Amirkabir University of Technology
(Tehran Polytechnic)
Department of Computer Engineering

Statistical Pattern Recognition

Assignment IV

Soroush Mahdi
99131050

supervised by
Dr. Mohammad Rahmati

February 17, 2021

Contents

1 PCA vs. LDA: Comparison of Two Different Linear Projections

1.1 a

pca is unsupervised and we know that the variation of values along the line that pca projects points on it should be maximal. so he first principal component will be aligned along the direction of major continuity.

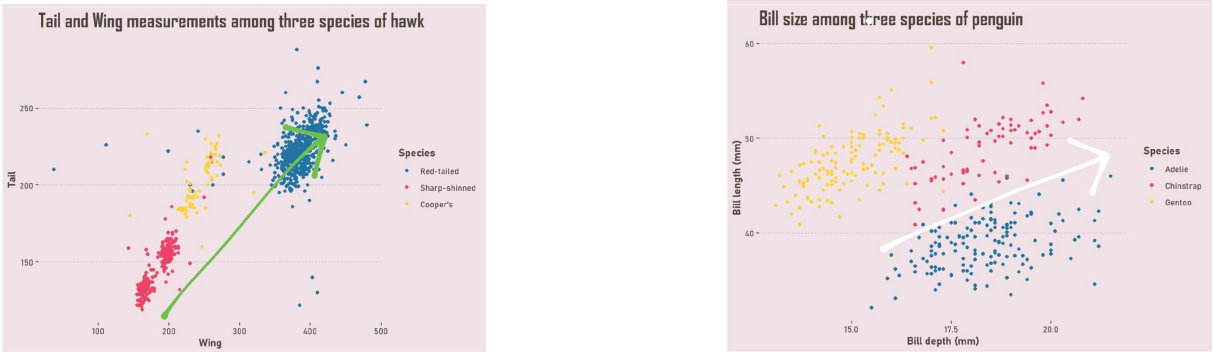

Figure 1: first pca direction for blue and red classes

1.2 b

FLD selects a projection that maximizes the class separation and it is supervised.

Figure 2: first fld direction for blue and red classes

1.3 c

Figure 3: first pca direction for yellow and red classes

1.4 d

Figure 4: first fld direction for yellow and red classes

1.5 e

Figure 5: first pca direction for all classes

1.6 f

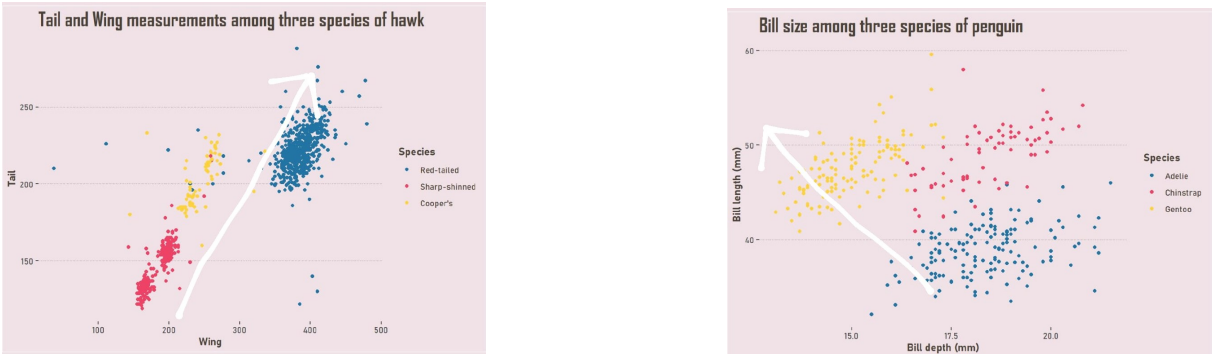

Figure 6: first fld direction for all classes

2 Understanding the Behavior of Clustering Techniques

2.1 a

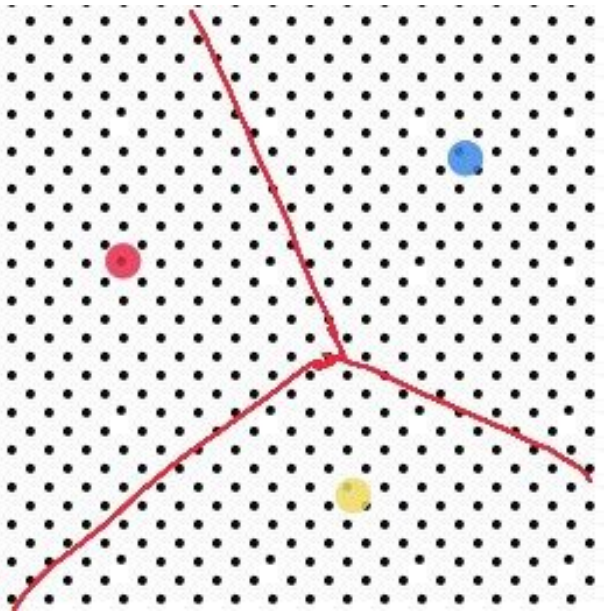

Figure 7: boundaries can rotate based on initial centroids

2.2 b

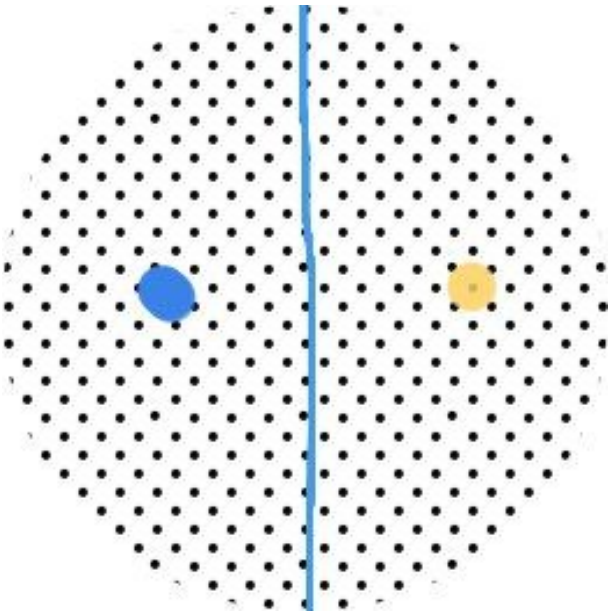

Figure 8: boundaries can rotate based on initial centroids

2.3 c

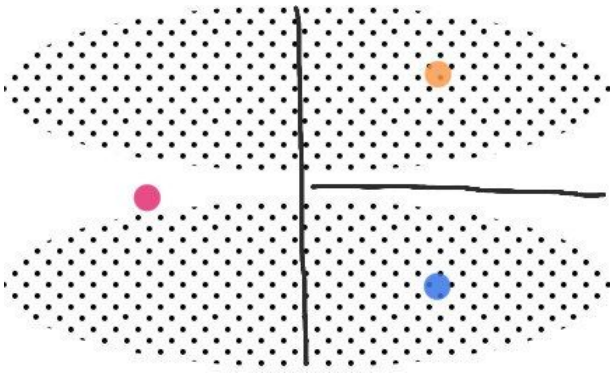

Figure 9: c

2.4 d

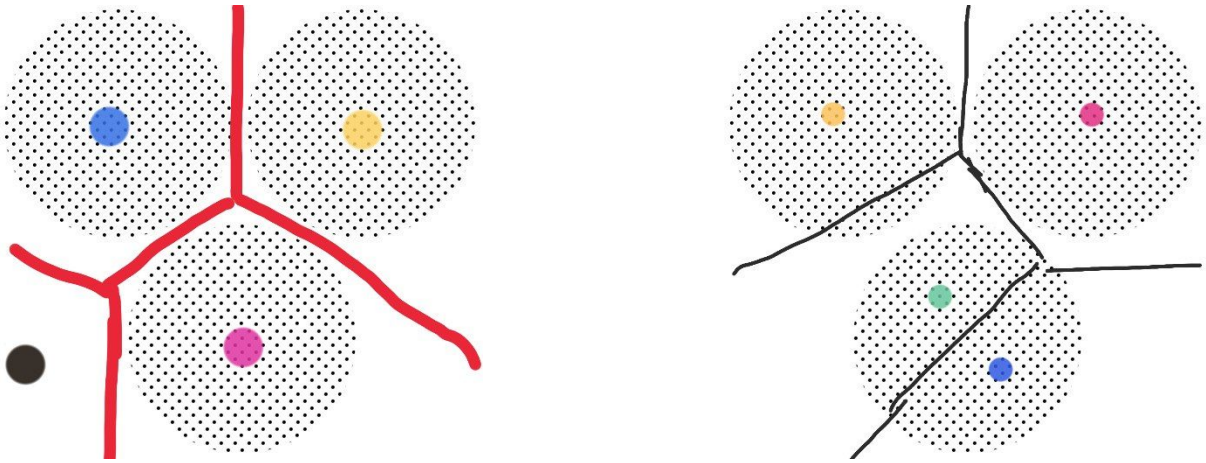

Figure 10: the left figure is global minimum

2.5 e

Figure 11: the left figure is global minimum

2.6 f

the diagram in the right.

2.7 g

it can't find the patterns that are created by empty spaces. in other words spaces that are surrounded by points.

