تمرین شماره 5

علیرضا حسینی

شماره دانشجویی : ۸۱۰۱۰۱۱۴۲

یادگیری ماشین

دکتر ابوالقاسمی و دکتر توسلی پور

بهار 1402

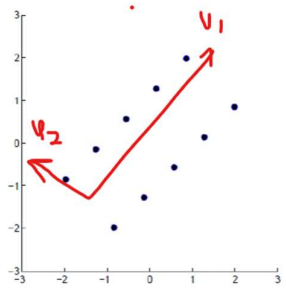
فهرست مطالب

فهرست اشکال

۱-۱- سوال 1 : PCA برای داده 2 بعدی + و –

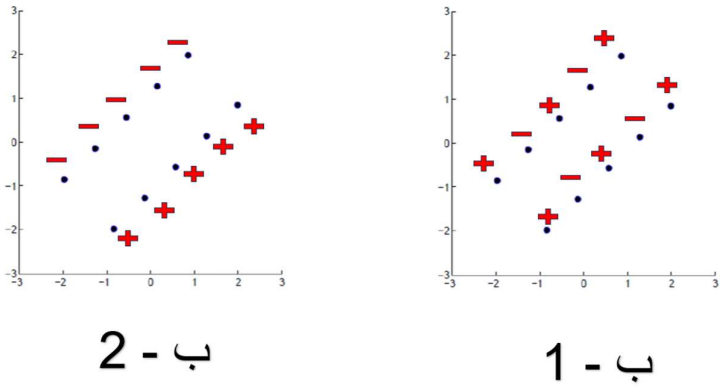
الف) برچسب های داده ها، که نشان دهنده کلاس یا دسته ای است که هر نقطه داده به آن تعلق دارد، در طول فرآیند PCA در نظر گرفته نمی شود. PCA فقط بر روی مقادیر عددی ویژگی ها عمل می کند. بر روی الگوها و روابط درون خود ویژگی ها تمرکز می کند، بدون توجه به برچسب ها.

محور هایی که انتخاب میکند در شکل زیر میباشد.



شکل (۱-۱) محور های u1 و u2 پس از اعمال pCA

ب) جواب در تصویر زیر آمده است.



شکل (۲-۱) پاسخ بخش 2 سوال 1

۲-۱- سوال 2 : SW در LDA

$$J = \frac{1}{n_1 n_2} \sum_{y_i} \sum_{y_j} (y_i - y_j)^2 = \frac{1}{n_1 n_2} \sum_{y_i} \sum_{y_j} (y_i^2 + y_j^2 - 2 y_i y_j)$$

$$= \frac{1}{n_1 n_2} \left[\sum_{y_i} \sum_{y_j} y_i^2 + \sum_{y_i} \sum_{y_j} y_j^2 - 2 \sum_{y_i} \sum_{y_j} y_i y_j \right]$$

$\sum_{y_j} = n_2$, $\sum_{y_i} = n_1$, $\sum_{y_i} \sum_{y_j} y_i y_j = m_1 m_2$

$$= \left[\frac{\sum y_i^2}{n_1} + \frac{\sum y_j^2}{n_2} - 2 m_1 m_2 \right]$$

$$= m_1^2 + m_2^2 - 2m_1m_2 + \frac{\sum y_i^2}{n_1} + m_1^2 - 2m_1^2 + \frac{\sum y_j^2}{n_2} + m_2^2 - 2m_2^2$$

خودش $2m_1^2$ و $2m_2^2$ اضافه کردیم.

$$= (m_1 - m_2)^2 + \frac{\sum y_i^2}{n_1} + m_1^2 + m_2^2 + \frac{\sum y_j^2}{n_2} + \frac{2m_1 \sum y_i m_1}{n_1} - \frac{2m_1 \sum y_i m}{m_1}$$

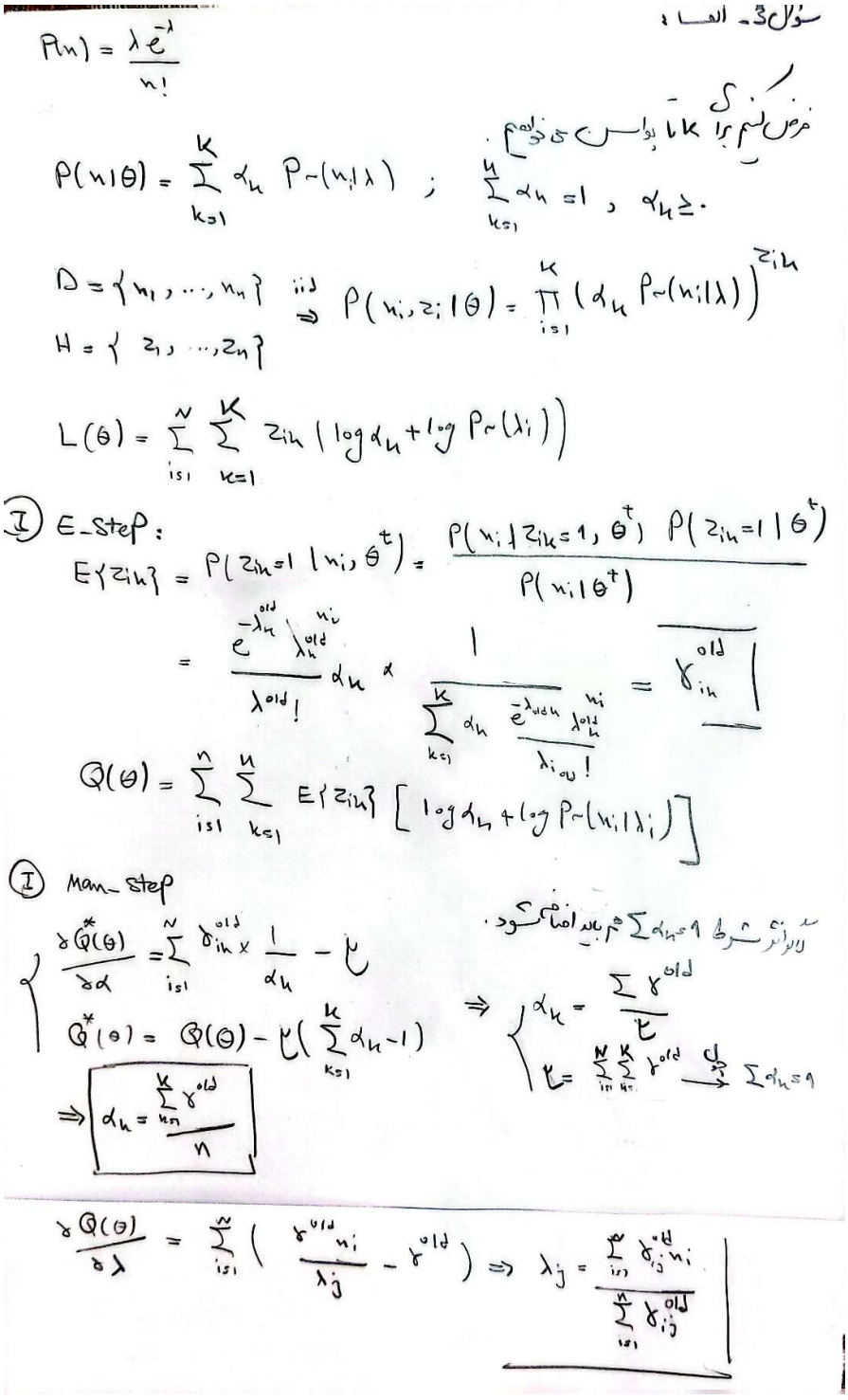
$$= (m_1 - m_2)^2 + \frac{\sum_{y_i} (y_i^2 + m_1^2 - 2 y_i m_1)}{n_1} + \frac{\sum_{y_j} (y_j^2 + m_2^2 - 2 y_j m_2)}{n_2}$$

