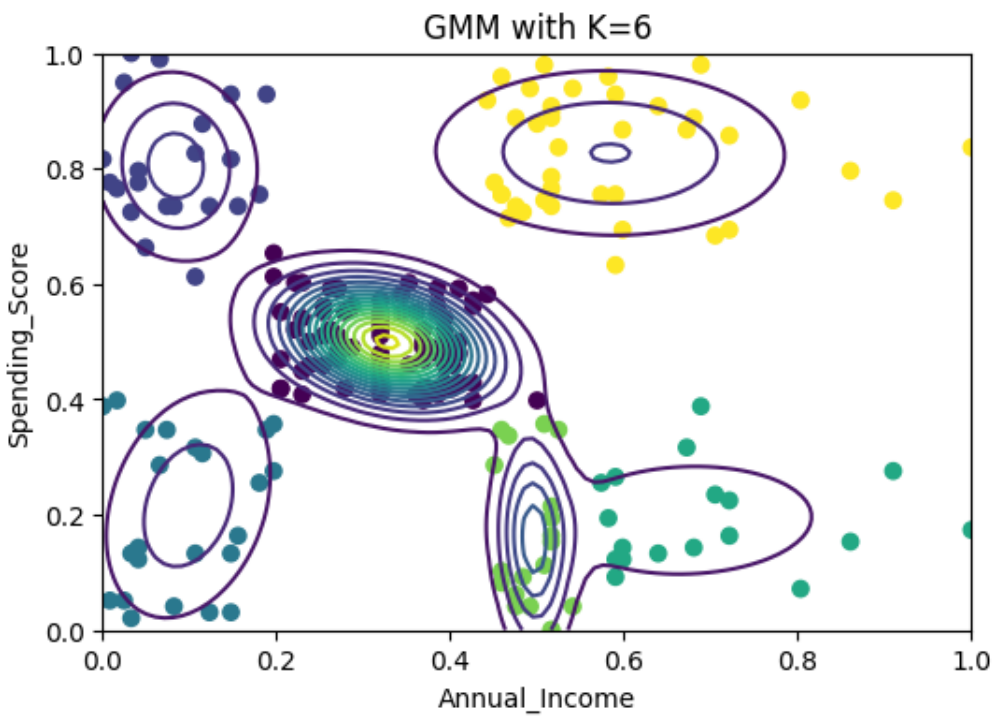


The figure above shows that K=6 is a proper choice for this dataset.

2 a.2

[4]:
```
weight, mean, cov, _ = GMM(x, proper_k)

X, Y = np.meshgrid(np.linspace(0, 1, 100), np.linspace(0, 1, 100))
XX = np.array([X.ravel(), Y.ravel()]).T
Z = np.zeros(XX.shape[0])
for k in range(proper_k):
    Z += weight[k] * multivariate_normal.pdf(XX, mean=mean[k], cov=cov[k])
Z = Z.reshape(X.shape)

r = np.zeros((x.shape[0], proper_k))
for k in range(proper_k):
    r[:, k] = weight[k] * multivariate_normal.pdf(x, mean=mean[k], cov=cov[k])
r /= r.sum(axis=1, keepdims=True)
y = np.argmax(r, axis=1)
```

```
# plot
plt.figure(figsize=(6, 4))
plt.scatter(x[:, 0], x[:, 1], c=y)
plt.contour(X, Y, Z, levels=20)
plt.title('GMM with K=6')
plt.xlabel('Annual_Income')
plt.ylabel('Spending_Score')
plt.show()
```

