نام درس : شناسایی آماری الگوریتم

تمرین شماره 4

نام و شماره دانشجویی : فاطمه توکلی , 400131016

سئوال 2 پیاده سازی:

a)همانطور که خواسته شده k-means را پیاده کرده و دقت خوشه بندی را روی دیتاست داده شده به دست آورده :

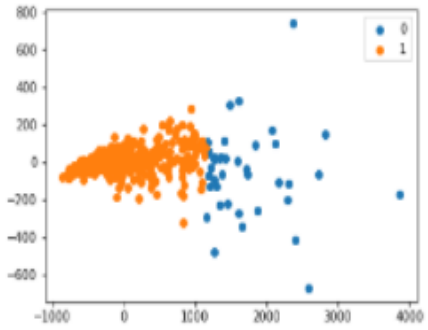
Part a

In [10]:
```
data = pd.read_csv('breast_data.csv',header=None)
pca = PCA(2)
df = pca.fit_transform(data)
```

In [8]:
```
def kmeans(x,k,mu,tol,no_of_iterations):

    centroids = x[mu, :]
    distances = cdist(x, centroids ,'euclidean')

    C = np.array([np.argmin(i) for i in distances])

    for m in range(no_of_iterations):
        new_centroids = []
        for mu in range(k):
            temp_cent = x[C==mu].mean(axis=0)
            new_centroids.append(temp_cent)

        if np.allclose(centroids,new_centroids,tol) :
            centroids = np.vstack(new_centroids)
        else:
            break
        distances = cdist(x, centroids ,'euclidean')
        C = np.array([np.argmin(i) for i in distances])

    return C
```

In [7]:
```
mu = np.random.choice(len(df), 2 , replace=False)

label = kmeans(df,2,mu,0.000001,10000)
u_labels = np.unique(label)

for i in u_labels:
    plt.scatter(df[label == i , 0] , df[label == i , 1] , label = i)
plt.legend()
plt.show()
```

```
[265 232]
```



In [10]:
```
y = pd.read_csv('breast_labels.csv',header=None)
accuracy_score(y,label)
```

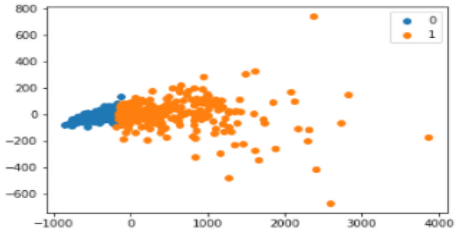
Out[10]:
```
0.5887521968365553
```

b) تعریف مقدار اولیه دسته بندی به صورت رندوم است و در هربار تکرار نقاط شروع متفاوت و در نتیجه نتایج ما متفاوت خواهد بود برای مثال :

Part b

I choose initial points random so by each run you start with diffrenet point.

In [124]:
```
mu = np.random.choice(len(df), 2 , replace=False)

label = kmeans(df,2,mu,0.000001,10000)
unique_labels = np.unique(label)

for i in unique_labels:
    plt.scatter(df[label == i , 0] , df[label == i , 1] , label = i)
plt.legend()
plt.show()
```



In [125]:
```
y = pd.read_csv('breast_labels.csv',header=None)
accuracy_score(y,label)
```

Out[125]: 0.8857644991212654

c) برای حل این قسمت از mu_init استفاده کرده و از ابتدای لیست به صورت جفت جفت انتخاب کردم و به عنوان نقاط اولیه به k-means داده علت انتخاب جفت جفت وجود دو خوشه بندی بود و بخاطر اینکه جزو نقاط اولیه در خود دیتاست نبودند نیاز به اندکی تغییر در k-means داشته و تعدادی از mu_init ها و accuracy را اورده شده:

برای انتخاب بهترین mu وaccuracy، دقت ها به ارایه ایی اضافه شد و بزرگترین آن ها برگردانده شده است

```
def modify_kmeans(x,k,mu,tol,no_of_iterations):

    centroids = mu
    distances = cdist(x, centroids ,'euclidean')

    C = np.array([np.argmin(i) for i in distances])

    for m in range(no_of_iterations):
        new_centroids = []
        for mu in range(k):
            temp_cent = x[C==mu].mean(axis=0)
            new_centroids.append(temp_cent)

        if np.allclose(centroids,new_centroids,tol) :
            centroids = np.vstack(new_centroids)
        else:
            break
        distances = cdist(x, centroids ,'euclidean')
        C = np.array([np.argmin(i) for i in distances])

    return C
```

```
best_accuracy = []
for i in range(len(mu_init)-1):
    a = mu_init[[i,i+1],:]
    result = modify_kmeans(df,2,a,0.00001,10)
    accuracy = accuracy_score(y,result)
    best_accuracy.append(accuracy)
    print('\nfor initial center : \n' + str(a) + '\n accuracy is : ' + str(accuracy))
```

```
for initial center :
[[ 1.1125  0.4038]
 [-0.7193 -0.3442]]
 accuracy is : 0.09666080843585237

for initial center :
[[-0.7193 -0.3442]
 [-0.2734 -0.9801]]
 accuracy is : 0.9261862917398945

for initial center :
[[-0.2734 -0.9801]
 [ 0.6774  1.8573]]
 accuracy is : 0.8453427065026362
```

```
for initial center :
[[ 0.4257 -0.4891]
 [-0.0745 -0.6949]]
 accuracy is : 0.09666080843585237

for initial center :
[[-0.0745 -0.6949]
 [-1.3988 -0.1804]]
 accuracy is : 0.0843585237258348

for initial center :
[[-1.3988 -0.1804]
 [ 0.1923  1.2563]]
 accuracy is : 0.8927943760984183
```

```
In [129]: max(best_accuracy)

Out[129]: 0.9261862917398945
```