$$= (m_1 - m_2)^2 + \frac{1}{n_1} \sum_{y_i} (y_i - m_1)^2 + \frac{1}{n_2} \sum_{y_i} (y_i - m_2)^2$$

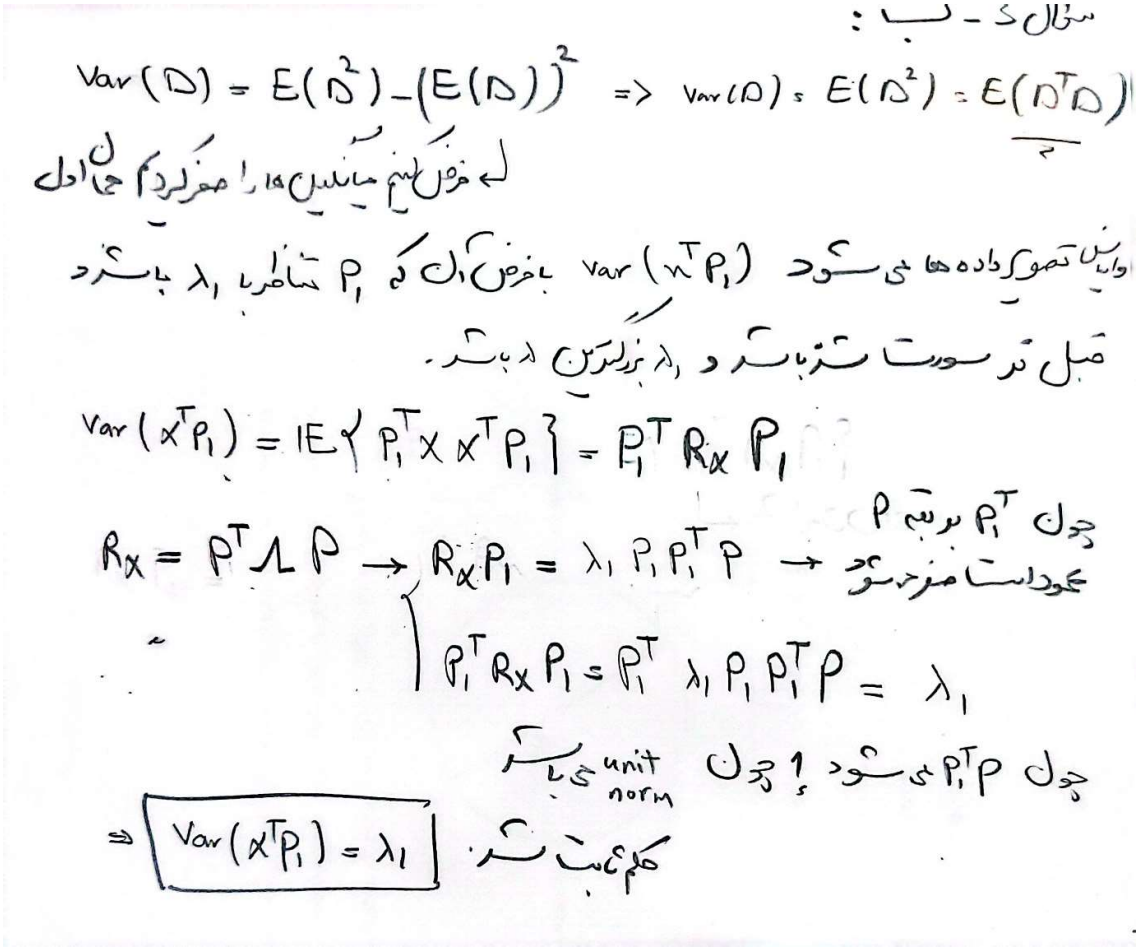
$$= (m_1 - m_2)^2 + \frac{1}{n_1} s_1^2 + \frac{1}{n_2} s_2^2$$

شکل (۳-۱) جواب سوال 2

۳-۱- سوال3 : سوالاتی در مورد EM و PCA



شکل (۴-۱) جواب سوال 3 – الف



شکل (۵-۱) جواب سوال 3 – ب

۴-۱- سوال 4 : سوالاتی در مورد EM

الف) مدل الف مناسب تر است چراکه با ماتریس کواریانس را به خوبی دخیل کرده و با توجه به شکل به خوبی توانسته clustering را انجام دهد.

شکل دومی 2 تا کلاستر با هم همپوشانی دارند به علاوه در شکل دوم فضای خالی با احتمال بالایی برای هر کدام از کلاستر ها دارد.

ب) الف خروجی گام بعدی میباشد .

شکل ب اصلا با روند الگوریتم EM مطابقت ندارد ، با فرض مقدار دهی گوسی اولیه داده شده در E-Step

و MAX گوسی ها یکی به سمت راست میرود یکی به سمت چپ ، به همین ترتیب مرکز شامل 2 مرکز در راست چپ ها میشود بنابراین تحت هیچ شرایطی شکل ب رخ نمیدهد و شکل الف میتواند گام بعدی باشد.