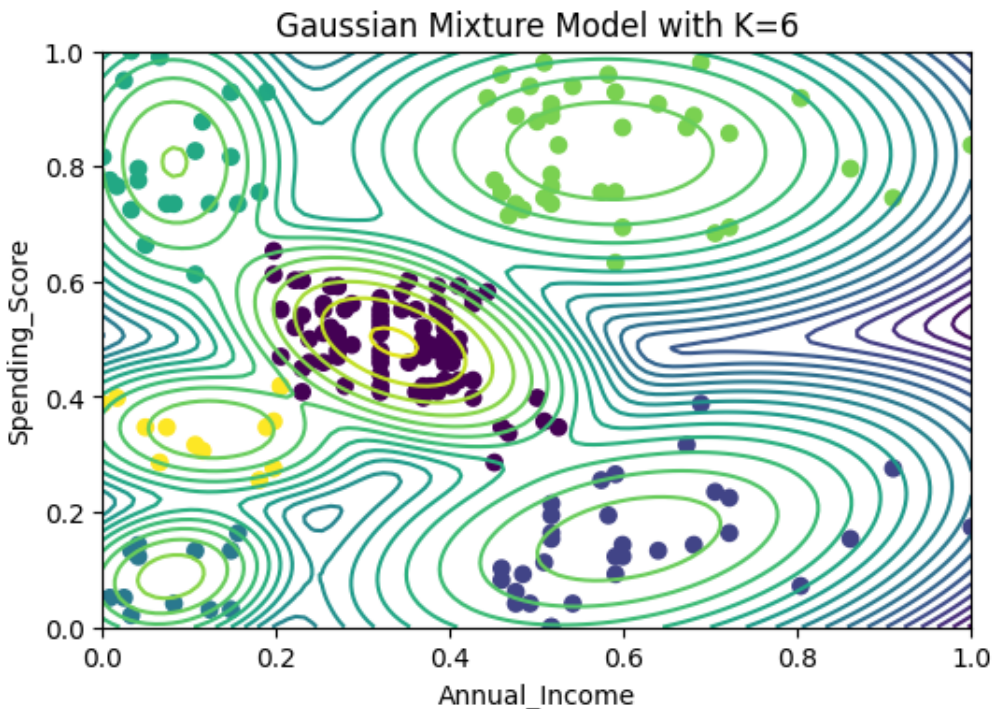


3 a.3

[5]:
```
np.random.seed(seed)
gmm = GaussianMixture(n_components=proper_k, covariance_type='full',␣
↪init_params='kmeans', random_state=seed).fit(x)
y = gmm.predict(x)
# print(y.shape)

X, Y = np.meshgrid(np.linspace(0, 1, 100), np.linspace(0, 1, 100))
XX = np.array([X.ravel(), Y.ravel()]).T
Z = np.zeros(XX.shape[0])
Z = gmm.score_samples(XX)
Z = Z.reshape(X.shape)
```

```
# plot
plt.figure(figsize=(6, 4))
plt.scatter(x[:, 0], x[:, 1], c=y)
plt.contour(X, Y, Z, levels=20)
plt.title("Gaussian Mixture Model with K=6")
plt.xlabel("Annual_Income")
plt.ylabel("Spending_Score")
```

[5]: `Text(0, 0.5, 'Spending_Score')`



The result using GaussianMixture is similar to the result in a.2.