3 Why Naples-Style Pizzas Are So Delicious?

3.1 a

```python
def pca(data , n_component):
    """performs PCA on a given dataset and return reduced data

    Args:
        data (m*n numpy array): [m samples that each has n faetures]
        n_component ([int]): [num of pca components to keep]

    Returns:
        [m * n_component numpy array]: [reduced data]
    """
    # computing mean of data set
    mu = data.mean(axis = 0)
    mu = mu.reshape(data.shape[1] , 1).T
    #subtracting mean from data samples
    data = data - mu
    #computing scatter matrix
    S = np.cov(data.T)*(data.shape[0]-1)
    #computing eigen values and eigen  vectors of scatter matrix
    eigen_values , eigen_vecs = np.linalg.eig(S)
    #selecting eigen vectors corresponding to first biggest n_component
    ind = eigen_values.argsort()[-n_component:][::-1]
    E = eigen_vecs[ : , ind ]
    #computing reduced data
    reduced_data = E.T @ data.T
    return reduced_data.T
```

Listing 1: PCA implementation

```python
#reading data
df = pd.read_csv('doughs.dat' , sep = ' ')
data = df.drop(['Restaurant'] , axis = 1).to_numpy()

reduced_data = pca(data , 3)

print(reduced_data)
```

Listing 2: performing pca on the given dataset

we droped restaurant column from data for performing pca since it's the label of the data

```
[[ 13.40370431  11.37083773   0.30190493]
 [ 10.2126015    4.24606046   0.2822107 ]
 [ -3.64788876  12.49211826   0.33206521]
 [ 13.89226015   5.2314273    0.45944424]
 [ 20.14596971   3.38448304   0.76215955]
 [ 16.21181324 -10.78931755  -0.2870618 ]
 [  3.84156067   8.448928     0.56334086]
 [ 15.87973653  -0.1018432   -1.24818735]
 [  0.69173569   4.33845887  -0.11756393]
 [  5.99252814  -9.26801967  -1.15644864]
 [-13.00799405   4.91431839   1.16260885]
 [ -5.55145301   1.7479005   -2.80424998]
 [  5.44247276  -5.48477484   0.48696545]
 [ -7.95719496   0.80218215  -0.04906952]
 [ -0.67076025  -6.08928916   1.25486264]
 [ -0.4312163    1.81151802   0.68236867]
 [ -2.50757979   7.79374866   0.70397545]
 [ -4.65523528   7.9756312   -0.46644912]
 [ 23.51914958  -3.26915492  -0.43334213]
 [ 17.39169082  -5.79685593  -0.16355244]
 [-21.99578986   7.83123773  -1.45754971]
 [-16.08178889  -9.23748228   1.81834199]
 [ -7.08086097   1.22832774  -0.5428211 ]
 [ -5.93294033 -10.87314559  -1.42558025]
 [-18.1445075   -8.01392006   0.3301058 ]
 [ -2.6109593    1.64739583   0.9739266 ]
```

```
[-15.64833777   -5.25127303   -0.05930765]
[ -4.2969673    -8.53637902    0.65459365]
[ -4.90669916  -11.37183107    0.15695123]
[-11.49704962    8.81871243   -0.71464218]]
```

Listing 3: first three principal components of the given data

based on the pca values it seems that they are sufficient to describe most of the variation in the dataset.

3.2 b

in this section i used scipy library for testing normality of data

```
reduced_data = pca(data , 6)

#testing normality with different PCAs
for i in range(6):
    print('PCA num: ' ,i+1 , '----',stats.kstest(reduced_data[: , i], 'norm'))
```

Listing 4: testing normality of data with different PCAs

```
PCA num:  1 ---- KstestResult(statistic=0.5272552766410683, pvalue=2.6588368553952183
    e-08)
PCA num:  2 ---- KstestResult(statistic=0.45026164389662604, pvalue=4.52986908971393e
    -06)
PCA num:  3 ---- KstestResult(statistic=0.11110902105959519, pvalue
    =0.8527367641464626)
PCA num:  4 ---- KstestResult(statistic=0.20589309188960686, pvalue
    =0.13612676348380753)
PCA num:  5 ---- KstestResult(statistic=0.3456856468354019, pvalue
    =0.0010470771123993342)
PCA num:  6 ---- KstestResult(statistic=0.4252427561605906, pvalue=1.9402352972195527
    e-05)
```

Listing 5: output for testing normality of data

The D statistic is the absolute max distance (supremum) between the CDFs of the two samples. The closer this number is to 0 the more likely it is that the two samples were drawn from the same distribution. You reject the null hypothesis that the two samples were drawn from the same distribution if the p-value is less than your significance level.we consider this level as 0.05 so if p value is more than 0.05 then we can say the distribution is normal. based on these values we can say that the third and fourth PCAs from data are likely to have normal distribution because they have p values greater than 0.05 and smallest D statistics.

3.3 c

```
#reading data
df = pd.read_csv('doughs.dat' , sep = ' ')
data = df.drop(['Restaurant'] , axis = 1).to_numpy()

#adding a column to dataset to check if sample is from naples or not
df['Naples'] = df.apply(lambda x: 0 if x['Restaurant'] in [1,2,3,4] else 1 , axis =
    1)

reduced_data = pca(data , 3)
print(reduced_data)
#plotting 3d scatter plot for first 3 PCAs
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
scatter = ax.scatter(reduced_data[: , 0], reduced_data[: , 1], reduced_data[: , 2],
    marker='o' , c=df['Naples'], cmap=plt.cm.Set1,edgecolor='k')
plt.xlabel('first pca')
plt.ylabel('sec pca')
ax.set_zlabel('third pca')
handles = scatter.legend_elements()[0]
labels = list(df['Naples'].unique())
```

```
19  legend1 = ax.legend(handles, labels, title="Naples")
20  plt.savefig('3c-3D.png')
21  plt.clf()
22
23  #plotting scatter plots for different pairs of first 3 PCAs
24  for i , j in [[0,1],[0,2],[1,2]]:
25      fig, ax = plt.subplots()
26      scatter = ax.scatter(reduced_data[: , i] ,reduced_data[: , j] ,c=df['Naples'],
        cmap=plt.cm.Set1,edgecolor='k')
27      handles = scatter.legend_elements()[0]
28      labels = list(df['Naples'].unique())
29      legend1 = ax.legend(handles, labels,loc="upper left", title="Naples")
30      plt.xlabel('pca num: '+str(i+1))
31      plt.ylabel('pca num: '+str(j+1))
32      ax.add_artist(legend1)
33      plt.savefig('3c-2D' + str(i+1) + str(j+1)+'png')
34      plt.clf()
```

Listing 6: plotting different scatter plots

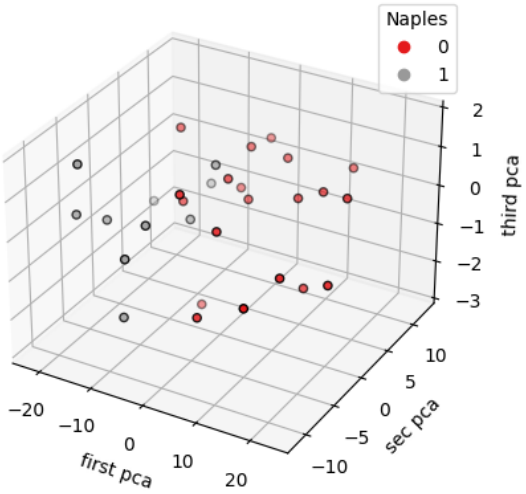

Figure 12: first 3 PCAs

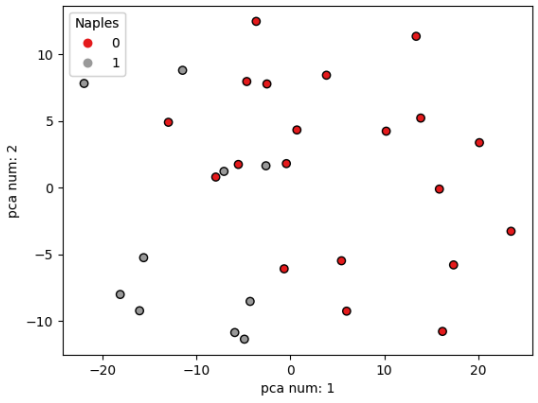

Figure 13: first 2 PCAs

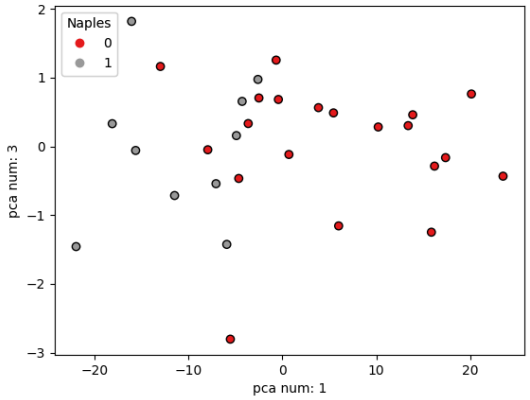

Figure 14: first and third PCAs

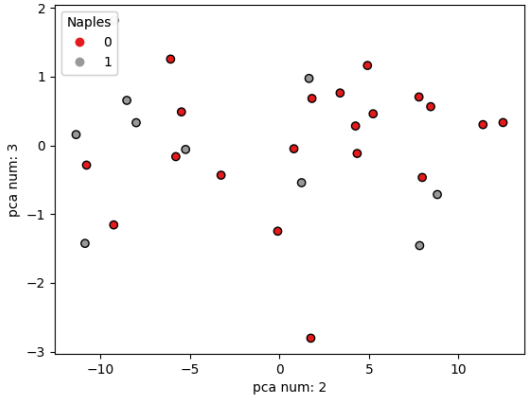

Figure 15: second and third PCAs

in 3D plot we can see that these three PCAs are enough to discriminate between doughs from Naples and doughs from other areas.

4 Interpretation of 1000 Genomes Project Results

4.1 a

we have 10101 features so we will have 10101 eigen values for scatter matrix and we will choose K of greatest eigen values and eigen vectors corresponding to them which K is less than or equal to 10101 so each vector would have K dimensions.