۵-۱- سوال 5 : PCA و LDA شبیه سازی

1-5-1- بخش 1 : PCA

ابتدا داده ها و لیبل های آن را میخوانیم و در لیست های test و ترین ذخیره میکنیم.

```
train_folder = 'train'
test_folder = 'test'

# Create empty lists to store the images and labels
train_images = []
train_labels = []
test_images = []
test_labels = []
```

```
# Load the train images and labels
for filename in os.listdir(train_folder):
    if filename.endswith('.tiff'):
        label = filename.split('.')[1]  # Extract the category label from the filename
        if label in ['disgust', 'happy', 'fear', 'angry', 'surprise', 'sad']:
            image = cv2.imread(os.path.join(train_folder, filename), cv2.IMREAD_GRAYSCALE)
            train_images.append(image)
            train_labels.append(label)

# Load the test images and labels
for filename in os.listdir(test_folder):
    if filename.endswith('.tiff'):
        label = filename.split('.')[1]  # Extract the category label from the filename
        if label in ['disgust', 'happy', 'fear', 'angry', 'surprise', 'sad']:
            image = cv2.imread(os.path.join(test_folder, filename), cv2.IMREAD_GRAYSCALE)
            test_images.append(image)
            test_labels.append(label)
```

شکل (۶-۱) لود کردن داده های سوال 5 بخش 1

در ادامه داده های عکس را به numpy تبدیل کرده و با توجه به اینکه برای PCA که روش کاهش بعد

میباشد لیبل اهمیتی ندارد تمامی داده ها را concat میکنیم.

```
# Convert the lists of images and labels into numpy arrays
train_images = np.array(train_images)
train_labels = np.array(train_labels)
test_images = np.array(test_images)
test_labels = np.array(test_labels)

# Combine train and test images for PCA
combined_images = np.concatenate((train_images, test_images), axis=0)

# Flatten the images
flat_images = combined_images.reshape(combined_images.shape[0], -1)
```

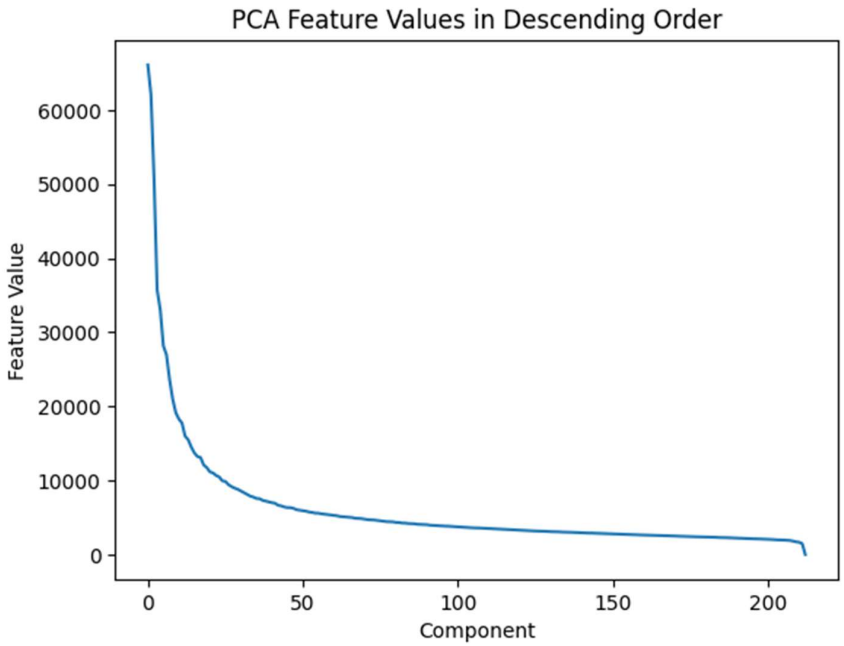
شکل (۷-۱) پیش پردازش داده های سوال 5 بخش 1

در ادامه مقادیر ویژه از PCA را پس از اعمال PCA به صورت کاهشی رسم میکنیم.

به جهت استفاده از BIC مقادیر variance_ratio هم در لیستی ذخیره میکنیم.

```
# Perform PCA on the flattened images
pca = PCA()
pca.fit(flat_images)

# Compute the explained variance ratios and cumulative explained variance
explained_variance_ratios = pca.explained_variance_ratio_
cumulative_variance = np.cumsum(explained_variance_ratios)

# Plot the PCA feature values in descending order
pca_feature_values = np.sort(pca.singular_values_)[::-1]
plt.plot(pca_feature_values)
plt.xlabel('Component')
plt.ylabel('Feature Value')
plt.title('PCA Feature Values in Descending Order')
plt.show()
```

شکل (۸-۱) کد پایتون رسم مقادیر ویژ] PCA به صورت کاهشی



شکل (۹-۱) مقادیر ویژه PCA به صورت کاهشی

جهت به دست آوردن K بهینه از BIC استفاده میکنیم.

BIC یک معیار آماری است که بین پیچیدگی مدل و احتمال داده ها تعادل برقرار می کند و آن را به ابزاری موثر برای انتخاب مدل تبدیل می کند. به عبارت دیگر BIC معیاری است که برای ارزیابی تناسب مدل‌های آماری در حالی که پیچیدگی مدل را جریمه می‌کند، استفاده می‌شود. لاگ احتمال داده ها را با یک جریمه بر اساس تعداد پارامترهای مدل ترکیب می کند. با به حداقل رساندن امتیاز BIC، می‌توانیم مناسب‌ترین مدلی را شناسایی کنیم که در عین سادگی، احتمال را به حداکثر می‌رساند.

مراحل تعیین بهترین تعداد (k) با استفاده از BIC:

آ. ترکیب مجموعه داده: برای انتخاب مؤلفه مؤثر، مجموعه داده‌های آموزشی و آزمایشی را ترکیب می‌کنیم تا نمایش جامعی از داده‌ها به دست آوریم.