d) بعد از به دست اوردن میانگین های واقعی ممکن است خطای دسته بندی ما کم شود با توجه به توزیع داده ها و تعداد میانگین انتخابی میتوان خطا را کم کرد در این راستا می توان از متد هایی (elbow method) کمک گرفت

E) در بین دسته بند های بدون نظارت این نوع دسته بندی با kmeans در شرایط ایده ال خطای مناسب و پیاده سازی اسانی دارد ممکن است در این سمئله روش های با نظارت جواب بهتری به ما بدهند

kmeans error is e<bayes error < 2e so usually it works Optimal if your dataset is linearly seperable

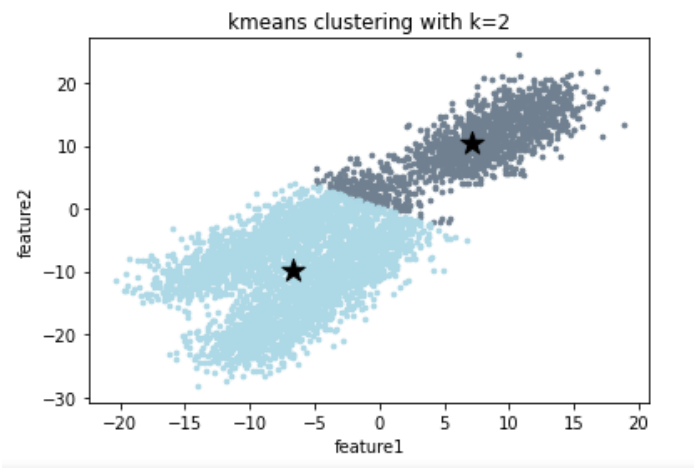
3.

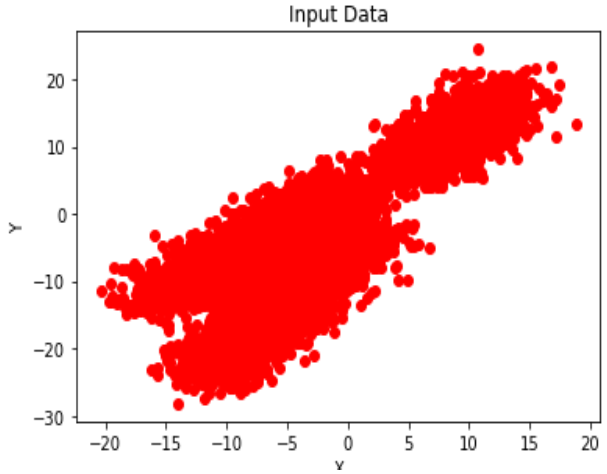
ابتدا k-means پیاده میکنیم :