4.2 b

```
#reading dataset
df = pd.read_csv('1KGP.txt', delim_whitespace=True , header = None)
df[2] = df[2].astype("category")

#computing Y matrix
Y = df.drop([0,1,2] , axis = 1)
Y = Y.apply(lambda x: x!=x.mode()[0]).to_numpy()
Y = Y.astype('int')
#changing mean to zero
Y = Y - Y.mean(axis = 0).reshape(10101,1).T
```

```
11
12 #performing pca on Y
13 pca = PCA(n_components=2)
14 Y_reduced = pca.fit_transform(Y)
15
16 #plotting data based on 2 first PCAs and population
17 fig, ax = plt.subplots()
18 scatter = ax.scatter(Y_reduced[:, 0], Y_reduced[:, 1] ,c=df[2].cat.codes, cmap=plt.cm
       .Set1,edgecolor='k')
19 handles = scatter.legend_elements()[0]
20 labels = list(df[2].cat.categories)
21 legend1 = ax.legend(handles, labels,loc="lower right", title="population")
22 plt.xlabel('first pca')
23 plt.ylabel('second pca')
24 ax.add_artist(legend1)
25 plt.savefig('4b.png')
```

Listing 7: plotting data based on 2 first PCAs and population

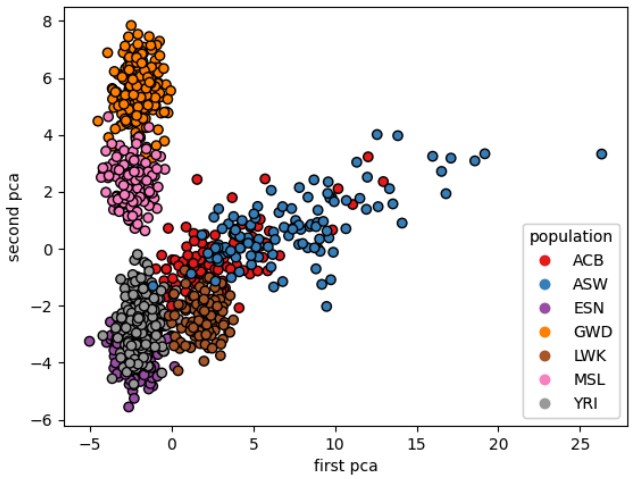

Figure 16: first 2 PCAs for data

4.3 c

we can see that these components are able to separate samples based on their locations. the ESN and YRI populations are in the same area because they are both samples from nigeria.

4.4 d

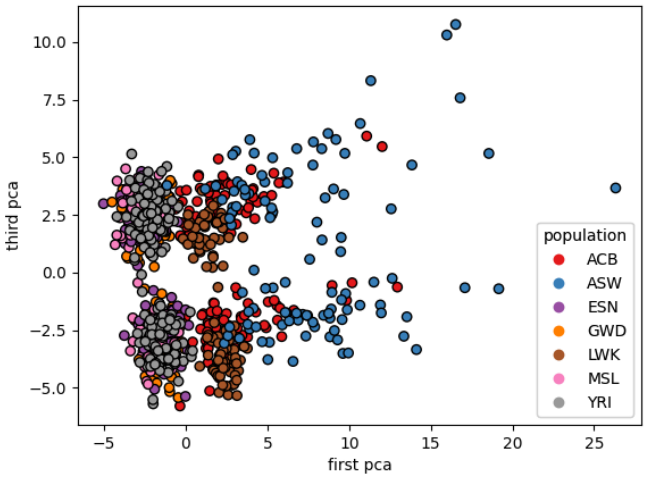

Figure 17: first and third PCAs

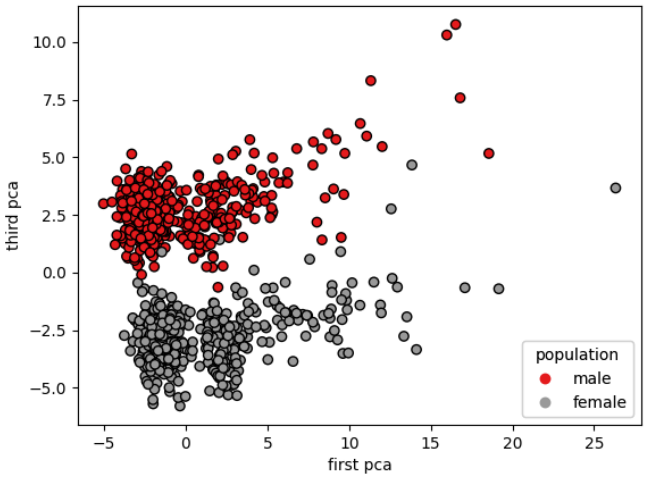

Figure 18: first and third PCAs

we can see that these components can separate clusters based on gender.

4.5 e

based on the plots in last section we can see that third PCA captures the gender features.

4.6 f

```
#computing Y matrix
Y = df.drop([0,1,2] , axis = 1)
Y = Y.apply(lambda x: x!=x.mode()[0]).to_numpy()
Y = Y.astype('int')
#changing mean to zero
Y = Y - Y.mean(axis = 0).reshape(10101,1).T

```

```
#performing pca on Y
pca = PCA(n_components=3)
Y_reduced = pca.fit_transform(Y)

#plotting nucleobase index versus the absolute value of the third principal component
fig, ax = plt.subplots()
scatter = ax.scatter(range(10101) , np.abs(pca.components_[2]) )
plt.xlabel('index')
plt.ylabel('abs of third pca')
plt.savefig('4f')
```

Listing 8: plotting nucleobase index versus the absolute value of the third principal component

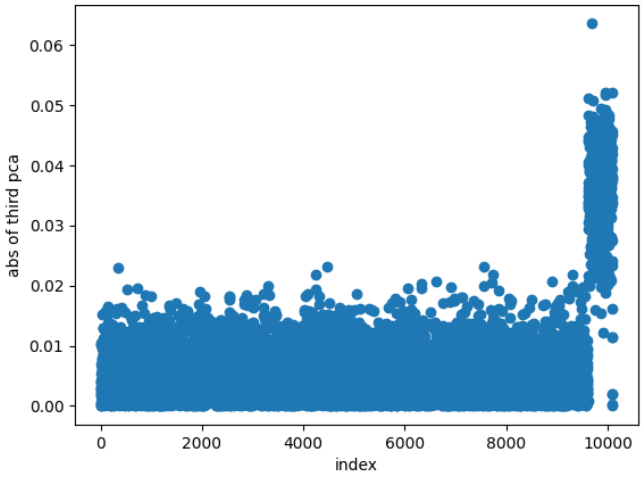

Figure 19: nucleobase index versus the absolute value of the third principal component

in part D we saw that third PCA can separate samples based on genders. in this section we can see third PCA absolute values are greater for last nucleobases. a possible explanation is maybe theses last nucleobase and last chromosomes are different for men and women.

5 How Twitter Reacts to a Crisis?

5.1 a

some functions that i implemented are these with their descriptions:

```
def jaccard_dist(a , b):
    """calculate jaccard dist between 2 sets

    Args:
        a ([set])
        b ([set])

    Returns:
        [int]: [jaccard dist between set a and set b]
    """
    jaccard_index = len(a.intersection(b)) / len(a.union(b))
    return 1 - jaccard_index

def assign_cluster(data , centroids):
    """for point data calculate which centroid it belongs to

    Args:
        data ([set]): [set of words of a tweet]
        centroids ([dict]): [centriods id's and their set of words]

```

```
    Returns:
        [int]: [it's the id of centroid which data belongs to]
    """
    min_dist = 1
    for key in centroids.keys():
        dist = jaccard_dist(data , centroids[key])
        if dist < min_dist:
            min_dist = dist
            assign = key
    return assign

def df_to_dict(df):
    """ a function for changing data to a dict with this format:
        every key is an id of a tweet and for every key we have
        a set which it has unique words of it's tweet

    Args:
        df ([dataframe]): [ids and tweets]

    Returns:
        [dict]
    """
    df.set_index("id", drop=True, inplace=True)
    df = df.to_dict(orient="index")
    for i in df.keys():
        df[i] = set(df[i]['text'].split())
    return df

def new_centroid(cluster):
    """for a cluster calculates it's centriod

    Args:
        cluster ([dict]): [it's a dict of id's as keys and their set of words of
    corresponding tweet
                           which these id's belongs to a single cluster]

    Returns:
        [int]: [new centroid id]
    """
    min_dist = 10**4
    for i in cluster.keys():
        dist = 0
        for j in cluster.keys():
            dist+=jaccard_dist(cluster[i] , cluster[j])
        if dist < min_dist:
            min_dist = dist
            centroid = i
    return centroid
```

Listing 9: usefull functions

```
#reading data and centroids
df = pd.read_json('tweets.json', lines=True)
data = df[['text' , 'id']]
data = df_to_dict(data)

centroids = pd.read_csv('initial_centroids.txt' , header=None)[0]
centroids = list(centroids)
```

Listing 10: reading data

and clustering steps are these:

```
converge = False
while(not converge):
    result = {}
    #assign each tweet to it's nearest centroid and saving them in result
    for i in data.keys():
        cluster = assign_cluster(data[i] , {centroid:data[centroid] for centroid in
    centroids})

```

```
            if cluster not in result.keys():
                result[cluster] = []

            result[cluster].append(i)

        centroids_new = []

        #calculating new centroid for each cluster
        for i in result.keys():
            cluster_ids = result[i]
            cluster_ids.append(i)

            centroids_new.append(new_centroid({j:data[j] for j in cluster_ids}))

        #checking converge
        if set(centroids_new) == set(centroids):
            converge = True
        centroids = centroids_new
```

Listing 11: clustring

and in this section results with wanted format will save on a file with name result-a.txt

```
with open("result_a.txt", "w") as file:
    for i in result.keys():
        file.write(str(i) + ' : ')
        for j in result[i]:
            file.write(str(j) + ' ')
        file.write('\n')
```