ب تکرار بر روی k: ما در محدوده ای از مقادیر مولفه بالقوه، از 1 تا حداکثر مقدار از پیش تعریف شده،

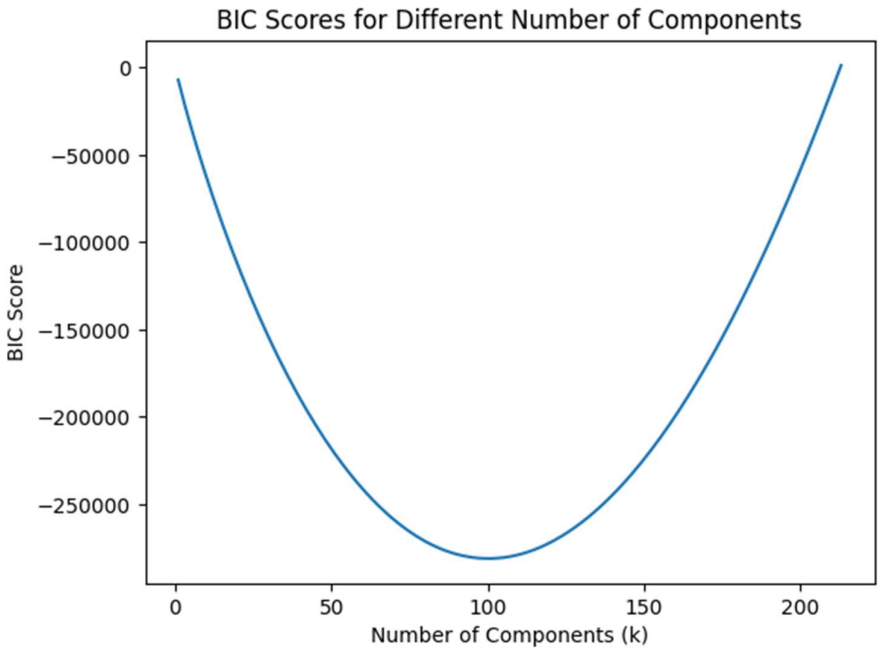
تکرار می کنیم و PCA را برای هر k انجام می دهیم.

ج محاسبه امتیاز BIC: برای هر مقدار k، امتیاز BIC را با استفاده از log-likelihood و یک عبارت جریمه که تعداد مؤلفه ها را محاسبه می کند، محاسبه می کنیم.

د مقدار k با حداقل امتیاز BIC بهترین تعداد مؤلفه ها در نظر گرفته می شود که تعادلی بین تناسب مدل و پیچیدگی ایجاد می کند.

```
# Determine the best number of components (k) using BIC
n_samples = combined_images.shape[0]  # Number of samples in the combined images
bic_scores = []

for k in range(1, pca.n_components_ + 1):
    pca = PCA(n_components=k)
    pca.fit(combined_images)
    log_likelihood = np.sum(np.log(pca.explained_variance_) * (n_samples - k))
    bic_score = -2 * log_likelihood + k * np.log(n_samples)
    bic_scores.append(bic_score)

# Plot the BIC scores
plt.plot(range(1, pca.n_components_ + 1), bic_scores)
plt.xlabel('Number of Components (k)')
plt.ylabel('BIC Score')
plt.title('BIC Scores for Different Number of Components')
plt.show()

# Find the best number of components (k) with the minimum BIC score
best_k = np.argmin(bic_scores) + 1
print("Best number of components (k):", best_k)
```

شکل (۱۰-۱) پیاده سازی معیار BIC



شکل (۱۱-۱) امتیاز BIC به ازای component های مختلف

در نهایت نیز 4 مقدار ویژه اول و eigen face نهایی را به صورت زیر بر اساس همین k بدست آمده که 100 میباشد به دست میآوریم.

```
# Display the first four eigenvalues
print("First four eigenvalues:")
print(pca_feature_values[:4])

# Compute and display the final four eigenfaces
final_four_eigenfaces = pca.components_[-4:]
image_shape = train_images[0].shape
fig, axes = plt.subplots(1, 4, figsize=(12, 3))
for i, eigenface in enumerate(final_four_eigenfaces):
    eigenface = eigenface.reshape(image_shape)
    axes[i].imshow(eigenface, cmap='gray')
    axes[i].axis('off')
    axes[i].set_title(f"Eigenface {i+1}")
plt.tight_layout()
plt.show()
```

شکل (۱۲-۱) رسم eigenface های نهایی

شکل (۱۳-۱) 4 مقدار ویژه اول و 4 eigenface نهایی

نتیجه گیری نهایی :

در تجسم چهار مقدار ویژه آخر، جایی که فقط لبه‌ها قابل مشاهده هستند، یک ویژگی مشترک فشرده‌سازی تصویر مبتنی بر PCA را مشاهده می‌کنیم. مقادیر ویژه نشان دهنده مقدار واریانس توضیح داده شده توسط هر جزء اصلی (بردار ویژه) است. با حرکت به سمت مقادیر ویژه نهایی، مقدار واریانس ثبت شده به طور قابل توجهی کاهش می یابد. این بدان معنی است که بردارهای ویژه مربوطه الگوهای کمتر برجسته را در مجموعه داده تصویر می گیرند.

لبه هایی که پس از اعمال PCA در تصاویر فشرده ظاهر می شوند را می توان به این واقعیت نسبت داد که لبه ها اغلب حاوی اطلاعات با فرکانس بالا در تصاویر هستند. لبه ها نشان دهنده تغییرات ناگهانی در شدت پیکسل هستند و برای درک بصری ما از اشیا و اشکال بسیار مهم هستند. با حفظ بردارهای ویژه متناظر با مقادیر ویژه نهایی، که برجسته ترین الگوها را در مجموعه داده ثبت می کنند، ما به طور موثر لبه ها را حفظ می کنیم و در عین حال تغییرات کمتر مهم را کنار می گذاریم.

فشرده‌سازی تصاویر با استفاده از PCA با نمایش داده‌های تصویر اصلی روی یک فضای فرعی با ابعاد پایین‌تر که توسط مؤلفه‌های اصلی انتخاب شده در بر می‌گیرد، انجام می‌شود. این فرآیند اجزای با مقادیر ویژه کمتر را حذف می کند، که کمتر به اطلاعات کلی تصویر کمک می کند. با دور انداختن این اجزا، ابعاد داده های تصویر

را کاهش می دهیم.