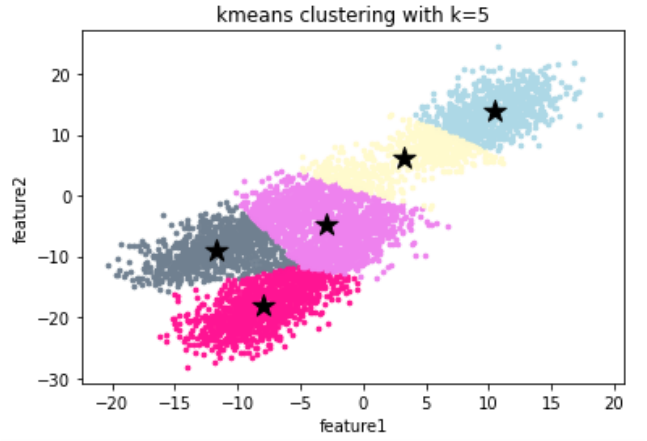
part a

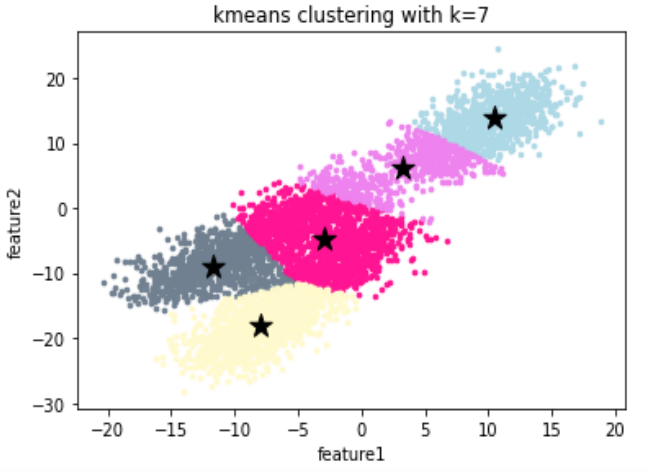
```
def dist(a, b, ax=1):
    return np.linalg.norm(a - b, axis=ax)


def kmeans_clustering(data, k):
    center_x = np.random.randint(np.min(data[:, 0]), np.max(data[:, 0]), size=k)
    center_y = np.random.randint(np.min(data[:, 1]), np.max(data[:, 1]), size=k)
    centers = [list(a) for a in zip(center_x, center_y)]


    new_centers = []
    while not np.array_equal(new_centers, np.array(centers)):

        if new_centers:
            centers = new_centers
            new_centers = []

        clusters = np.zeros(len(data))
        for i in range(len(data)):
            distances = dist(data[i], centers)
            cluster = np.argmin(distances)
            clusters[i] = cluster

        colors = ['slategray', 'lightblue', 'deeppink', 'violet', 'lemonchiffon', 'tomato', 'darkorange']
        fig, ax = plt.subplots()

        for i in range(k):
            points = np.array([data[j] for j in range(len(data)) if clusters[j] == i])
            if points.size != 0:
                new_centers.append(np.mean(points, axis=0))
                ax.scatter(points[:, 0], points[:, 1], s=7, c=colors[i])

        ax.scatter(np.array(centers)[:, 0], np.array(centers)[:, 1], marker='*', s=200, c='k')
        plt.title('kmeans clustering with k=' + str(k))
        plt.xlabel('feature1')
        plt.ylabel('feature2')
        plt.show()
```

a) ابتدا داده های ورودی را به نمایش گذاشتیم و سپس k-means پیاده شده را با k=2,3,5,7 اجرا کرده و center هر خوشه را با ستاره سیاه رنگ در هر دسته بندی نشان داده ایم :







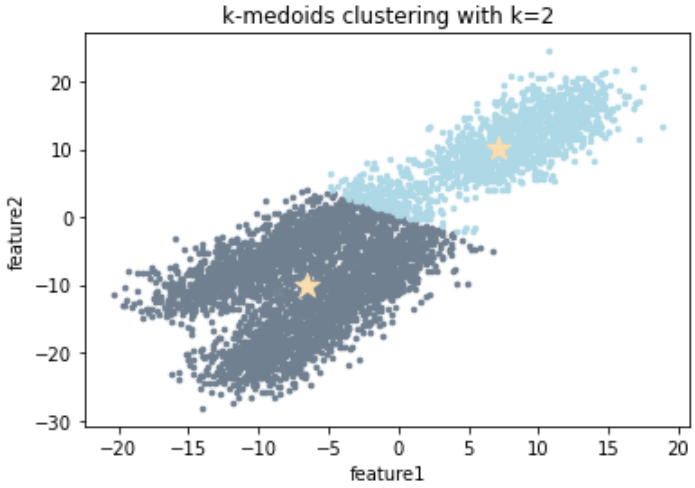


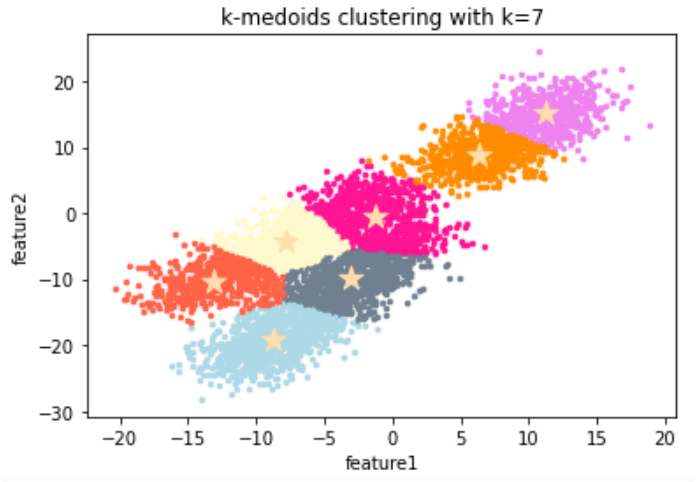
b) برای k=1,....,10 متد elbow را اجرا کرده و نقطه شکست اولیه در k=3 مشاهده می شود:

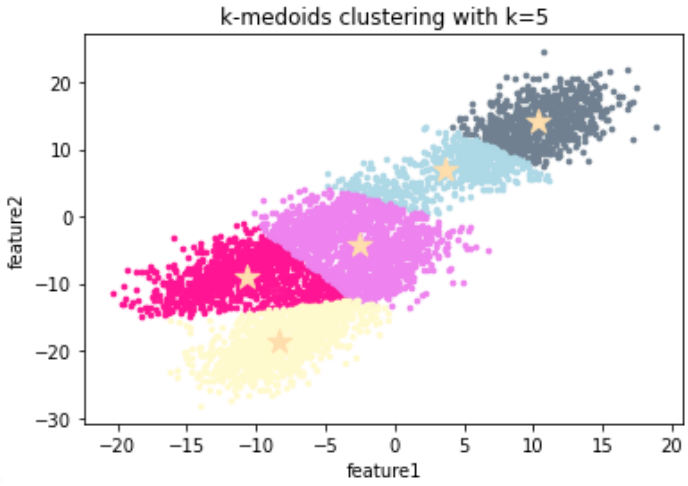
c) تنها تفاوت k_medoids در این است که مراکز باید جزو داده های باشند تغییرات لازم را داده و دوباره پیاده سازی میکنیم:

part c

```
def dist(a, b, ax=1):
    return np.linalg.norm(a - b, axis=ax)


def kmedoids_clustering(data, k):
    centers = data[np.random.choice(data.shape[0], k, replace=False), :]

    plt.scatter(data[:, 0], data[:, 1], c='#050505', s=7)
    plt.scatter(centers[:, 0], centers[:, 1], marker='*', s=200, c='g')
    plt.title('original data with initialized random centroids')
    plt.xlabel('feature1')
    plt.ylabel('feature2')

    new_centers = []
    while not np.array_equal(new_centers, np.array(centers)):

        if new_centers:
            centers = new_centers
            new_centers = []

        clusters = np.zeros(len(data))
        for i in range(len(data)):
            distances = dist(data[i], centers, 1)
            cluster = np.argmin(distances)
            clusters[i] = cluster

        colors = ['slategray', 'lightblue', 'deeppink', 'violet', 'lemonchiffon', 'tomato', 'darkorange']
        fig, ax = plt.subplots()

        for i in range(k):
            points = np.array([data[j] for j in range(len(data)) if clusters[j] == i])
            if points.size != 0:
                distances = dist(np.mean(points, axis=0), points, 1)
                new_centers.append(points[np.argmin(distances)])
                ax.scatter(points[:, 0], points[:, 1], s=7, c=colors[i])

        ax.scatter(np.array(centers)[:, 0], np.array(centers)[:, 1], marker='*', s=200, c='#FFDEAD')
        plt.title('k-medoids clustering with k=' + str(k))
        plt.xlabel('feature1')
        plt.ylabel('feature2')
        plt.show()
```