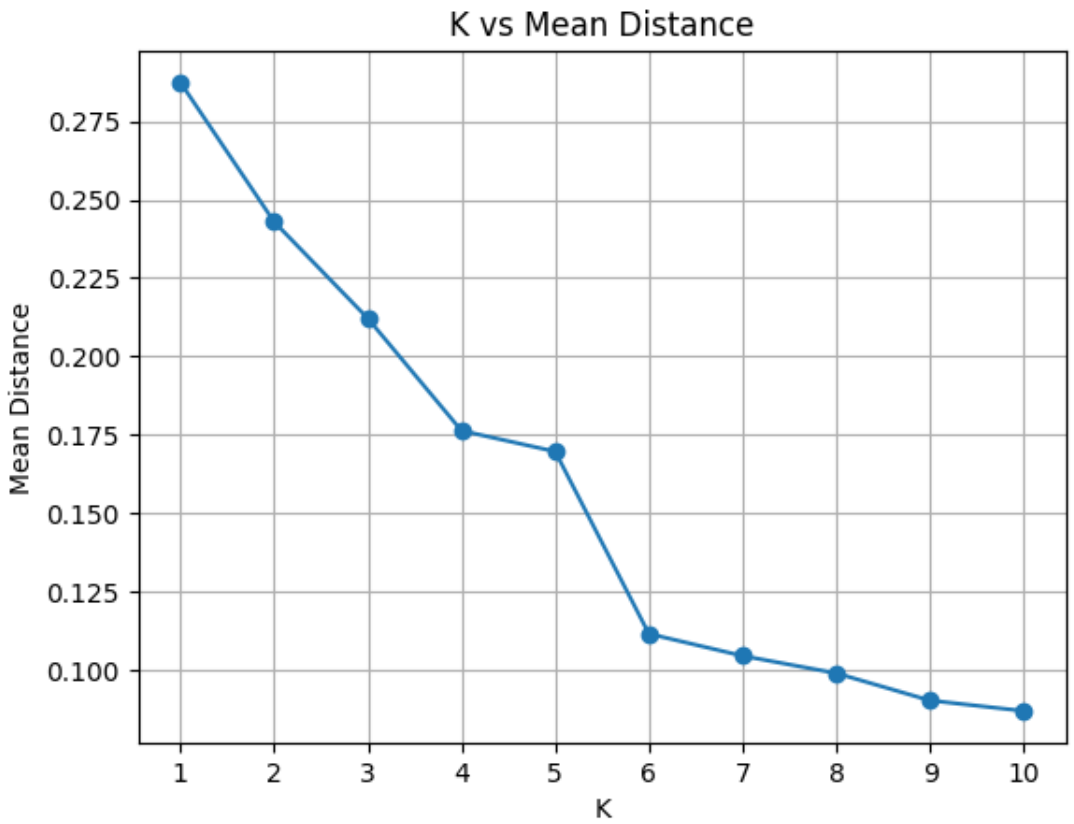
However, there are slight differences in the spread of the contours due to different initialization and optimization process.

4 b.1

[6]:
```
def KMeans(data, K, max_iter=100, tol=1e-4):
    np.random.seed(42)
    N, d = data.shape
```

```
    center = data[np.random.choice(N, K, replace=False)]
    center_old = np.zeros(center.shape)
    label = np.zeros(N, dtype=int)

    for t in range(max_iter):
        for i in range(N):
            distances = np.linalg.norm(data[i] - center, axis=1)
            label[i] = np.argmin(distances)

        center_old = center.copy()
        for k in range(K):
            center[k] = np.mean(data[label == k], axis=0)

        if np.linalg.norm(center - center_old) < tol:
            break

    return label, center
```

[7]:
```
mean_distance = []
for i in range(1, 11):
    label, center = KMeans(x, i)
    total = 0
    for k in range(i):
        total += np.sum(np.linalg.norm(x[label == k] - center[k], axis=1))
    mean_distance.append(total / x.shape[0])
# print(mean_distance)

plt.plot(range(1, 11), mean_distance, marker='o')
plt.xticks(range(1, 11))
plt.title('K vs Mean Distance')
plt.xlabel('K')
plt.ylabel('Mean Distance')
plt.grid(True)
plt.show()

proper_k = 6
```