تجزیه و تحلیل تصاویر فشرده نشان می دهد که فشرده سازی مبتنی بر PCA ویژگی های اساسی تصاویر، به ویژه لبه ها را حفظ می کند، در حالی که نیازهای ذخیره سازی را کاهش می دهد. این تکنیک فشرده‌سازی از افزونگی در داده‌های تصویر استفاده می‌کند و بر ثبت مهم‌ترین الگوها تمرکز می‌کند. با این حال، توجه به این نکته مهم است که فشرده سازی مبتنی بر PCA یک روش فشرده سازی با اتلاف است، به این معنی که مقداری از دست دادن اطلاعات در طول فرآیند فشرده سازی رخ می دهد.

انتخاب تعداد اجزاء (k) برای فشرده سازی در متعادل کردن نسبت تراکم و مقدار اطلاعات حفظ شده بسیار مهم است. معیار BIC به ما کمک می کند تا با در نظر گرفتن پیچیدگی مدل و احتمال ورود به سیستم، مقدار بهینه k را تعیین کنیم. با انتخاب بهترین k، هدف ما دستیابی به یک مبادله بهینه بین فشرده سازی و حفظ اطلاعات است.

به طور خلاصه، تجسم تصاویر فشرده شده با PCA حفظ لبه ها را برجسته می کند، که برای درک بصری بسیار مهم هستند. تجزیه و تحلیل فرآیند فشرده سازی نشان می دهد که فشرده سازی مبتنی بر PCA به طور موثری ابعاد تصاویر را کاهش می دهد در حالی که ویژگی های اساسی را حفظ می کند. انتخاب تعداد اجزاء نقش بسزایی در تعیین نسبت فشرده سازی و سطح از دست دادن اطلاعات دارد.

2-5-1- بخش دوم

ابتدا داده های داده شده را خوانده و داده ها را numpy array تبدیل میکنیم.

```python
# Load the data
train_data = pd.read_csv("trainData.csv", header=None)
train_labels = pd.read_csv("trainLabels.csv", header=None)
test_data = pd.read_csv("testData.csv", header=None)
test_labels = pd.read_csv("testLabels.csv", header=None)
```

```python
# Convert the data and labels to NumPy arrays
train_data = train_data.values
train_labels = train_labels.values.flatten()
test_data = test_data.values
test_labels = test_labels.values.flatten()
```

```python
# Get the unique class labels
unique_labels = np.unique(train_labels)
```

شکل (۱۴-۱) خواندن و پردازش داده های بخش 2 سوال 5

در ادامه به محاسبه ماتریس های Sw و Sb پرداخته میشود.

```python
# Calculate the overall mean
overall_mean = np.mean(train_data, axis=0)

# Calculate Sw and Sb
Sw = np.zeros((train_data.shape[1], train_data.shape[1]))
Sb = np.zeros((train_data.shape[1], train_data.shape[1]))

for label in unique_labels:
    class_data = train_data[train_labels == label]
    class_cov = np.cov(class_data.T)
    class_scatter = class_data.shape[0] * class_cov
    Sw += class_scatter

    n_samples = np.sum(train_labels == label)
    mean_diff = np.mean(class_data, axis=0) - overall_mean
    class_scatter = n_samples * np.outer(mean_diff, mean_diff)
    Sb += class_scatter
```

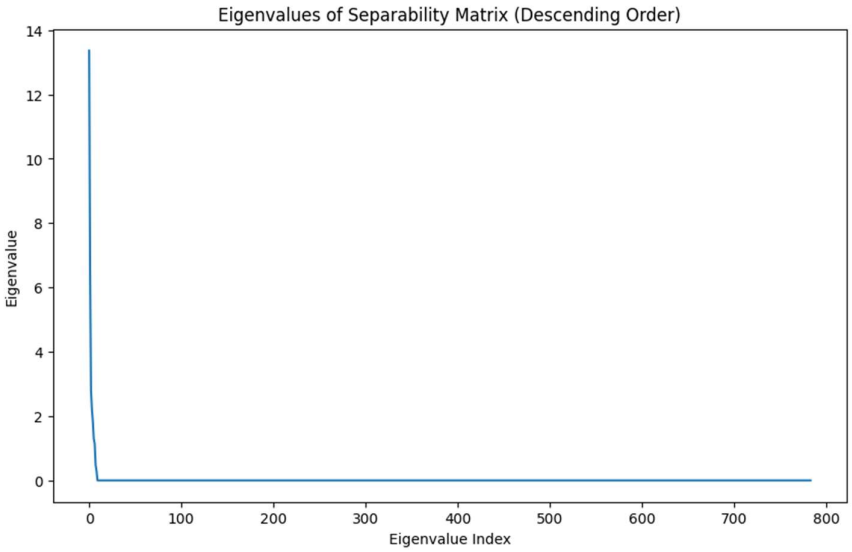
شکل (۱۵-۱) محاسبه Sw و Sb

پس از مشخص شدن ماتریس ها به کمک generalized eigen value Decomposition مقادیر ویژه را حساب کرده و ان ها را سورت میکنیم.

```
# Compute the eigenvalues of the separability matrix
eigenvalues, _ = np.linalg.eig(np.linalg.inv(Sw) @ Sb)

# Sort the eigenvalues in descending order
sorted_eigenvalues = np.sort(eigenvalues)[::-1]
```

شکل (۱۶-۱) محاسبه مقادیر ویژه برایLDA

رسم مقادیر ویژه به صورت نزولی به صورت زیر میشود.

