به نام خدا

تمرین سوم بخش عملی
مهدی کافی ۹۹۲۱۰۷۵۳

۵.۱
آ)
در این بخش در ابتدا تمامی مقادیر "Yes" و "No" را به مقادیر ۱و ۰ تبدیل می‌کنیم. سپس با استفاده از کتابخانه sklearn داده‌ها را به دو بخش آموزش و تست با نسبت ۷۰ درصد آموزش و ۳۰ درصد تست تقسیم می‌کنیم. در بخش بعدی لازم است که ماتریس کوواریانس را محاسبه کنیم و با انجام عملیات ماتریسی مانند زیر این کار را انجام می‌دهیم.

```python
import numpy as np
def calculate_covariance_matrix(X, Y=None):
    """ Calculate the covariance matrix for the dataset X """
    if Y is None:
        Y = X
    ##TODO##
    n_samples = np.shape(X)[0]
    X = X - X.mean()
    covariance_matrix = (1 / (n_samples-1)) * (X.T @ X)

    return covariance_matrix
```

سپس در بخش بعدی باید که PCA را پیاده‌سازی کنیم در ابتدا ماتریس کوواریانس را محاسبه کرده، با استفاده از کتابخانه numpy، مقادیر و بردارهای ویژه آن را به دست می‌آوریم و سپس با مرتب کردن بردارهای ویژه بر اساس مقادیر ویژه منطبق با آن‌ها، اگر از لیست بردارهای مرتب شده به تعداد ابعادی که می‌خواهیم به آن کاهش ابعاد دهیم بردار ویژه برداریم، ویژگی‌های جدید استخراج شده را به دست می‌آوریم.

```python
class PCA_():
    """A method for doing dimensionality reduction by transforming the feature
    space to a lower dimensionality, removing correlation between features and
    maximizing the variance along each feature axis. This class is also used throughout
    the project to plot data.
    """
    def transform(self, X, n_components):
        """ Fit the dataset to the number of principal components specified in the
        constructor and return the transformed dataset """
        covariance_matrix = calculate_covariance_matrix(X)

        # Where (eigenvector[:,0] corresponds to eigenvalue[0])
        eigenvalues, eigenvectors = np.linalg.eig(covariance_matrix)

        # Sort the eigenvalues and corresponding eigenvectors from largest
        # to smallest eigenvalue and select the first n_components
        idx = eigenvalues.argsort()[::-1]
        eigenvalues = eigenvalues[idx]
        eigenvectors = eigenvectors[:, idx][:, :n_components]

        # Project the data onto principal components
        X_transformed = X @ eigenvectors

        return X_transformed
      ##TODO##
```

سپس PCA را بر روی داده‌های آموزش و تست داده‌های کرونا به دست می‌آوریم.
در ادامه چندین روش کاهش ابعاد معرفی شده‌است که با استفاده از کتابخانه sklearn، این روش‌ها را بر روی داده‌های آموزش و تست داده‌های کرونا به دست می‌آوریم.

ب)
در این بخش می‌خواهیم که با استفاده از شبکه‌های عصبی و به طور خاص با Autoencoder کاهش ابعاد را انجام دهیم. برای این منظور از کتابخانه keras استفاده کردیم و با استفاده از قرار دادن چند لایه به صورت Dense که تعداد نورون‌های آن‌ها کم و کمتر میشد بخش encoder و سپس با چند لایه Dense که تعداد نورون‌های آن‌ها بیشتر میشدند بخش decoder را ساختیم. سپس مدل به دست آمده را با استفاده از داده آموزش، آموزش دادیم و در نهایت مقادیر موجود در لایه Latent را از encoder استخراج کردیم.

```python
#TODO
from keras.layers import Input, Dense
from keras.models import Model

input_data = Input(shape=(20, ))
encoded = Dense(units=15, activation='relu')(input_data)
encoded = Dense(units=10, activation='relu')(encoded)
encoded = Dense(units=5, activation='relu')(encoded)
encoded = Dense(units=2, activation='relu')(encoded)
decoded = Dense(units=5, activation='relu')(encoded)
decoded = Dense(units=10, activation='relu')(decoded)
decoded = Dense(units=15, activation='relu')(decoded)
decoded = Dense(units=20, activation='sigmoid')(decoded)
autoencoder = Model(input_data, decoded)
encoder = Model(input_data, encoded)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
autoencoder.fit(X_train, X_train, epochs=50, batch_size=512, validation_data=(X_test, X_test)
# autoencoder.evaluate(X_train, X_train)
```

```
Epoch 1/50
8/8 [==============================] - 1s 54ms/step - loss: 0.6875 - accuracy: 0.0323 - val_loss: 0.6825 - val_accuracy: 0.0362
Epoch 2/50
8/8 [==============================] - 0s 12ms/step - loss: 0.6817 - accuracy: 0.0349 - val_loss: 0.6750 - val_accuracy: 0.0362
Epoch 3/50
8/8 [==============================] - 0s 10ms/step - loss: 0.6740 - accuracy: 0.0336 - val_loss: 0.6658 - val_accuracy: 0.0386
Epoch 4/50
8/8 [==============================] - 0s 14ms/step - loss: 0.6642 - accuracy: 0.0443 - val_loss: 0.6548 - val_accuracy: 0.0405
Epoch 5/50
8/8 [==============================] - 0s 12ms/step - loss: 0.6539 - accuracy: 0.0419 - val_loss: 0.6424 - val_accuracy: 0.0435
Epoch 6/50
```