Listing 12: saving result in a file


323906397618196483 : 323906397600791488 323906397618196483 323906397735641088 323906397853073410 323906397962121216 323906398012461057 323906398230544385 323906398314438656 323906398352195585 323906398826164225
323906398999332289 323906399149199248 323906399295926273 323906399300100096 323906656318676993 323907087551836160 323907771256938496 323908455545049088 323908795254312962 323908795396943872 323906397618196483
323913403460636673 : 323906398423490561 323906482489921537 323906484629032960 323911097818173440 323913403460636673 323913403951357953 323913746231750656 323915624835342337 323916814490939393 323917413789868034
323918009901150208 323919716039155712 323921167381577728 323922279182520320 323924155391164416 323924496144797697 323924665074581504 323925432548335616 323927822072684544 323928504523694080 323929272685318147
323930212121317377 323932515599515649 323933624816787456 323944113764438016 323946504752603136 323950257832603648 323913403460636673
323906398176030720 : 323906398176030720 323906567294562306 323907258301939713 323910330315075584 323910330457669633 323955716392112128 323963901769297921 324226045052071936 323932094190439874 323906398176030720
323906485249789952 : 323906485804653360 323906485249789952 323911610236293120 323915009567697409 323916051614138368 323920146454425600 323921510282702848 323923559799996417 323925264352546816 323906485249789952
323906651209994241 : 323906650987692034 323906651209994241 323906653651079168 323906657333682176 323906651209994241
323909308188344320 : 323909308188344320 324229792834674689 323909308188344320
324070589214117888 : 324002623571255296 324070589214117888 324125626284007424 324132606608306176 324138055772561408 324154782782742529 324160230760005632 324179340675915779 324188481918230528 324227779530985472
324460055724433408 324534627039588355 324070589214117888
324038827578626048 : 324038827578626048 324067437886713856 324210976629067776 325063081929629696 324038827578626048
324117950774775809 : 324117950774775809 324117952695771137 324117950774775809
324375472681148416 : 324219503401644033 324358798179459073 324366333829586945 324375472681148416 324376801382133761 324419757661097984 324375472681148416
324427314060484608 : 324320247018573824 324327405743382529 324327410344542208 324423829088784384 324425542629732352 324427314060484608 324427596253253633 324427930631544832 324428255794978816 324427314060484608
324348252763873283 : 324346553835868161 324348252763873283 324368024561909760 324351038741901865 324326736198238763 324334932112843721 324310958418232934 324354128398432734 324337983412847463 324373063433847463
324382984012350293 324326985423982395 324348252763873283
324372750330363904 : 324372750330363904 324380629779247106 324418112680823616 324458862537228289 324470253339959298 324372750330363904
324408861563944960 : 324408472441585664 324408861563944960 324409861041438720 324409871061618689 324432383115935744 324437436077854721 324438127102013440 324439166496690176 324439832715747328 324440178842296321
324441529030701057 324441544377634816 324445289912074240 324445292130877440 324446653635522560 324452116863258624 324408861563944960
324422817565257728 : 324422817565257728 324423146381901825 324423157106749441 324423160525103105 324423494098120705 324423506957852672 324422817565257728
324448013999304704 : 324448013999304704 324448359932887040 324448711860158464 324451075501457408 324452781371060224 324448013999304704
324785129241341952 : 324785120085176320 324785122073251840 324785129241341952 324785131569168384 324787508997144576 324790912431583232 324799030859280384 324785129241341952
325059351209443329 : 325059351209443329 325059374814994434 325059598488082953 325059598511899648 325059967044452608 325060608246362816 325061078994284544 325063478140346370 325059351209443329
325060115408730931 2 : 325060138891350016 325060145082163200 325060154087309312 325060324992643072 325060826346184707 325060154087309312
325171606253080577 : 325162944931438592 325163043233337344 325165164561330176 325166486215856129 325170591164403712 325170934115864576 325171288526163968 325171606253080577 325172960413503489 325173634563964929
325175391654408192 325176378595102720 325178436660375552 325171606253080577
325253327048822784 : 325253327048822784 325253327640223744 325253669408874496 325253670746849280 325253670780416000 325254353877352448 325253327048822784
325335253487063041 : 325329802032721920 325335253487063041 325337623910559745 325337623785730752 1 325337640322871296 325338650076405761 325342218942554112 325350010961141760 325586012854185985 325335253487063041
325409910588317696 : 325409910588317696 325409910642835456 325409910714138625 325409910827401217 325409910869327872 325409910886117377 325409927193567232 325409992121389058 325409996290527232 325409996529602560
325409996726743041 325409996764479488 325409996965810176 325410254697414658 325410327036571649 325410327183364096 325410327206145409 325410327296614400 325409910588317696
3257028090297643008 : 325701932083716096 325701934273134594 325701941478973440 325702031912349696 325702115865538560 325702269288992768 325702271591665664 325702273684627456 325702285621612545 325702287978795009
325702294287048705 325702294928764929 325702363174285312 325702375430057984 325702377279729664 325702626169729024 325702632603803648 325702633174204416 325702635405574145 325702711729348608 325702890297643008
325702936036524032 325702950859198464 325702955896553472 325702956185948160 325702958786441216 326228987233841152 325702890297643008
325946283368013824 : 325917940564250624 325927476560032385 325946283368013824 325946633986641920 325946659739668480 325946918842818560 325947086145212416 325947301745029121 325947340777222145 325947343281205248
325946283368013824
```

Figure 20: result

## 5.2 b

in this section i used kmeans++ idea for initialization seeds , and the function for this work and it's description is shown here:

```
def initial_seeds(data , k):
    """this function works based on initialization of kmeans++
        The first seed is randomly selected from the set of tweets.
        Then, for each tweet t, the distance between t and the nearest seed that has already
        been chosen is computed. Another seed is then chosen with probability of
        its distance squared among the sum of all distances squared.
        These steps are computed until k seeds have been selected.

    Args:
        data ([dict]): [a dict with tweet id's as keys and their words as values]
        k ([int]): [number of centroids that we want]
```
```

```
12
13      Returns:
14          [list]: [id's of tweets that the function choose to be centroid]
15      """
16      centroids = []
17      centroids.append(np.random.choice(list(data.keys())))
18
19      while(len(centroids) < k):
20          dists_from_near_seed = {}
21          probs = {}
22          for key in data.keys():
23              dists_from_near_seed[key] = min([jaccard_dist(data[key] , data[centroid])
        for centroid in centroids])
24          for key in data.keys():
25              probs[key] = dists_from_near_seed[key]**2 / (sum([dists_from_near_seed[i
        ]**2 for i in dists_from_near_seed.keys()]))
26
27          centroids.append(np.random.choice(a = list(probs.keys()) , p = [probs[key]
        for key in probs.keys()]))
28      return centroids
```

Listing 13: initilizaliza

the result of this section is saved in a file name result-b.txt

6 A Tribute to Stephen Hawking

6.1 a

```
1  #reading data
2  data = pd.read_csv('ALS_train.csv')
3  data = data.set_index('ID')
4
5  #covariance matrix of data
6  cov = data.corr()
7  #selecting 10 columns with biggest value of covariance with ALSFRS_slope
8  cov = cov['ALSFRS_slope'].abs()
9  cov = cov.sort_values()
10 columns = cov[-11:].index
11 data = data[columns]
```

Listing 14: selecting 10 features

```
1  #plotting heatmap of cov matrix
2  sns_plot = sns.heatmap(data.corr())
3  sns_plot.get_figure().savefig("6a1.png")
4
5  #plotting data in ALSFRS_slope vs other featurs spaces
6  fig, axs = plt.subplots(2, 5 , figsize=(12, 5))
7  temp = 0
8  for i in range(2):
9      for j in range(5):
10         axs[i,j].scatter(data['ALSFRS_slope'], data[columns[temp]],marker='.' )
11         axs[i,j].set_title(columns[temp])
12         temp+=1
13
14 for j in range(5,10):
15     axs.flat[j].set( xlabel='ALSFRS_slope')
16
17 for ax in fig.get_axes():
18     ax.label_outer()
19
20 plt.savefig('6a2')
```

Listing 15: visualization

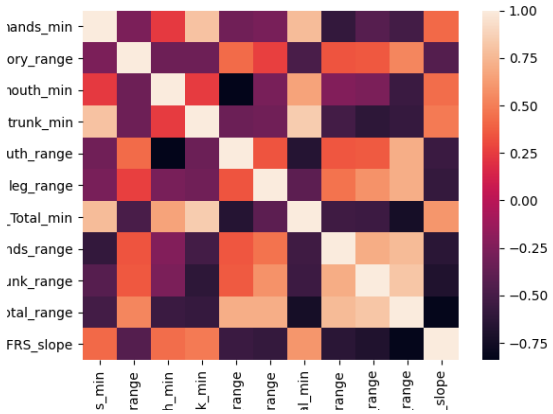

Figure 21: heatmap for covariance matrix of new data, the last column is for ALSFRS_slope column

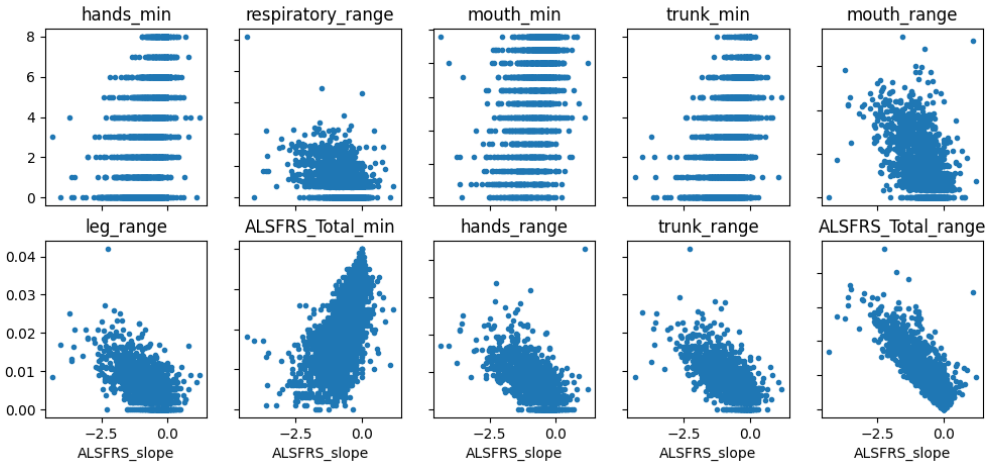

Figure 22: plotting data in different spaces with ALSFRS_slope as X axis and other features as Y axis