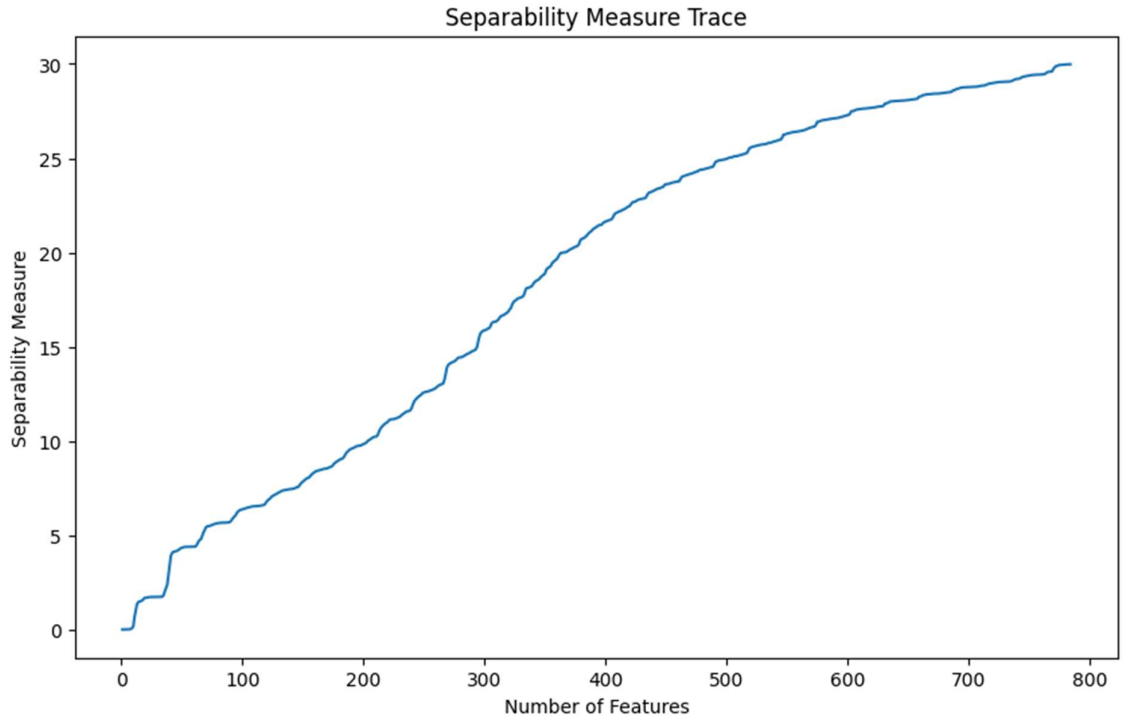


شکل (۱۷-۱) رسم مقادیر ویژه ماتریس جدا کننده به صورت نزولی

در نهایت نیز ماتریس جدا کننده را بر حسب تعداد فیچر ها رسم میکنیم.

```python
# Calculate the separability measure trace for different numbers of features
num_features = range(1, train_data.shape[1] + 1)
separability_measures = []

for n in num_features:
    selected_features = train_data[:, :n]
    Sw_n = Sw[:n, :n]
    Sb_n = Sb[:n, :n]

    separability_measure = np.trace(np.linalg.inv(Sw_n) @ Sb_n)
    separability_measures.append(separability_measure)

# Plot the separability measure trace
plt.figure(figsize=(10, 6))
plt.plot(num_features, separability_measures)
plt.xlabel('Number of Features')
plt.ylabel('Separability Measure')
plt.title('Separability Measure Trace')
plt.show()
```

شکل (۱۸-۱) رسم ماتریس جداکننده بر حسب تعداد فیچر ها



شکل (۱۹-۱) Trace ماتریس جدا کننده در حسب تعداد فیچر ها

نمودار فوق قدرت تمایز ویژگی های انتخاب شده را کمی نشان می دهد. اندازه گیری تفکیک پذیری به عنوان اثری از حاصل ضرب ماتریس معکوس ماتریس پراکندگی درون کلاسی (Sw) و ماتریس پراکندگی بین کلاسی (Sb) محاسبه می شود.

نمودار به طور کلی نشان می دهد که چگونه معیار تفکیک پذیری با افزایش تعداد ویژگی ها تغییر می کند. در اینجا این مورد مشاهده میشود که با افزایش تعداد ویژگی ها، معیار تفکیک پذیری نیز افزایش میابد(همانطور که انتظار میرفت). این نشان می‌دهد که ویژگی‌های انتخاب‌شده بیشتر به تبعیض بین طبقات کمک می‌کنند و در نتیجه تفکیک‌پذیری کلاس‌ها بهتر است.

۶-۱- سوال 6 : EM شبیه سازی

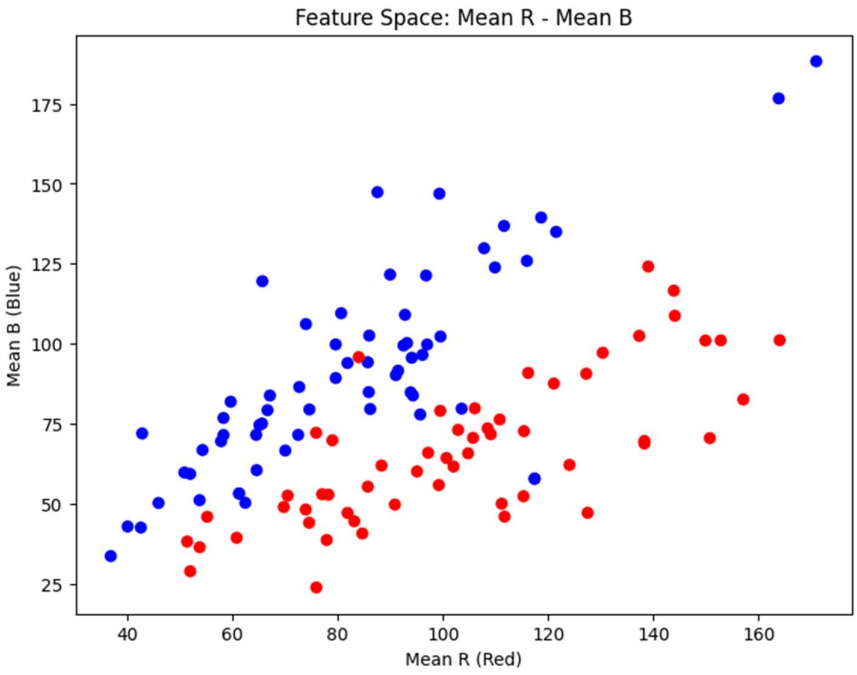
1-6-1- لود کردن و پیش پردازش داده ها

ابتدا داده ها را خوانده در هر عکس کانال R و B را پیدا کرده و مقدار میانگین value های هر کانال را برمیداریم به علاوه با توجه به اسم عکس لیبل آن را هم بر میداریم این کار جهت مقایسه انجام میشود.

```python
# Step 1: Load and Process the Data
data_directory = "./"

labels = []
features = []

for filename in os.listdir(data_directory):
    if filename.lower().startswith('c'):
        labels.append(0)  # Chelsea label
    elif filename.lower().startswith('m'):
        labels.append(1)  # Manchester label
    else:
        continue  # Skip files that don't match the required format

    image_path = os.path.join(data_directory, filename)
    image = plt.imread(image_path)
    r_mean = np.mean(image[:, :, 0])  # Mean of Red channel values
    b_mean = np.mean(image[:, :, 2])  # Mean of Blue channel values
    features.append([r_mean, b_mean])

labels = np.array(labels)
features = np.array(features)
```

شکل (۲۰-۱) لود و پیش پردازش داده های سوال 6

توزیع داده ها را ابتدا رسم میکنیم.