۵.۲

آ)

در این بخش، می‌خواهیم که روش‌های مختلف خوشه‌بندی را بدون استفاده از کتابخانه و با کتابخانه پیاده سازی و استفاده کنیم. در ابتدا KMeans را پیاده سازی می‌کنیم. برای پیاده سازی KMeans نیاز داریم تعداد کلاسترها را بدانیم و سپس به تعداد کلاسترها کافی است که در ابتدا از داده وروی به عنوان مرکز کلاستر، به صورت تصادفی انتخاب کنیم. سپس با تابع cdist فاصله تمامی نقاط از این مراکز را به دست می‌آوریم و سپس با argmin در کتابخانه numpy، برای هر داده، مرکزی که به آن نزدیک‌تر است را محاسبه می‌کنیم و سپس با قرار دادن داده‌ها در کلاستر آن مرکز داده‌ها را خوشه بندی اولیه می‌کنیم. سپس برای تمامی داده‌های یک کلاستر میانگین ویژگی‌های تمامی اعضای آن را محاسبه می‌کنیم و به عنوان مرکز جدید کلاستر قرار می‌دهیم. حال به تعداد دفعات چرخش الگوریتم این فرآیند محاسبه نزدیکترین مرکز و انتخاب مرکز جدید را انجام می‌دهیم و در انتها کلاس مربوط به هر داده و همینطور نقاط مرکز کلاسترها را بر میگردانیم.

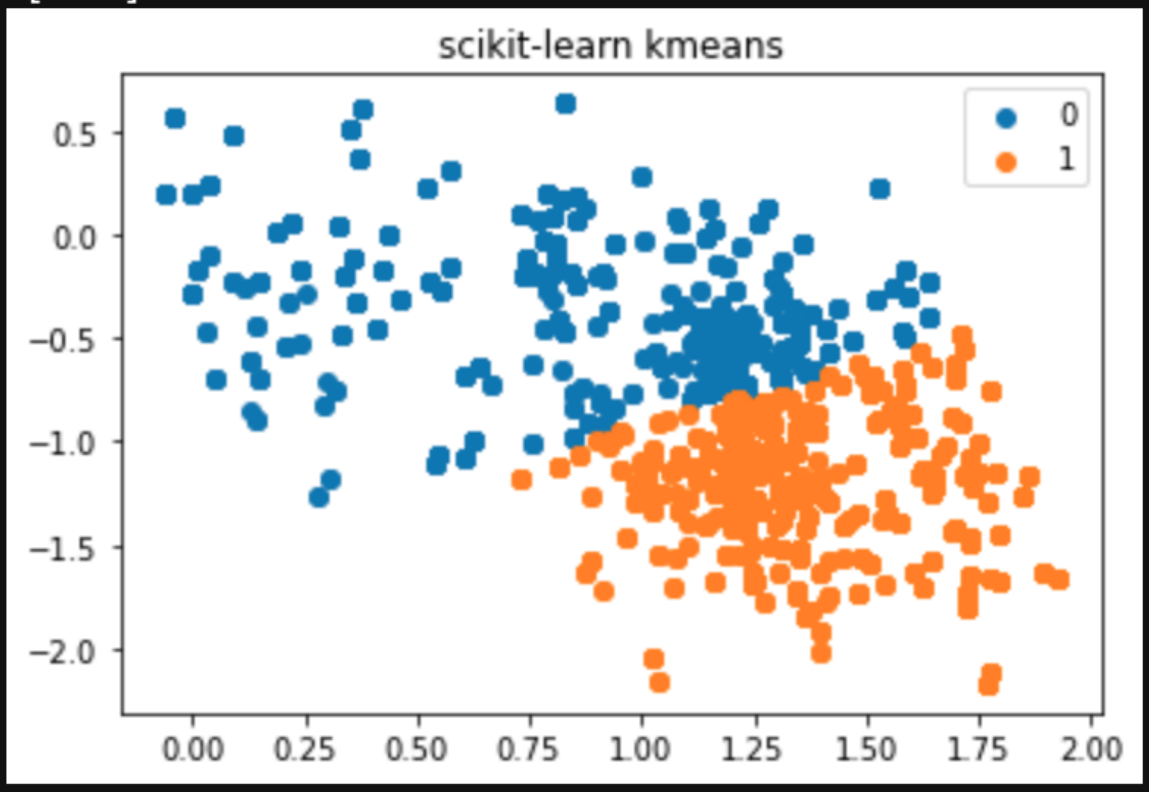
```python
#Defining our function
def kmeans(x,k, no_of_iterations):
    idx = np.random.randint(0, x.shape[0], k)
    #Randomly choosing Centroids
    centroids = x[idx]

    #finding the distance between centroids and all the data points
    distances = cdist(x, centroids ,'euclidean') #Step 2

    #Centroid with the minimum Distance
    points =  distances.argmin(axis=1)

    #Repeating the above steps for a defined number of iterations
    #Step 4
    for _ in range(no_of_iterations):
        centroids = []
        for idx in range(k):
            #Updating Centroids by taking mean of Cluster it belongs to
            temp_cent = x[points==idx].mean(axis=0)
            centroids.append(temp_cent)
        #Updated Centroids
        centroids = np.array(centroids)

        distances = cdist(x, centroids ,'euclidean')
        points = distances.argmin(axis=1)
    return points, centroids
```