6.2 b

for selecting K i used elbow method with SSE criteria and silhouette method.

```
train_data = data.drop('ALSFRS_slope' , axis = 1)

#elbow method
sse = {}
for k in range(1, 15):
    kmeans = KMeans(n_clusters=k,init = 'random').fit(train_data)
    sse[k] = kmeans.inertia_ # Inertia: Sum of distances of samples to their closest
    cluster center
plt.figure()
plt.plot(list(sse.keys()), list(sse.values()))
plt.xlabel("Number of cluster")
plt.ylabel("SSE")
```

Listing 16: elbow method

```
sil = []
# dissimilarity would not be defined for a single cluster, thus, minimum number of
    clusters should be 2
for k in range(2, 15):
  kmeans = KMeans(n_clusters = k,init = 'random').fit(train_data)
```

```
    sil.append(silhouette_score(train_data, kmeans.labels_, metric = 'euclidean'))

plt.plot(range(2,15) , sil)
plt.xlabel("Number of cluster")
plt.ylabel("silhouette score")
```

Listing 17: silhouette method

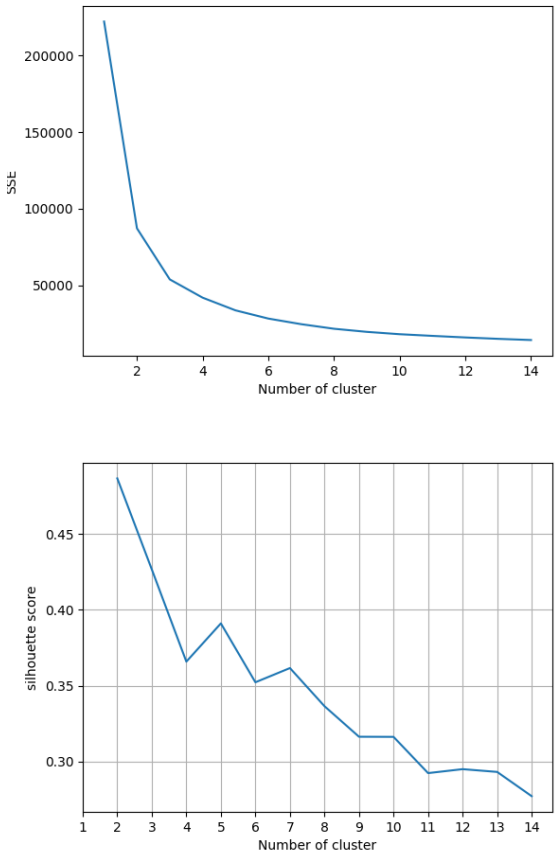

Figure 23: elbow and silhouette method plots for selecting K

based on elbow method it seems in K = 2 and K = 3 we have elbow. based on silhouette method the optimal K is 2.. i choose 3 for K.

6.3 c

```
#reading data
data = pd.read_csv('ALS_train.csv')
data = data.set_index('ID')

#covariance matrix of data
cov = data.corr()
#selecting 10 columns with biggest value of covariance with ALSFRS_slope
cov = cov['ALSFRS_slope'].abs()
cov = cov.sort_values()
columns = cov[-11:].index
data = data[columns]
train_data = data.drop('ALSFRS_slope' , axis = 1)

kmeans = KMeans(n_clusters = 3,init = 'random' ).fit(train_data)

print('centers of clusters:')
```

```
print(kmeans.cluster_centers_)
print()
print('silhouette score: ' , silhouette_score(train_data, kmeans.labels_, metric = '
    euclidean'))

#calculating number of samples in each cluster
cluster1_num = np.count_nonzero(kmeans.labels_ == 0)
cluster2_num = np.count_nonzero(kmeans.labels_ == 1)
cluster3_num = np.count_nonzero(kmeans.labels_ == 2)

#plotting bar chart
fig, ax = plt.subplots()
plt.bar([0,1 , 2], [cluster1_num , cluster2_num , cluster3_num] , width = 0.3)
plt.xticks([0,1,2], ('cluster 0', 'cluster 1' , 'cluster 3'))
plt.ylabel('number of samples')
plt.savefig('6c')
```

Listing 18: Evaluating model performance

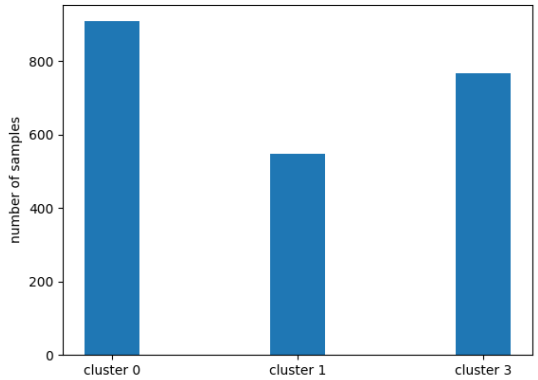

Figure 24: number of samples in each cluster

centers of clusters: [[2.32857143e+00 2.57275631e-03 8.22967033e+00 2.31868132e+00 5.85175584e-03 6.49584660e-03 1.86230769e+01 7.92365925e-03 7.89464384e-03 2.71946639e-02] [7.57299270e-01 4.14316504e-03 3.71532847e+00 6.66058394e-01 1.32495969e-02 8.34827940e-03 8.69160584e+00 9.64347110e-03 1.00976663e-02 4.23871766e-02] [5.54248366e+00 1.27555133e-03 1.01516340e+01 5.35424837e+00 2.71081487e-03 4.20080686e-03 2.93816993e+01 3.66684084e-03 4.11118900e-03 1.29428526e-02]]

silhouette score: 0.4266824551341931

6.4 d

```
kmeans = KMeans(n_clusters = 3,init = 'k-means++').fit(train_data)

#calculating number of samples in each cluster
cluster1_num = np.count_nonzero(kmeans.labels_ == 0)
cluster2_num = np.count_nonzero(kmeans.labels_ == 1)
cluster3_num = np.count_nonzero(kmeans.labels_ == 2)

#plotting bar chart
fig, ax = plt.subplots()
plt.bar([0,1 , 2], [cluster1_num , cluster2_num , cluster3_num] , width = 0.3)
plt.xticks([0,1,2], ('cluster 0', 'cluster 1' , 'cluster 3'))
plt.ylabel('number of samples')
plt.savefig('6d')
```