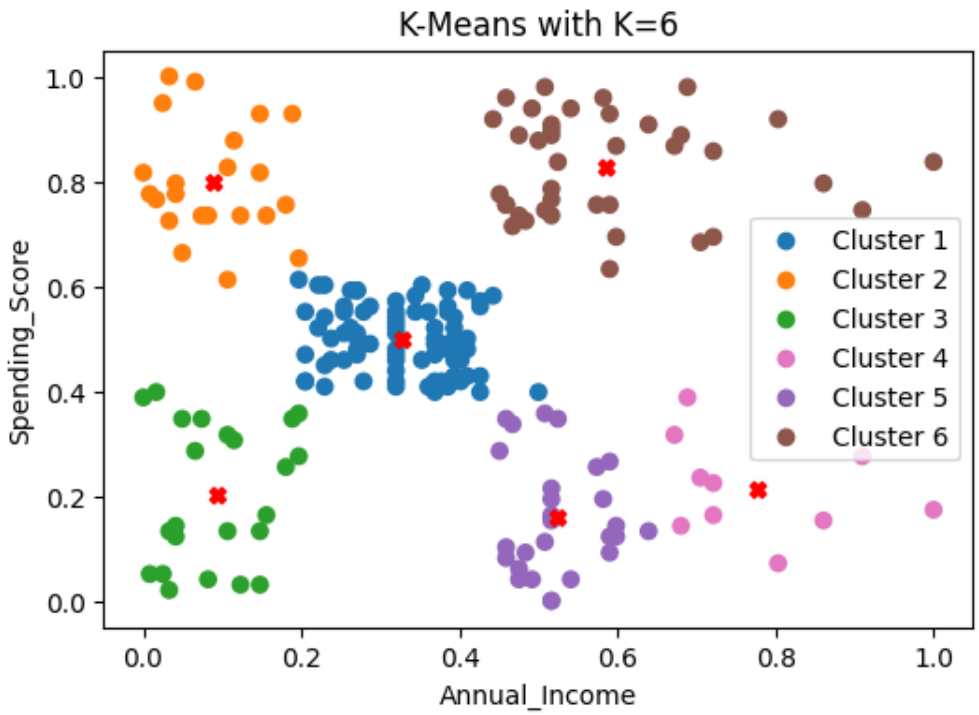


The figure above shows that K=6 is a proper choice for this dataset for K-Means.

5 b.2

[8]:
```
label, center = KMeans(x, proper_k)
# print(center)
plt.figure(figsize=(6, 4))
colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#e377c2', '#9467bd', '#8c564b']
for k in range(proper_k):
    plt.scatter(x[label == k][:, 0], x[label == k][:, 1], c=[colors[k] for _ in␣
 ↪range(x[label == k].shape[0])], label=f'Cluster {k+1}')
# plt.scatter(x[:, 0], x[:, 1], c=label)
plt.scatter(center[:, 0], center[:, 1], marker='X', c='r')
plt.title("K-Means with K=6")
plt.legend()
plt.xlabel("Annual_Income")
plt.ylabel("Spending_Score")
```

[8]:
```
Text(0, 0.5, 'Spending_Score')
```



Cluster 1: Customers in this cluster have low to moderate annual income and moderate spending scores.

Cluster 2: Customers in this cluster have low annual income but high spending scores.

Cluster 3: Customers in this cluster have low annual income and low spending scores.

Cluster 4: Customers in this cluster have high annual income but low spending scores.

Cluster 5: Customers in this cluster have moderate annual income but low spending scores.