سپس تابع predict و accuracy به منظور ارزیابی مدل ساخته شده تعریف شده‌اند که اولی با استفاده از داده ورودی و مراکز کلاسترها، به هر داده یک برچسب می‌زند و دومی دقت تخمین را محاسبه می‌کند.

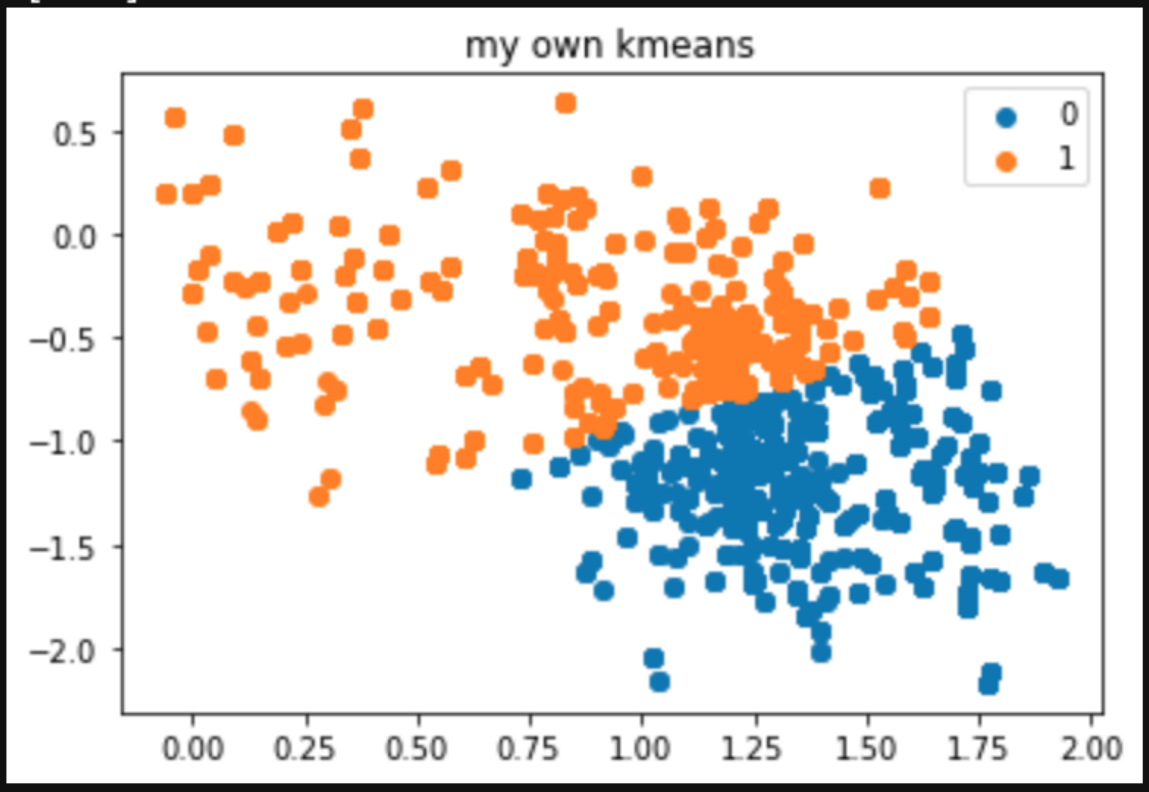
```python
def predict_kmeans(X, centroids):
    distances = cdist(X, centroids, 'euclidean')
    points = distances.argmin(axis=1)
    return points

def accuracy(y_pred, y_test):
    true = (y_pred == y_test).sum()
    return true/len(y_pred)
```