Listing 19: kmeans++

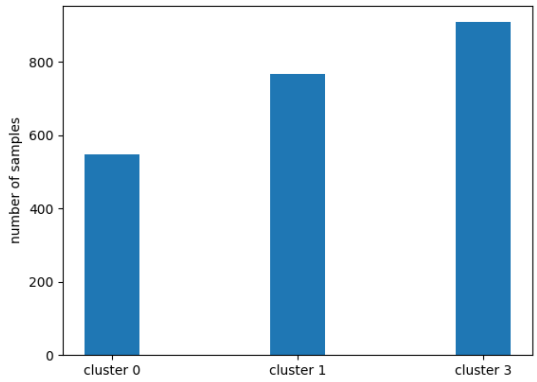

Figure 25: number of samples in each cluster for kmeans++

6.5 e

```
KMeans(n_clusters = 3,init = 'k-means++')
```

Listing 20: model with optimal parameters

6.6 f

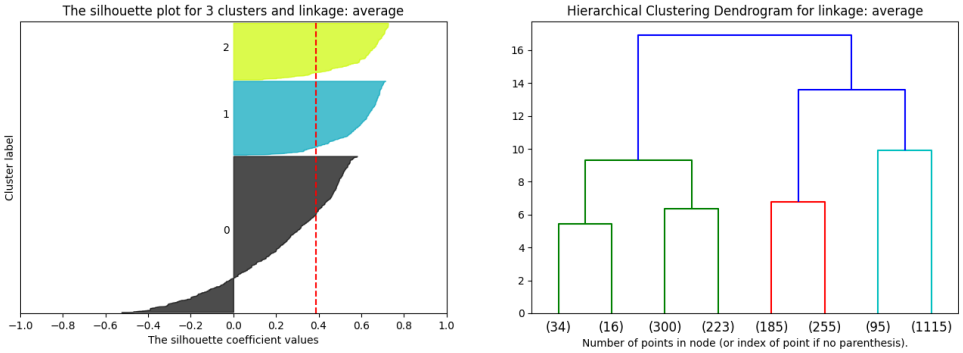

Figure 26: silhouette plot and dendrogram for average linkage

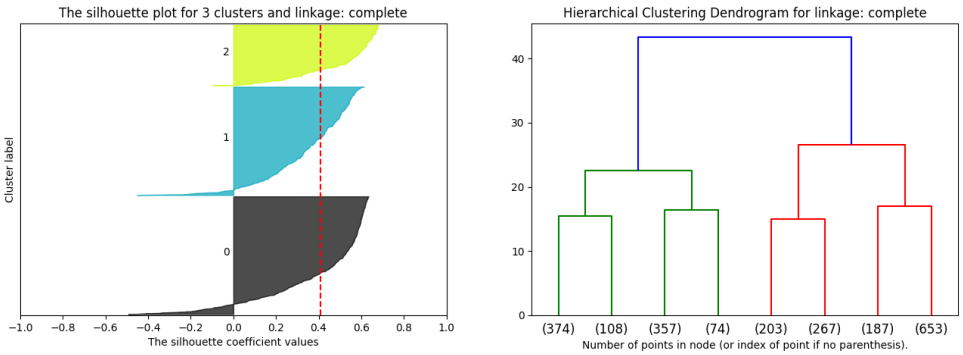

Figure 27: silhouette plot and dendrogram for complete linkage

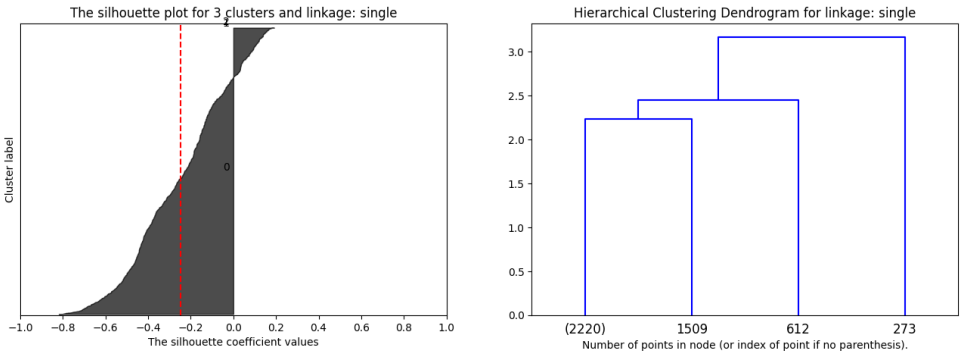

Figure 28: silhouette plot and dendrogram for single linkage

6.7 g

we can see that single linkage does not works good comparing to complete and average linkage.

7 Are You Into Fashion?

7.1 a

```
#reading data
images = idx2numpy.convert_from_file('t10k-images-idx3-ubyte')
labels = idx2numpy.convert_from_file('t10k-labels-idx1-ubyte')
labels = labels.reshape(10000,1)
images = images.reshape(10000,28*28)

#performing PCA
pca = PCA(n_components=20)
pca_reduced = pca.fit_transform(images)

print('eigenvalues: ' , pca.explained_variance_)
```

Listing 21: performing PCA

top 20 eigenvalues: eigenvalues: [1288319.52477778 779197.62253773 265730.43854769 218669.76933454 169257.23458071 152452.76424582 104674.41864859 83982.28146169 58343.40691272 57195.68410513 43687.9739565 40723.99085159 33555.72005452 28600.95680321 27416.98190461 25850.84607944 24624.94870779 23563.26086334 21252.81703952 19652.47616396]

```
fig, ax = plt.subplots()
scatter = ax.scatter(pca_reduced[:, 0], pca_reduced[:, 1] ,c=labels, cmap=plt.cm.
    tab10,edgecolor='k' , s = 30)
handles = scatter.legend_elements()[0]
labels_unique = np.unique(labels)
legend1 = ax.legend(handles, labels_unique,loc="upper right", title="category" ,
    bbox_to_anchor=(1.15, 1))
plt.xlabel('first pca')
plt.ylabel('second pca')
ax.add_artist(legend1)
plt.savefig('a1.png')
plt.clf()

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(pca_reduced[: , 0], pca_reduced[: , 1], pca_reduced[: , 2], marker='o' , c
    =labels, cmap=plt.cm.tab10,edgecolor='k'  ,s = 20)
handles = scatter.legend_elements()[0]
labels_unique = np.unique(labels)
legend1 = ax.legend(handles, labels_unique,loc="upper right", title="category" ,
    bbox_to_anchor=(1.35, 1))
plt.xlabel('first pca')
plt.ylabel('second pca')
```

```
20  ax.set_zlabel('third pca')
21  ax.add_artist(legend1)
22  plt.savefig('a2.png')
23  plt.clf()
```

Listing 22: plotting data on first 2 and 3 PCs

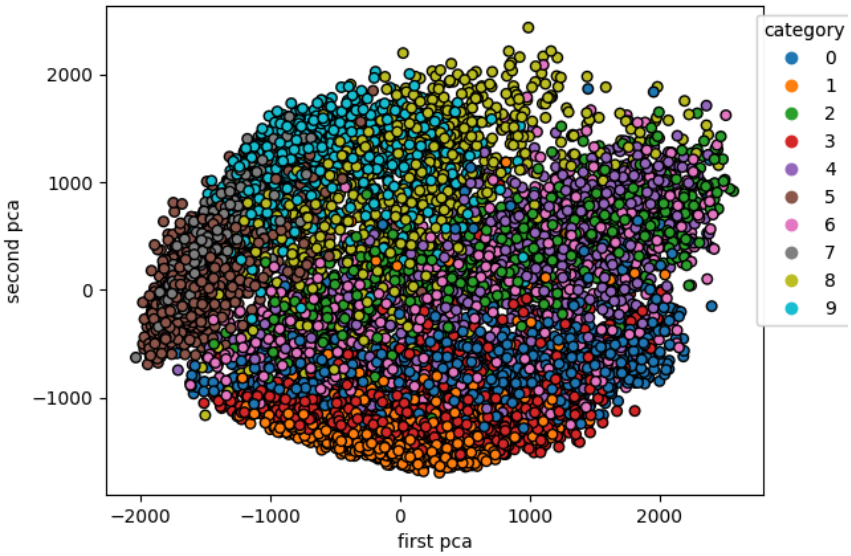

Figure 29: data projection on first 2 PCs

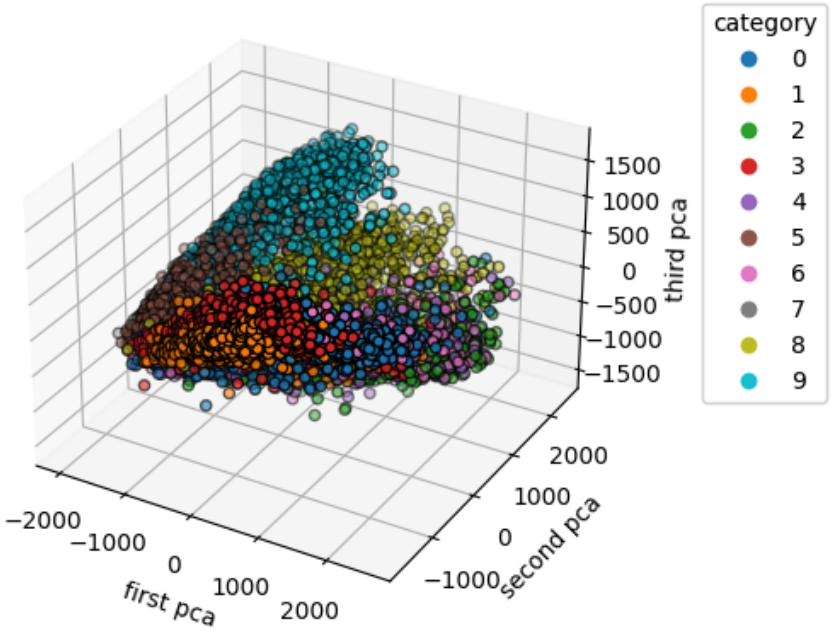

Figure 30: data projection on first 2 PCs

7.2 b

```
#performing LDA
lda = LinearDiscriminantAnalysis(n_components=2)
lda_reduced = lda.fit(images, np.ravel(labels)).transform(images)
```

Listing 23: applying LDA on data

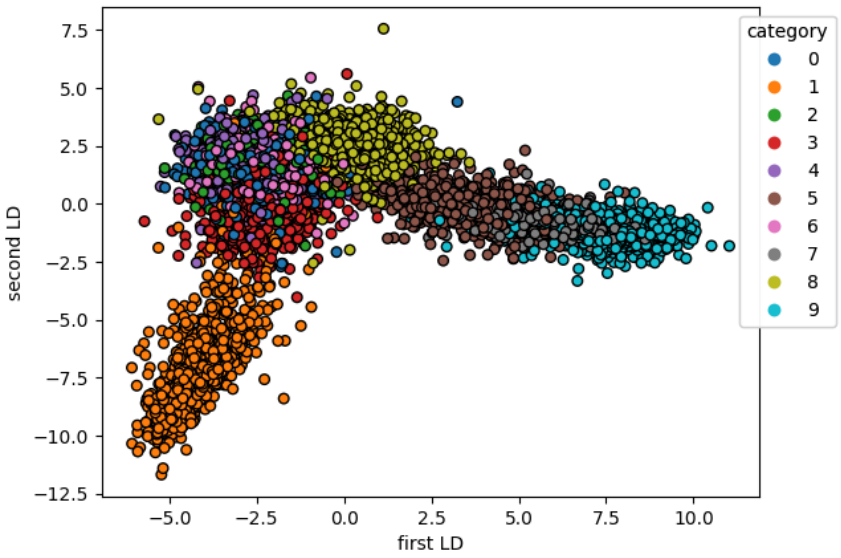

Figure 31: data projection on first 2 LDs

7.3 c

```
#performing kmeans and plotting clusters
for k in [4,7,10]:
    kmeans = KMeans(n_clusters=k,init = 'random').fit(pca_reduced)
    fig, ax = plt.subplots()
    scatter = ax.scatter(pca_reduced[:, 0], pca_reduced[:, 1] ,c=kmeans.labels_, cmap
    =plt.cm.tab10,edgecolor='k' , s = 30)
    handles = scatter.legend_elements()[0]
    plt.xlabel('first pca')
    plt.ylabel('second pca')
    plt.savefig('7c,K= '+str(k))
    plt.clf()
```