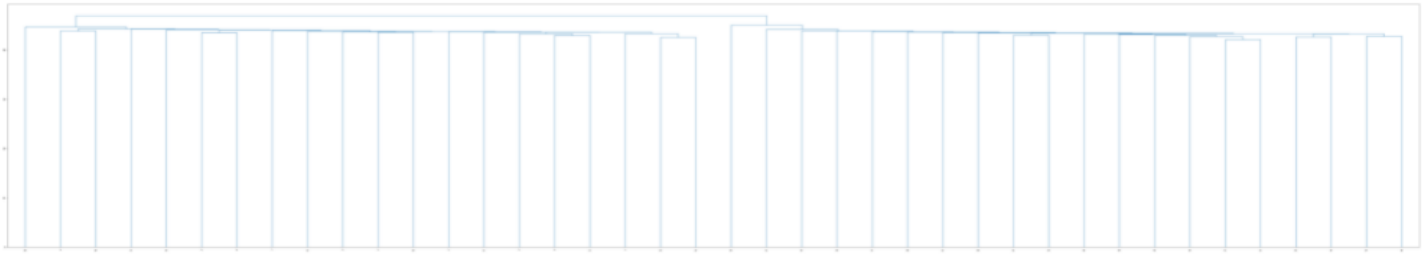




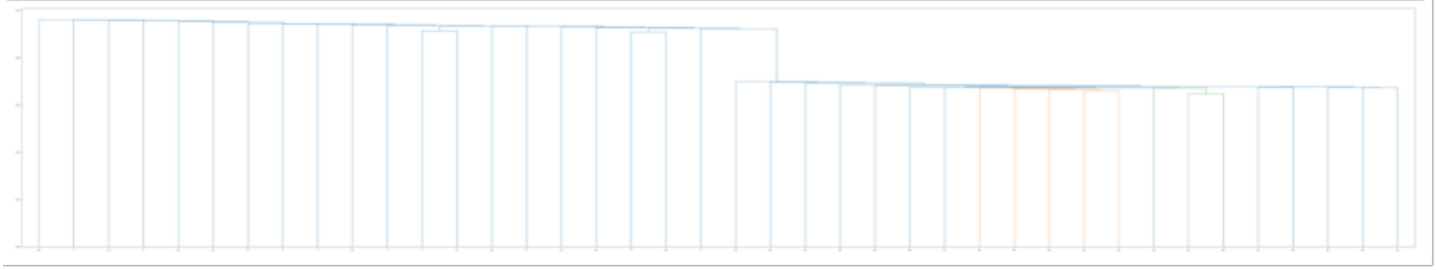


d) هرکدام از hierarchical clustering ها پیاده سازی شده و در کد پیوست موجود می باشد و نتایج به شرح زیر می باشد:

```
Z = hierarchy.linkage(np.array(data.T), 'single', 'euclidean')
plt.figure(figsize=(100, 20))
dn = hierarchy.dendrogram(Z)
```



```
Z = hierarchy.linkage(np.array(data.T), 'single', 'correlation')
plt.figure(figsize=(100, 20))
dn = hierarchy.dendrogram(Z)
```



```
Z = hierarchy.linkage(np.array(data.T), 'complete', 'euclidean')
plt.figure(figsize=(100, 20))
dn = hierarchy.dendrogram(Z)
plt.show()
```

```
Z = hierarchy.linkage(np.array(data.T), 'complete', 'correlation')
plt.figure(figsize=(100, 20))
dn = hierarchy.dendrogram(Z)
plt.show()
```



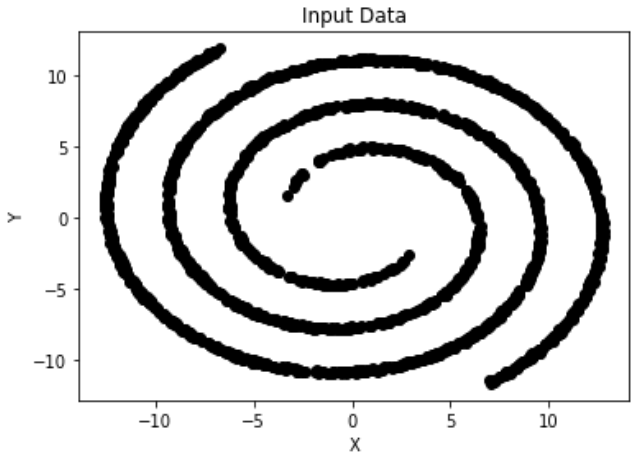
```
Z = hierarchy.linkage(np.array(data.T), 'average', 'euclidean')
plt.figure(figsize=(100, 20))
dn = hierarchy.dendrogram(Z)
plt.show()
```

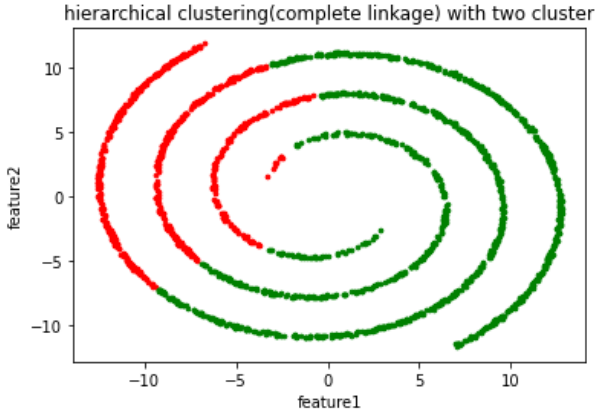


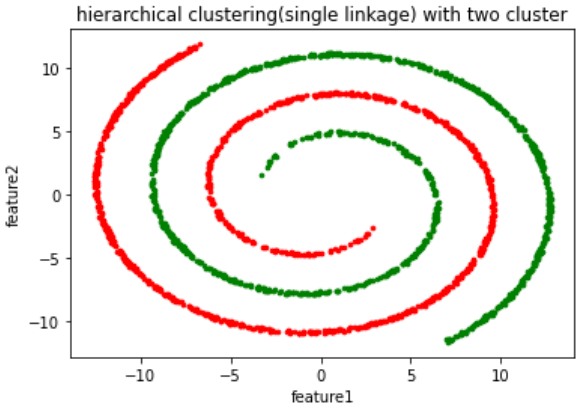
```
Z = hierarchy.linkage(np.array(data.T), 'average', 'correlation')
plt.figure(figsize=(100, 20))
dn = hierarchy.dendrogram(Z)
plt.show()
```


همانطور که مشاهده میشود نتایج در خوشه بندی های متفاوت است و بستگی به نوع خوشه بندی و معیار فاصله دارد

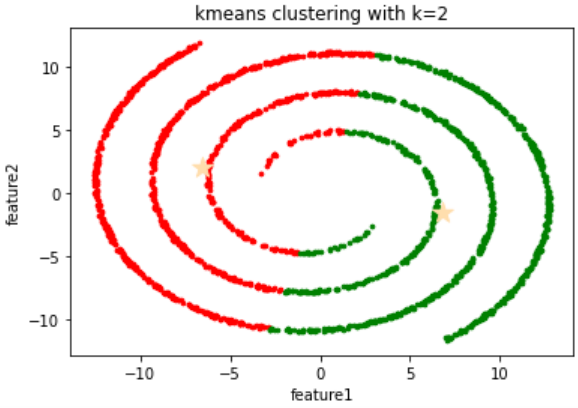
f) پیاده سازی ها به علت زیاد بودن در گزارش آورده نشده و میتوان در کد پیوست مشاهده کرد برای درک عملکرد نتایج به شرح زیر می باشد:








```
single linkage: 1.0
score of single linkage: 1.0

complete linkage: 0.036320148104460596
score of complete linkage: 0.6045

kmeans: 0.023528929442967347
score of kmeans: 0.59
```