سپس می‌خواهیم که این کلاسترهای انجام شده به وسیله KMeansای که خودمان نوشته‌ایم و KMeans از کتابخانه sklearn را مصور کنیم و نتایج آن‌ها را مقایسه کنیم که تقریبا با یکدیگر برابر بودند.

```
[0 1]
```

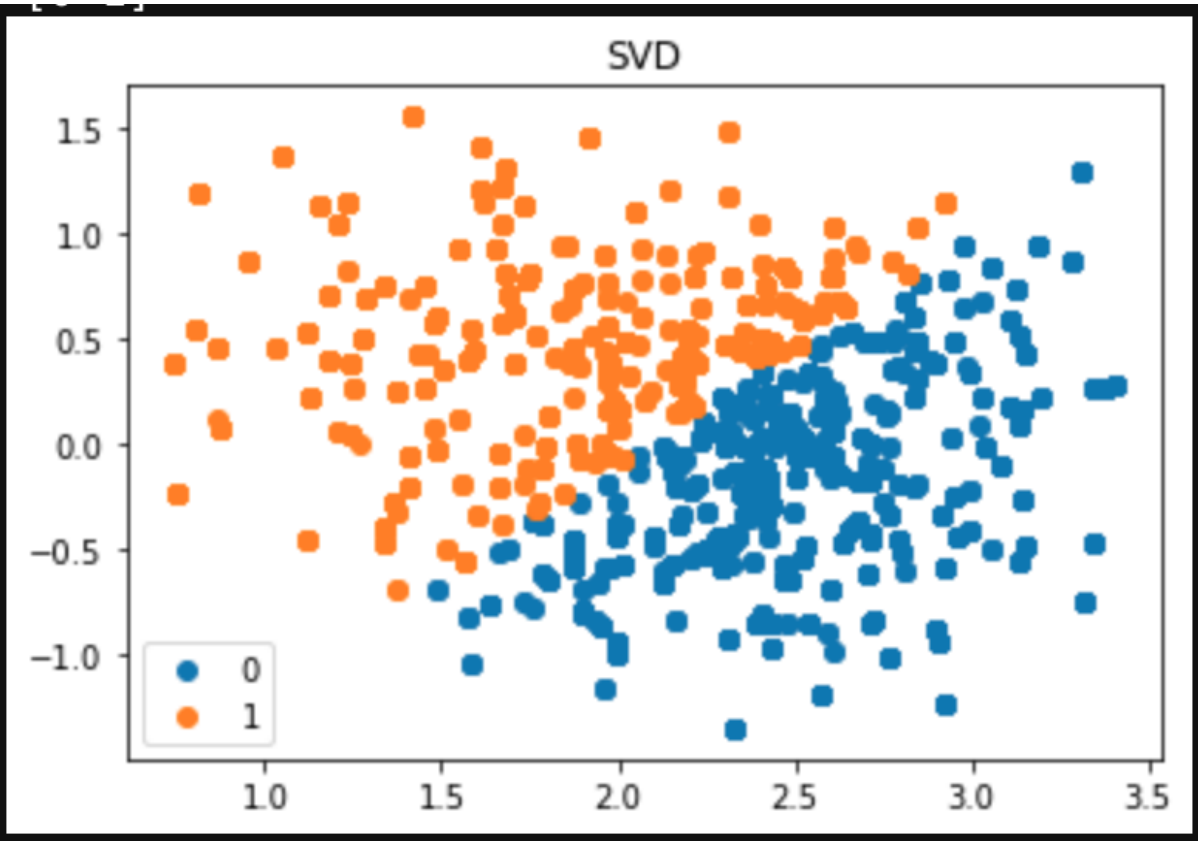

scikit-learn kmeans
0
1
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

```
[0 1]
```