Listing 24: performing kmeans

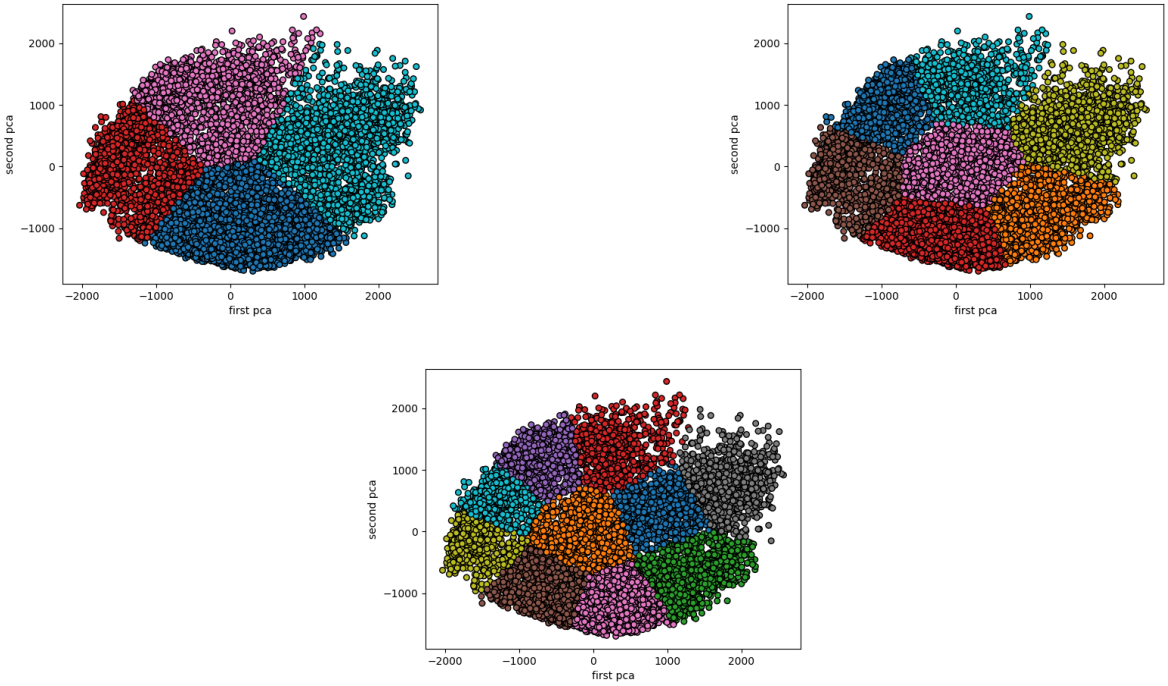

Figure 32: performing kmeans on data using first 2 PCs with different Ks

kmeans try to cluster data with linear boundaries so based on part a plot it cant cluster data using these 2 features in a good way because classes are not linearly separable in this feature space.

7.4 d

```
sets = [k_4sets , k_7sets , k_10sets]
#performing kmeans and plotting clusters
#a loop for different values of k
for set in sets:
    k = len(set)
    #computing mean of each set
    means = [pca_reduced[np.isin(labels , i)].mean(axis=0) for i in set]
    means = np.array(means)
    #performing kmeans
    kmeans = KMeans(n_clusters=k,init = means).fit(pca_reduced)
```

Listing 25: computing means and performing kmeans for different Ks

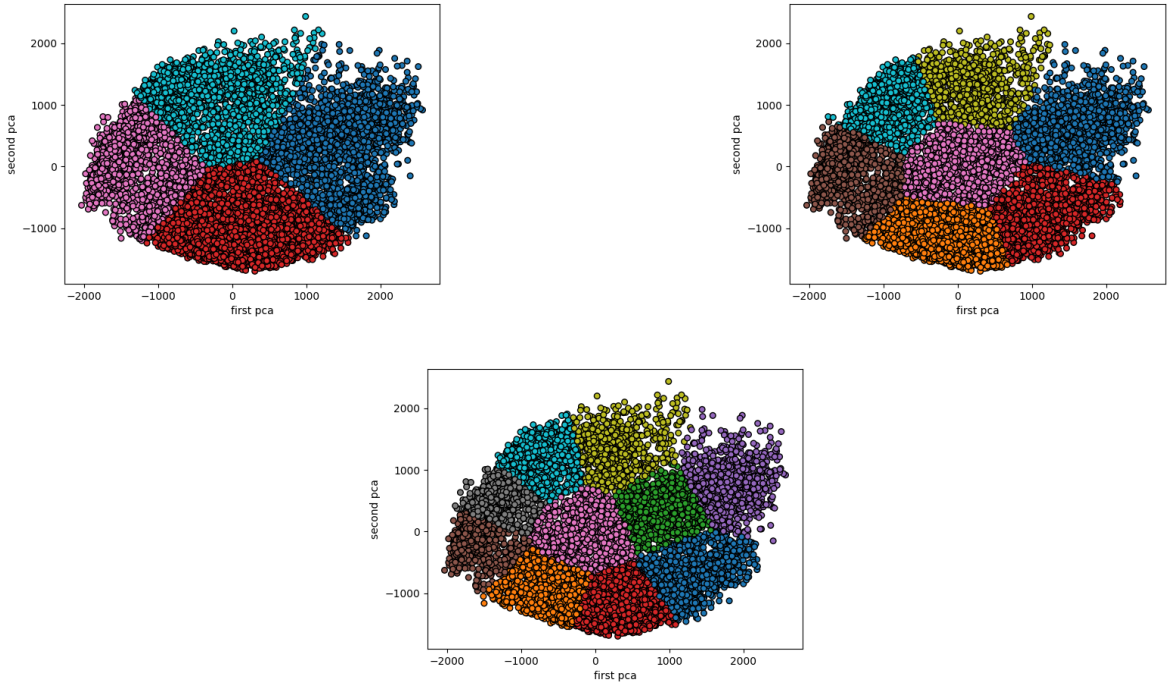

Figure 33: performing kmeans on data using first 2 PCs with different Ks and means as centroids

we can see with centroids as means the result dose not change from random centroids very much

7.5 e

185 first PCs are enough

```
#performing PCA
pca = PCA(n_components=200)
pca_reduced = pca.fit_transform(images)

#calculating min number of PCs for 95% information
var = pca.explained_variance_ratio_[0]
i = 1
while(var < .95):
    var += pca.explained_variance_ratio_[i]
    i+=1

print('enough numbers of PCs for capturnig 95% of information: ',i+1)
```

Listing 26: finding enough number of PCs for capturing 0.95 information from data

```
#performing pca with i component
pca = PCA(n_components=i+1)
pca_reduced = pca.fit_transform(images)
#reversing reduced data to images
inverse_images = pca.inverse_transform(pca_reduced)

#saving 3 random image and their reduced versions
fig, axs = plt.subplots(3, 2 , figsize=(9, 9))
for i in range(3):
    #select 10 indexes randomly from cluster i
    random_indxs = np.random.choice(range(10000))
    axs[i,0].imshow(images[random_indxs].reshape(28,28), cmap='gray', vmin=0, vmax
    =255 )
    axs[i,1].imshow(inverse_images[random_indxs].reshape(28,28), cmap='gray', vmin=0,
     vmax=255 )
for j in range(3):
    axs[j,0].set( xlabel='original images')
```

```
16      axs[j,1].set( xlabel='reduced images')
17  for ax in fig.get_axes():
18      ax.label_outer()
19  plt.savefig('7e.png')
```

Listing 27: performing inverse transform on reduced data and showing 3 random pics

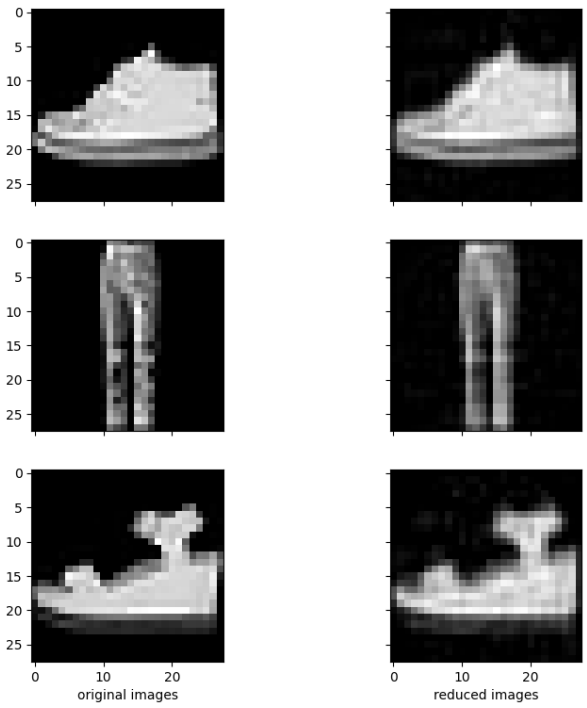

Figure 34: 3 random photos and their reduced versions

7.6 f

```
1  #performing PCA
2  pca = PCA(n_components=185)
3  pca_reduced = pca.fit_transform(images)
4  #performing kmeans
5  kmeans = KMeans(n_clusters=10,init = 'k-means++').fit(images)
6
7  #plotting 10 samples from each cluster randomly
8  fig, axs = plt.subplots(10, 10 , figsize=(14, 14))
9
10 for i in range(10):
11     #select 10 indexes randomly from cluster i
12     random_indxs = np.random.choice(np.arange(10000)[kmeans.labels_==i],10)
13
14     for j in range(10):
15         axs[i,j].imshow(images[random_indxs[j]].reshape(28,28), cmap='gray', vmin=0,
       vmax=255 )
16
17 for j in range(0,100,10):
18     axs.flat[j].set( ylabel='C'+str(j//10))
19
20 for ax in fig.get_axes():
21     ax.label_outer()
22
```

```
plt.savefig('7f.png')
```