4.

a) ابتدا pca را پیاده کرده و reigenvalue را sort کرده و 20 تا بهترین را به عنوان خروجی برمیگردانیم

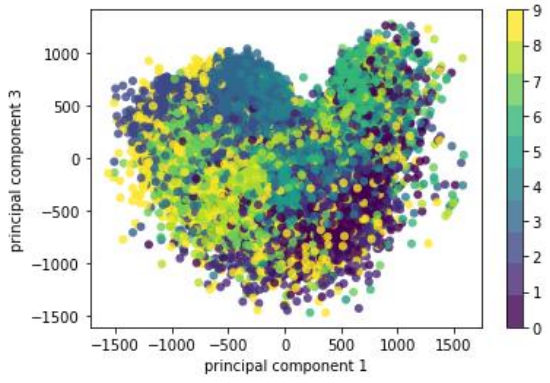
PCA

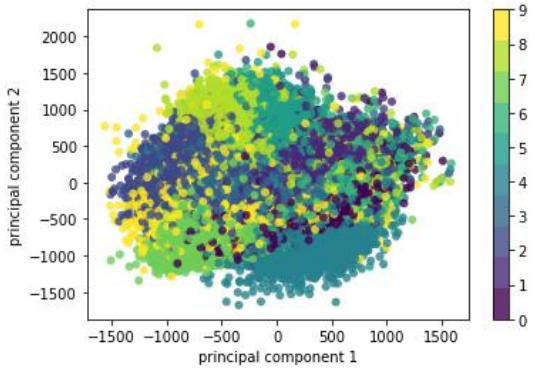
```
def PCA(x):
    #compute mean
    mean = np.mean(x, axis= 0)
    mean_data = x - mean

    # Compute covariance matrix
    cov = np.cov(mean_data.T)
    cov = np.round(cov, 2)

    # Perform eigen decomposition of covariance matrix
    eig_val, eig_vec = np.linalg.eig(cov)

    # Sort eigen values and corresponding eigen vectors in descending order
    indices = np.arange(0,len(eig_val), 1)
    indices = ([x for _,x in sorted(zip(eig_val, indices))])[::-1]
    eig_val = eig_val[indices]
    eig_vec = eig_vec[:,indices]
    max_eig = eig_val[0:20]
    print("top 20 of Sorted Eigen values ", max_eig, "\n")

    # Get explained variance
    sum_eig_val = np.sum(eig_val)
    explained_variance = eig_val/ sum_eig_val
    cumulative_variance = np.cumsum(explained_variance)

    # Take transpose of eigen vectors with data
    pca_data = np.dot(mean_data, eig_vec)
    return pca_data,eig_val
```

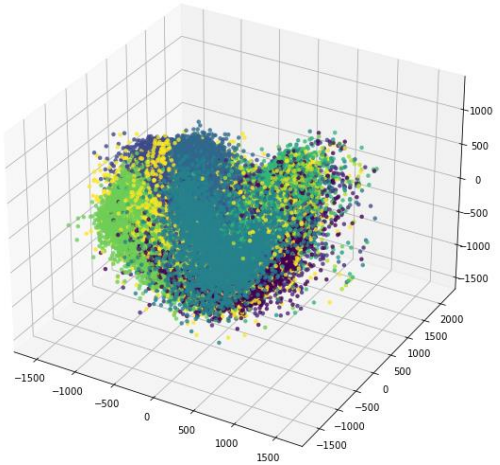
```
pca_data,eig_value = PCA(dt)
```

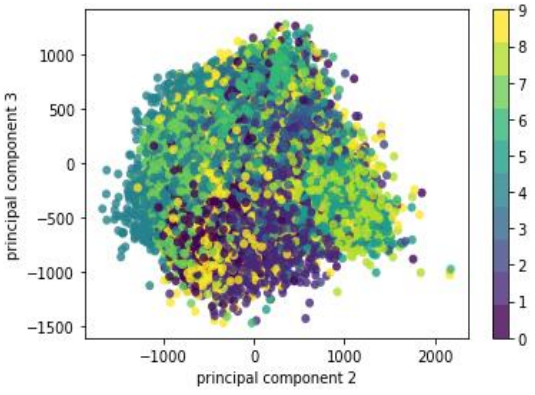
```
top 20 of Sorted Eigen values  [284765.00077037 276909.80324943 159995.1528915  135060.21091957
 105714.10748809 101280.08805835  87571.24118236  80833.85961203
  72591.14578202  64722.4816955   58352.21724876  55682.61059702
  53992.2666051   51298.59036831  46820.62174801  46547.7500008
  45423.80578955  43774.08701081  42466.96332802  41816.50578826]
```