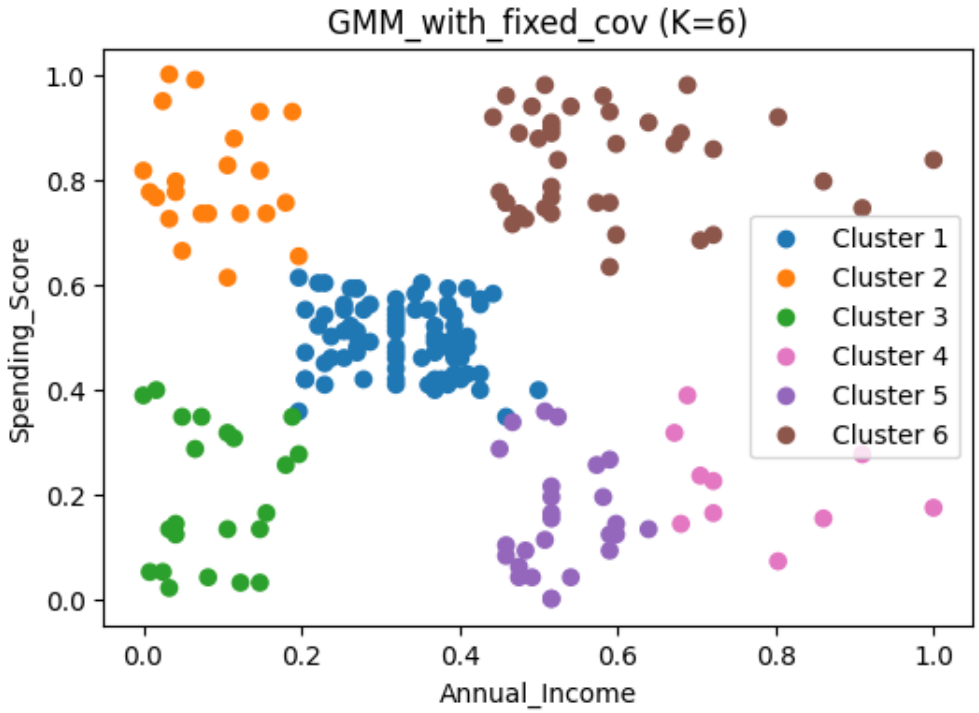
Cluster 6: Customers in this cluster have moderate to high annual income and high spending scores.

6 c

[9]:
```
def GMM_with_fixed_cov(data, K, max_iter=100, tol=1e-4):
    np.random.seed(seed)
    N, d = data.shape
    weight = np.full(K, 1/K)
    mean = data[np.random.choice(N, K, replace=False)]
```

```
    cov = np.array([1e-3 * np.eye(d) for _ in range(K)])
    ll = 0

    for t in range(max_iter):
        # Expectation step
        r = np.zeros((N, K))
        for k in range(K):
            r[:, k] = weight[k] * multivariate_normal.pdf(data, mean=mean[k],␣
 ↪cov=cov[k])
        r /= r.sum(axis=1, keepdims=True)

        # Maximization step
        for k in range(K):
            sum_rk = r[:, k].sum()
            mean[k] = (data * r[:, k, np.newaxis]).sum(axis=0) / sum_rk
            weight[k] = sum_rk / N

        # Compute log-likelihood
        last_11 = ll
        ll = np.sum(np.log(np.sum([w * multivariate_normal.pdf(data,␣
 ↪mean=mean[k], cov=cov[k]) for k, w in enumerate(weight)], axis=0)))

        if np.abs(ll - last_11) < tol:
            break

    return weight, mean, cov

weight, mean, cov = GMM_with_fixed_cov(x, proper_k)

r = np.zeros((x.shape[0], proper_k))
for k in range(proper_k):
    r[:, k] = weight[k] * multivariate_normal.pdf(x, mean=mean[k], cov=cov[k])
r /= r.sum(axis=1, keepdims=True)
y = np.argmax(r, axis=1)

# plot
plt.figure(figsize=(6, 4))
plt.figure(figsize=(6, 4))
colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#e377c2', '#9467bd', '#8c564b']
for k in range(proper_k):
    plt.scatter(x[y == k][:, 0], x[y == k][:, 1], c=[colors[k] for _ in␣
 ↪range(x[y == k].shape[0])], label=f'Cluster {k+1}')
plt.title('GMM_with_fixed_cov (K=6)')
plt.legend()
plt.xlabel('Annual_Income')
plt.ylabel('Spending_Score')
plt.show()
```

```
<Figure size 600x400 with 0 Axes>
```



When the covariate matrix is fixed as a relatively small value, the result of GMM (with same K) is similar to the result of K-Means Algorithm.

This is because that when covariate matrix is small, the responsibility of cluster k for x_i will be mainly influenced by the distance between x_i and the cluster's mean, which makes it behave like K-Means.