Listing 28: performing kmeans on first 185 PCs

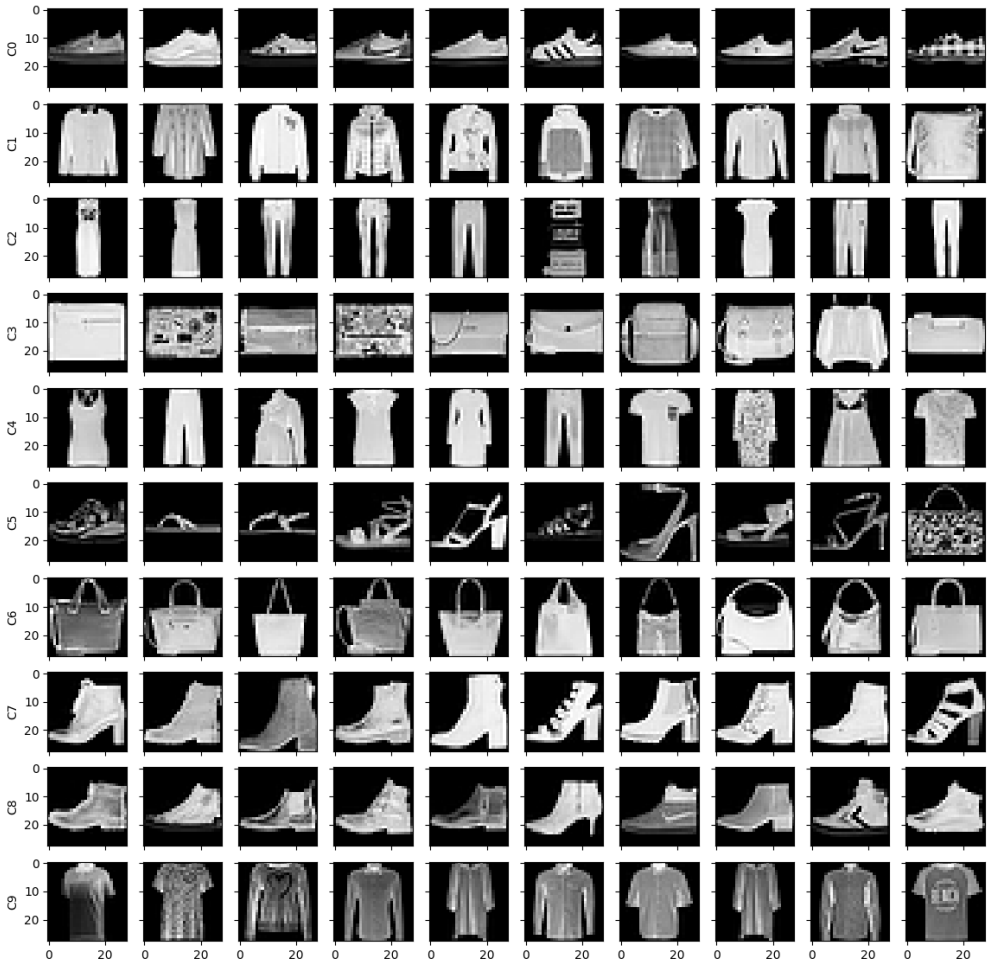

Figure 35: 10 random pics from each cluster

each row represent a cluster. we can see that kmeans on first 185 PCs can divide data in a meaningful way based on fashion categories and can cluster data with a good accuracy based on classes.

7.7 g

```
#performing PCA
pca = PCA(n_components=185)
pca_reduced = pca.fit_transform(images)

#performing kmeans
kmeans = KMeans(n_clusters=10,init = 'k-means++').fit(pca_reduced)

#caculating a dataframe that each row represent a cluster and each column
```

```
#reperesent a class and each cell tell us the percentage of class in cluster
percentage = {}
for i in range(10):
    percentage['cluster' + str(i)] = [int(np.count_nonzero(labels[kmeans.labels_ == i
    ]==k)*100 / np.count_nonzero(kmeans.labels_ == i)) for k in range(10)]
df = pd.DataFrame.from_dict(percentage,orient='index')

#plotting bor plot
ax = df.plot(kind='bar', figsize=(20,5), width=0.8, edgecolor=None)

for p in ax.patches:
    width = p.get_width()
    height = p.get_height()
    x, y = p.get_xy()
    if height >0:
        ax.annotate(f'{height}', (x + width/2, y + height*1.02), ha='center')

plt.legend(labels=df.columns , title = 'classes')
plt.savefig('7g')
```

Listing 29: drawing bar grapha

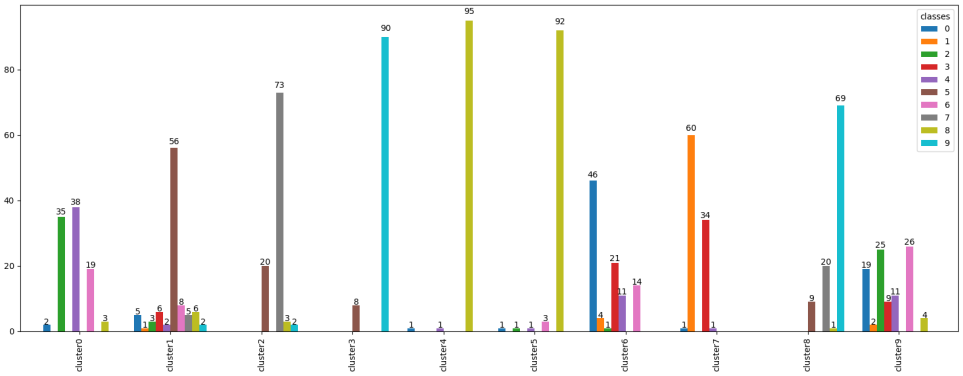

Figure 36: representing the distribution of the samples of different classes in each cluster
i have rounded percentages so they are not exact.we can see cluster 4 and 5 are like each other.and clusters 3 and 8 are like each other.

7.8 h

in this section i selected first 3 PCs as features for plotting.

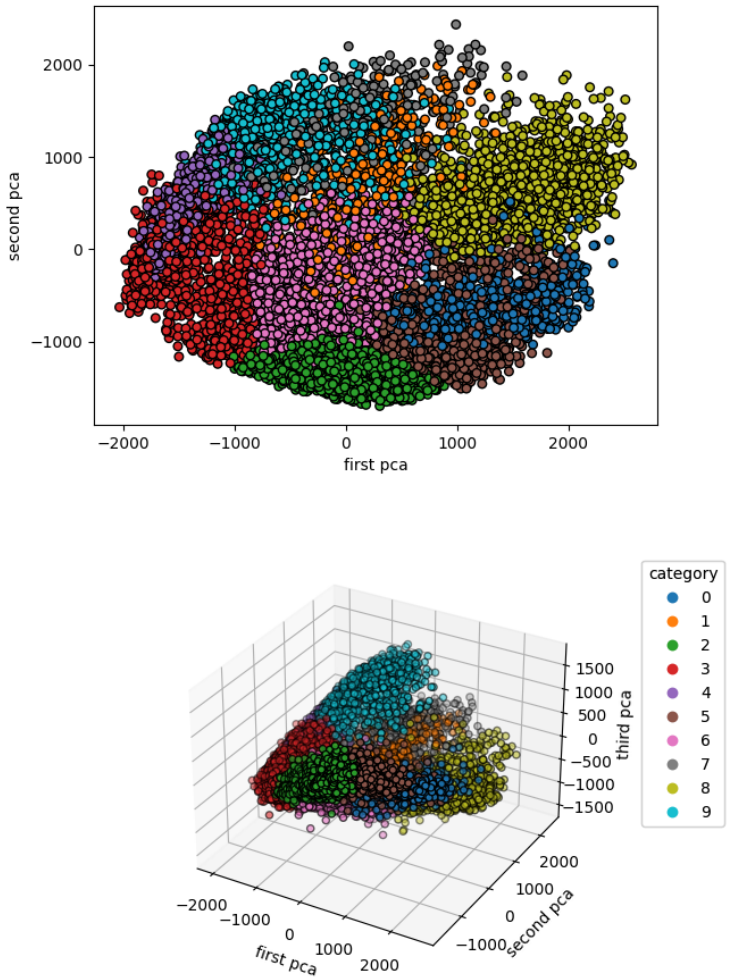

Figure 37: clusters represented on 2d and 3d space based on first 3 PCs for K = 10

8 Some Explanatory Questions

8.1 a

because it's based on eigen vectors of scatter matrix of data which has n * n dimension. and each eigen vector give us one of the PCs.so maximum number of PCs that we can have is n because it's the maximum number of eigen vectors of scatter matrix.

8.2 b

an image is a matrix of pixels represented by RGB color values. Thus, principal component analysis can be used to reduce the dimensions of the matrix (image) and project those new dimensions to reform the image that retains its qualities but is smaller.

8.3 c

yes, basic idea is to replace the euclidean distance computations by kernelized version.

$$\|\Phi(x_n) - \Phi(\mu_k)\|^2 = \|\Phi(x_n)\|^2 + \|\Phi(\mu_n)\|^2 - 2\Phi(x_n)^T\Phi(\mu_k) = K(x_n, x_n) + K(\mu_n, \mu_n) - 2K(x_n, \mu_n) \quad (1)$$

which K is kernel function and Phi is it's feature map.the advantage of this method is it works for nonlinear samples too.

8.4 d

it will decrease. when we increase K, points that their cluster changes will have smaller error because their new centroids are closer to them, otherwise the cluster which they belongs to would not change. so we would have smaller error and smaller variance.

8.5 e

when K is equal to n this happens where n is the number of samples, because each sample will become a centroid and each cluster is just a sample which the sample itself is centroid so the variance of each cluster becomes zero.