شکل (۲۱-۱) توزیع داده های 2 کلاس بر حسب فیچر انتخاب شده

```
plt.figure(figsize=(8, 6))
colors = ['blue' if label == 0 else 'red' for label in labels]
plt.scatter(features[:, 0], features[:, 1], c=colors)
plt.xlabel('Mean R (Red)')
plt.ylabel('Mean B (Blue)')
plt.title('Feature Space: Mean R - Mean B')
plt.show()
```

شکل (۲۲-۱) کد پایتون رسم توزیع داده های 2 کلاس در فضای فیچر ها

1-6-2- پیاده سازی GMM با K=2

حال به پیاده سازی GMM با K=2 میپردازیم .

```
# Implement the EM Algorithm for GMM
k = 2  # Number of components

gmm = GaussianMixture(n_components=k)
gmm.fit(features)

# Report the Obtained Parameters
means = gmm.means_
covariances = gmm.covariances_
weights = gmm.weights_
```

شکل (۲۳-۱) پیاده سازی GMM با K=2

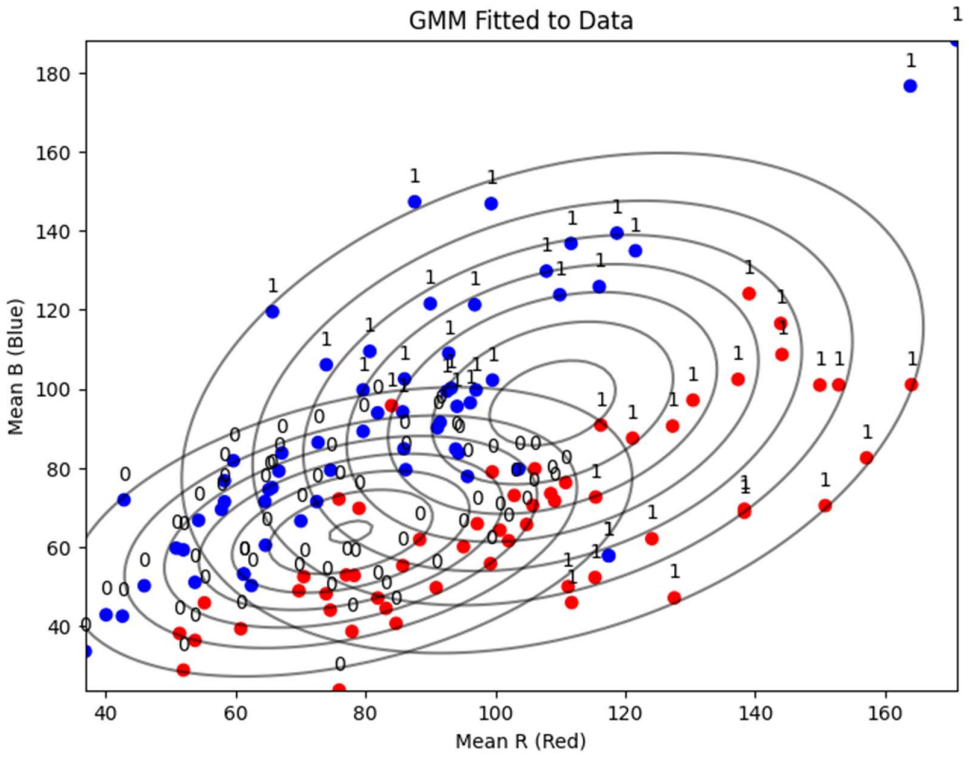
در ادامه پارامتر های به دست آمده و منحنی کانتور و گوسی های فیت شده و کلاس های cluster شده آمده است.

```
GMM Parameters:
Component 1:
Mean: [77.7119139  63.88297216]
Covariance: [[479.54335265 153.32680407]
 [153.32680407 341.27168401]]
Weight: 0.5333754854713472

Component 2:
Mean: [108.78462184  96.44794159]
Covariance: [[813.35066883 273.7337986 ]
 [273.7337986  997.6403329 ]]
Weight: 0.46662451452865283
```

شکل (۲۴-۱) پارامتر های بدست آمده از GMM

```python
# Plotting the Results
plt.figure(figsize=(8, 6))
plt.scatter(features[:, 0], features[:, 1], c=colors)

# Plot the contours of the GMM models fitted to each class
for i in range(k):
    mean = gmm.means_[i]
    cov = gmm.covariances_[i]
    plt.contour(X, Y, multivariate_normal(mean, cov).pdf(np.dstack((X, Y))), colors='black', alpha=0.5)

# Assign data points to clusters based on the highest probability
cluster_labels = gmm.predict(features)
for i, label in enumerate(cluster_labels):
    plt.annotate(str(label), (features[i, 0], features[i, 1]), textcoords="offset points", xytext=(0,10), ha='center')

plt.xlabel('Mean R (Red)')
plt.ylabel('Mean B (Blue)')
plt.title('GMM Fitted to Data')
plt.show()
```

شکل (۲۵-۱) کد پایتون رسم نتایج سوال 6



شکل (۲۶-۱) گوسی های فیت شده و cluster های دیتا ها

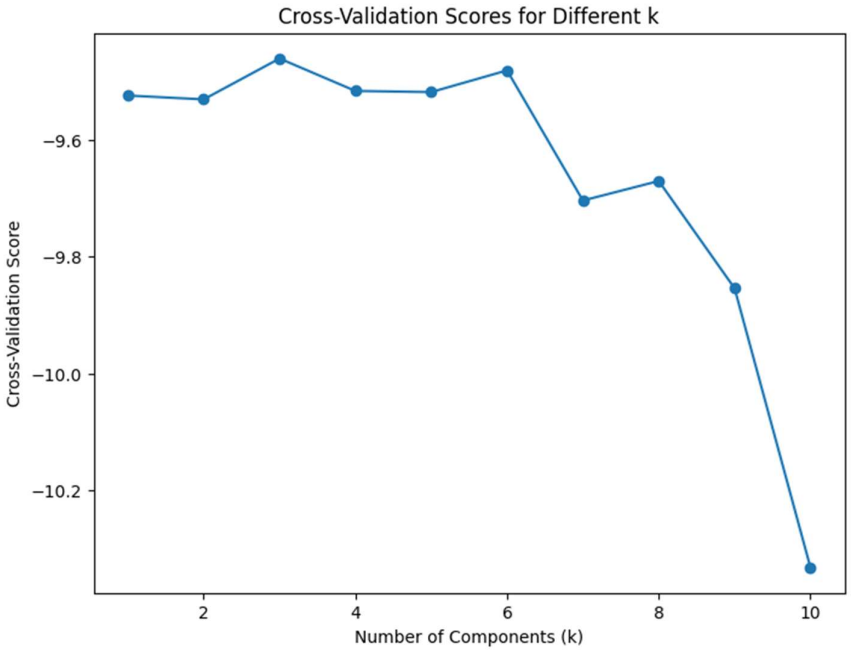
مشاهده میشود که کاملا متفاوت از لیبل ها cluster کرده