سه تا از principal component ها را برحسب یکدیگر و با برپسب دهی برحسب کلاس ها رسم کرده ایم









b) پیاده سازی LDA و plot خواسته شده:

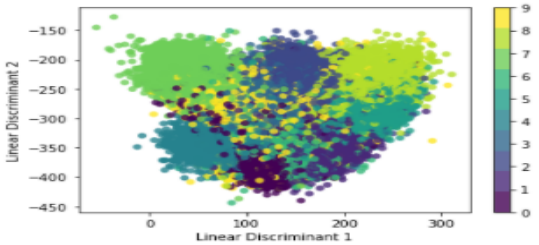
LDA

```
class LDA:
    def __init__(self, n_components):
        self.n_components = n_components
        self.linear_discriminants = None

    def fit(self, X, y):
        n_features = X.shape[1]
        class_labels = np.unique(y)

        # Within class scatter matrix:
        # SW = sum((X_c - mean_X_c)^2 )

        # Between class scatter:
        # SB = sum( n_c * (mean_X_c - mean_overall)^2 )

        mean_overall = np.mean(X, axis=0)
        SW = np.zeros((n_features, n_features))
        SB = np.zeros((n_features, n_features))
        for c in class_labels:
            X_c = X[y == c]
            mean_c = np.mean(X_c, axis=0)
            # (4, n_c) * (n_c, 4) = (4,4) -> transpose
            SW += (X_c - mean_c).T.dot((X_c - mean_c))

            # (4, 1) * (1, 4) = (4,4) -> reshape
            n_c = X_c.shape[0]
            mean_diff = (mean_c - mean_overall).reshape(n_features, 1)
            SB += n_c * (mean_diff).dot(mean_diff.T)

        # Determine SW^-1 * SB
        A = np.linalg.pinv(SW).dot(SB)
        # Get eigenvalues and eigenvectors of SW^-1 * SB
        eigenvalues, eigenvectors = np.linalg.eig(A)
        # -> eigenvector v = [:,i] column vector, transpose for easier calculations
        # sort eigenvalues high to low
        eigenvectors = eigenvectors.T
        idxs = np.argsort(abs(eigenvalues))[::-1]
        eigenvalues = eigenvalues[idxs]
        eigenvectors = eigenvectors[idxs]
        # store first n eigenvectors
        self.linear_discriminants = eigenvectors[0 : self.n_components]

    def transform(self, X):
        # project data
        return np.dot(X, self.linear_discriminants.T)
```

```
# Testing
if __name__ == "__main__":
    # Imports
    import matplotlib.pyplot as plt
    from sklearn import datasets

#     data = datasets.load_iris()
    X, y = dt,label

    # Project the data onto the 2 primary linear discriminants
    lda = LDA(2)
    lda.fit(X, y)
    X_projected = lda.transform(X)

    print("Shape of X:", X.shape)
    print("Shape of transformed X:", X_projected.shape)

    x1, x2 = X_projected[:, 0], X_projected[:, 1]

    plt.scatter(
        x1, x2, c=y, edgecolor="none", alpha=0.8, cmap=plt.cm.get_cmap("viridis", 10)
    )

    plt.xlabel("Linear Discriminant 1")
    plt.ylabel("Linear Discriminant 2")
    plt.colorbar()
    plt.show()
```

```
Shape of X: (80000, 784)
Shape of transformed X: (80000, 2)
```

```
C:\Users\fateme\anaconda3\lib\site-packages\numpy\core\_asarray.py:171: ComplexWarning: Casting complex values to real discards
the imaginary part
  return array(a, dtype, copy=False, order=order, subok=True)
```



c) k-means را بر روی pca_data که از اعمال pca به دست آمده به دست می آوریم و با k=3,7,10 نتایج خوبی نمیدهد ونتوانسته به خوبی متمایز کند و جواب به دست آمده در قسمت a بهتر است

k-Means

Select random initial centroids and set K = 3, 7, 10.

```
fs = pd.DataFrame(pca_data).iloc[:,[0,1]]
X_train, X_test, y_train, y_test = train_test_split(fs, label, test_size=0.8 , random_state=10)
```

```
C = KMeans(n_clusters = 3).fit(X_train)
pred = C.predict(X_test)
accuracy_score(y_test,pred)
```

```
0.123796875
```

```
C = KMeans(n_clusters = 7).fit(X_train)
pred = C.predict(X_test)
accuracy_score(y_test,pred)
```

```
0.10353125
```

```
C = KMeans(n_clusters = 10).fit(X_train)
pred = C.predict(X_test)
accuracy_score(y_test,pred)
```

```
0.05665625
```

d) در k=3 با مقدار دهی جدید نتیجه بهتر شده است:

K = 3, initial centroids are the mean of samples of classes {1,3,5,7}, {2,4}, and {6,8,9,10}

```
df = pd.DataFrame(fs)
df[2] = label
np.unique(label)
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=uint8)
```

```
one = df.loc[df[2] == 0].iloc[:,[0,1]].mean()
three = df.loc[df[2] == 2].iloc[:,[0,1]].mean()
five = df.loc[df[2] == 4].iloc[:,[0,1]].mean()
seven = df.loc[df[2] == 6].iloc[:,[0,1]].mean()
a =( one+three+five+seven)/4
a
```

```
0    222.357353
1   -219.280847
dtype: float64
```

```
two = df.loc[df[2] == 1].iloc[:,[0,1]].mean()
four = df.loc[df[2] == 3].iloc[:,[0,1]].mean()
b = (two + four)/2
b
```

```
0      8.033282
1    183.412071
dtype: float64
```

```
six = df.loc[df[2] == 5].iloc[:,[0,1]].mean()
eight = df.loc[df[2] == 7].iloc[:,[0,1]].mean()
nine = df.loc[df[2] == 8].iloc[:,[0,1]].mean()
ten = df.loc[df[2] == 9].iloc[:,[0,1]].mean()
c = (six + eight + nine + ten)/4
c
```

```
0   -227.562657
1    127.973875
dtype: float64
```

```
fs = pd.DataFrame(pca_data).iloc[:,[0,1]]
X_train, X_test, y_train, y_test = train_test_split(fs, label, test_size=0.8 , random_state=10)
z = [a,b,c]
z = np.array(z)
```

```
C = KMeans(n_clusters=3,init=z).fit(X_train)
pred = C.predict(X_test)
accuracy_score(y_test,pred)
```

```
0.1808125
```

در k=7 دسته بندی مطابق صورت سئوال انجام شده و نتیجه نهایی به صورت زیر است و اندکی از حالت رندوم بهتر است:

```
C = KMeans(n_clusters=7,init=res).fit(X_train)
pred = C.predict(X_test)
accuracy_score(y_test,pred)
```

```
0.158640625
```

در k=10 نتیجه به صورت چشمگیری افزایش یافته است:

```
C = KMeans(n_clusters=10,init=result).fit(X_train)
pred = C.predict(X_test)
accuracy_score(y_test,pred)
```

```
0.45646875
```

با این حال هنوز دقت ما زیر 0.5 است.

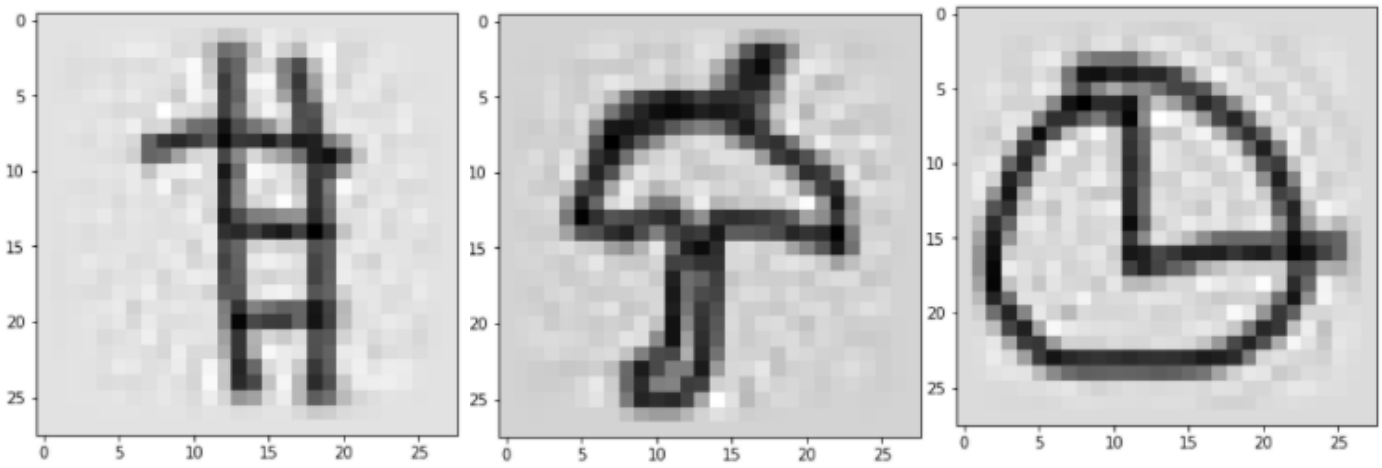
e) با استفاده از 270 principal component میتوان به 0.95< correlation رسید:

part e

```
pca = PCA()
pca_dt = pca.fit(dt)

var_cumu = np.cumsum(pca.explained_variance_ratio_)*100

k = np.argmax(var_cumu>95)
print("Number of components explaining 95% variance: "+ str(k))
```

```
Number of components explaining 95% variance: 270
```

سه نمونه تولید شده:



(f

```
C = KMeans(n_clusters=10,init=result).fit(finall_result)
pred = C.predict(finall_result)
accuracy_score(label,pred)
```

```
0.5756125
```