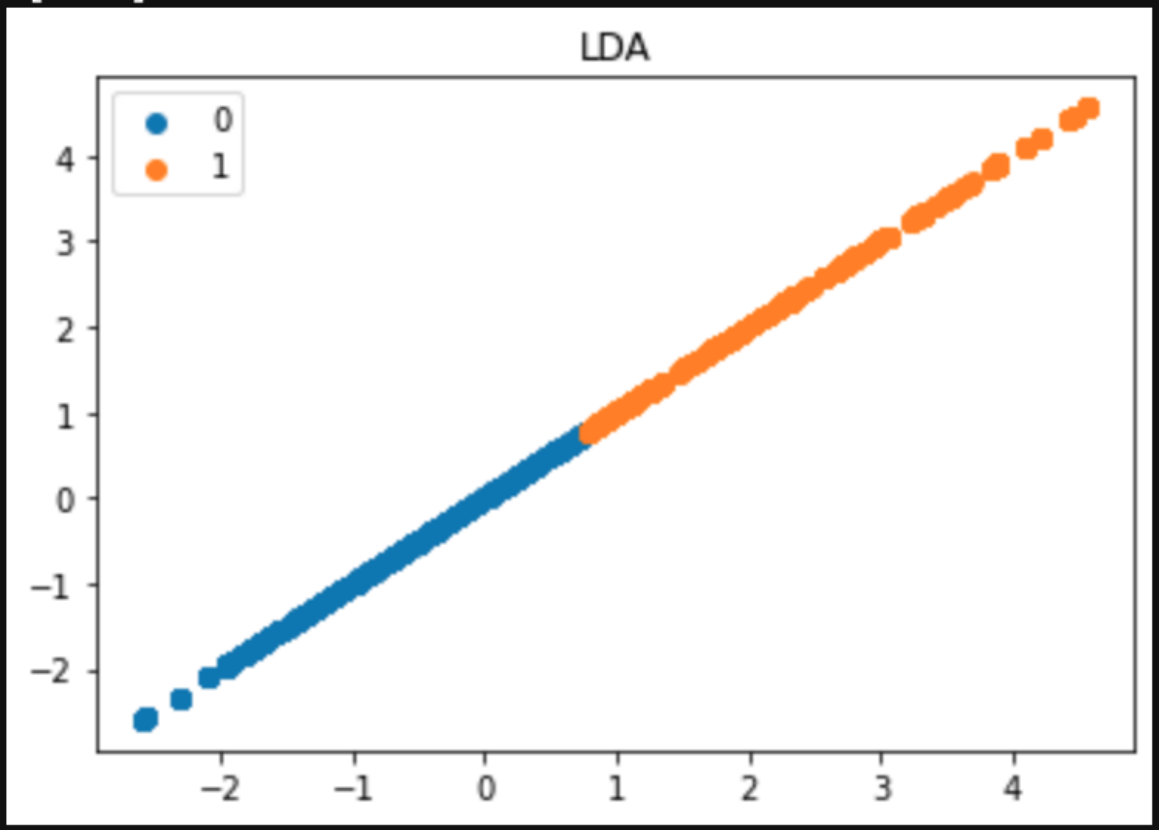

my own kmeans
0
1
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

```
[0 1]
```

سپس با استفاده از KMeansای که خودمان نوشته‌بودیم، حاصل خوشه بندی روی تمامی داده‌های کاهش ابعاد یافته از مرحله قبل را پلات کردیم که در نوت بوک موجود است. به طور مثال دو نمونه در زیر آمده‌است.



[0 1]



[0 1]

سپس با استفاده از داده‌های تست، مدل آموزش داده شده روی داده آموزش را ارزیابی کردیم که دقتی برابر ۸۰ درصد ارائه داد.
در بخش بعدی الگوریتم EM را پیاده سازی کردیم که تقریبا تمام آن انجام شده‌بود و فقط با اضافه کردن ۴ خط به نمودارهای مورد نیاز رسیدیم.
و در نهایت با روش‌های مختلف کلاسترینگ داده‌ها را خوشه بندی کرده و حاصل را پلات کردیم.
همینطور داده‌ها را بر اساس برچسب واقعی داده آموزش رسم کردیم که نشان داد تمامی روش‌های کلاسترینگ در بخشی خوب عمل نکرده‌اند و با خوشه‌بندی واقعی تفاوت داشتند. و در نهایت به نظر میرسید که Kmeans بهترین عملکرد را ارائه میداد.