3-6-1- انتخاب بهترین K به کمک cross validation

در ادامه به کمک کد زیر بهترین K انتخاب میشود .

```
# Find Optimal k using Leave-One-Out Cross-Validation
max_k = 10  # Maximum value of k to consider
cv_scores = []  # To store the cross-validation scores for each value of k

for k in range(1, max_k+1):
    gmm = GaussianMixture(n_components=k)
    loo = LeaveOneOut()

    scores = []
    for train_index, test_index in loo.split(features):
        train_data, test_data = features[train_index], features[test_index]
        gmm.fit(train_data)
        score = gmm.score(test_data)
        scores.append(score)

    cv_scores.append(np.mean(scores))
```

شکل (۲۷-۱) انتخاب بهترین K به کمک الگوریتم Leave one out

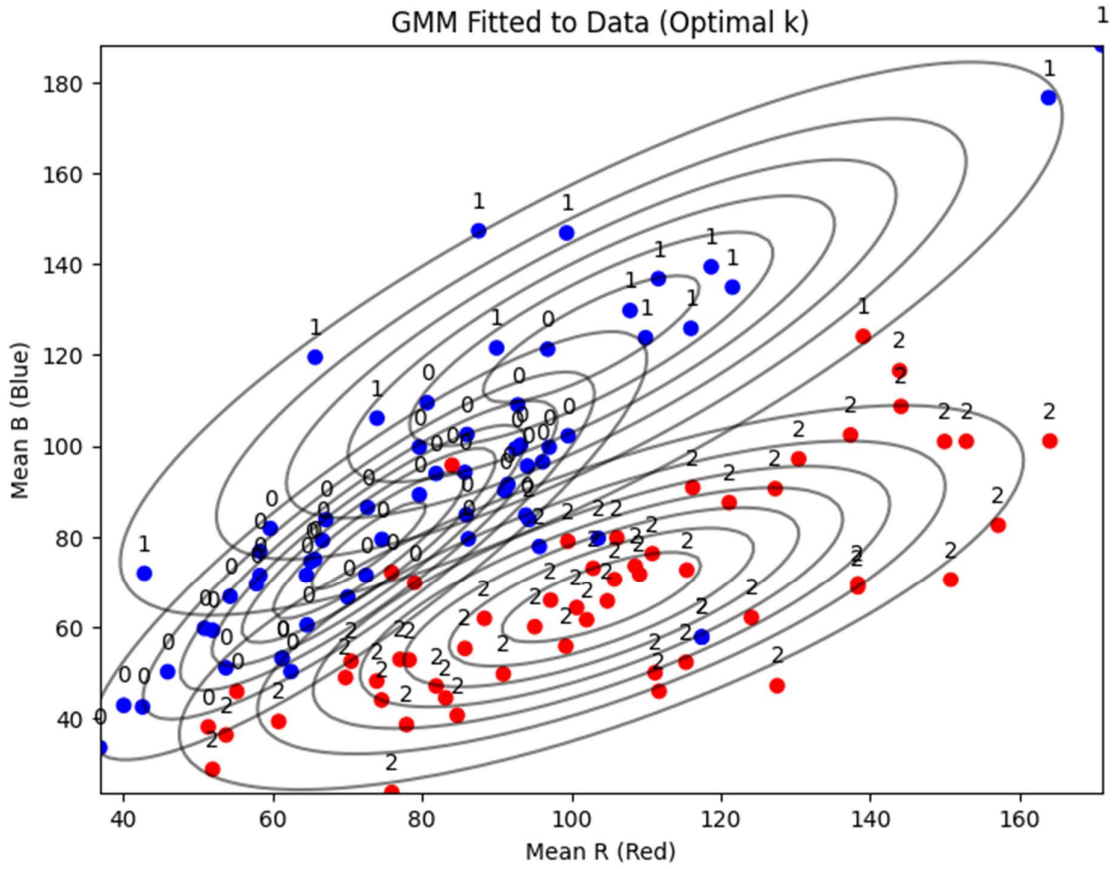


شکل (۲۸-۱) منحنی cross validation به ازای k

که در اینجا چون max در k=3 میباشد پس k=3 انتخاب میشود.

```python
# Fit GMM with Optimal k and Plot Results
gmm_optimal = GaussianMixture(n_components=optimal_k)
gmm_optimal.fit(features)

plt.figure(figsize=(8, 6))
plt.scatter(features[:, 0], features[:, 1], c=colors)

# Plot the contours of the GMM models fitted to each class
for i in range(optimal_k):
    mean = gmm_optimal.means_[i]
    cov = gmm_optimal.covariances_[i]
    plt.contour(X, Y, multivariate_normal(mean, cov).pdf(np.dstack((X, Y))), colors='black', alpha=0.5)

# Assign data points to clusters based on the highest probability
cluster_labels = gmm_optimal.predict(features)
for i, label in enumerate(cluster_labels):
    plt.annotate(str(label), (features[i, 0], features[i, 1]), textcoords="offset points", xytext=(0,10), ha='center')

plt.xlabel('Mean R (Red)')
plt.ylabel('Mean B (Blue)')
plt.title('GMM Fitted to Data (Optimal k)')
plt.show()
```

شکل (۲۹-۱) فیت کردن و رسم نتایج با k بهینه



شکل (۳۰-۱) فیت کردن با k بهینه

مشاهده میشود با این k کلاس که 3 تا cluster کرده کلاستر 1 و 0 و همان کلاس چلسی بوده و کلاستر 2 همان کلاس منچستر میشود.