Assignment 4: PCA and Clustering (due midnight 2/24)

```
# imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Part 1: Parsing data

You're given a dataset of gene expression for 20 samples that might represent different biological situations.

- Read in the data as a matrix of n rows (number of samples) by m (number of genes measured for each sample). **Each sample (row) is a data point, and each gene (column) is a feature**.
- Normalize the data: calculate the mean expression for each gene, and rescale each individual measurement relative to that mean (giving the fold change relative to the mean).
- Then, log_2 transform the data.
- Zero mean the data - subtract each column's mean from each value in that column

Add code cells below this cell in order to parse the data.

```
gene_exp_table = pd.read_csv("gene_expression.csv",header =0,index_col=0).to_numpy()

#Get mean for each column
gene_means = np.mean(gene_exp_table,axis=0)

gene_exp_table_divide = np.divide(gene_exp_table,gene_means)

#gene_exp_table_divide = gene_exp_table_divide.float()

print(gene_exp_table_divide)

gene_exp_array_log = np.log2(gene_exp_table_divide)
```

```
[[0.11116726 1.7788608  2.36620139 ... 2.03278689 0.2006689  0.44585811]
 [2.30419404 0.38379531 0.15724914 ... 0.39344262 2.20735786 1.28723551]
 [2.62758969 0.39597929 0.10350576 ... 0.45901639 2.27424749 1.32319181]
 ...
 [0.13137948 1.48035334 2.21989998 ... 1.63934426 0.26755853 0.53049371]
 [1.81910056 0.41425525 0.16421587 ... 0.45901639 2.27424749 1.43714562]
 [0.76806468 1.01127018 0.91015401 ... 0.85245902 0.80267559 1.29110773]]
```

```
#Zero mean the data
#Get mean for each column
gene_means = gene_exp_array_log.mean(axis=0)

gene_exp_final = gene_exp_array_log - gene_means

gene_exp_final
```

```
array([[-2.34999555,  0.99011414,  1.77535796, ...,  1.22944536,
        -1.82090512, -1.02564461],
       [ 2.02346285, -1.2224305 , -2.13609091, ..., -1.13978845,
         1.6385265 ,  0.50397486],
       [ 2.21294065, -1.17734261, -2.73943194, ..., -0.91739603,
         1.68159522,  0.54372103],
       ...,
       [-2.10898745,  0.72510208,  1.68327977, ...,  0.91910524,
        -1.40586762, -0.77489363],
       [ 1.68242593, -1.11224758, -2.07354945, ..., -0.91739603,
         1.68159522,  0.66290508],
       [ 0.43850035,  0.17532901,  0.3969677 , ..., -0.02431123,
         0.17909488,  0.50830821]])
```

Part 2: PCA

We want to be able to distill the most important aspects, both for the purposes of visualization and de-noising. It may be the case that only the *d* most important axes of variation contribute significantly to our understanding of the data.

2.1: Plot the singular values of the matrix

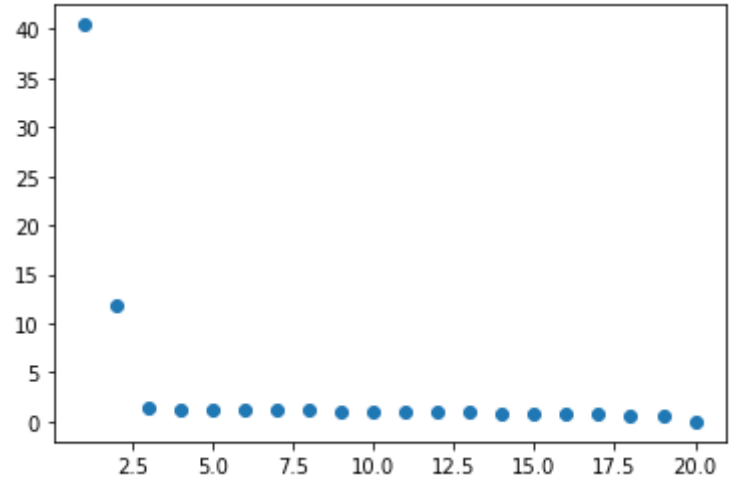
In the cell below, create a chart comparing the singular values of the data matrix. Your x-axis should range from 1 to *m* (total dimension of the vector space), and your y-axis should plot the singular values in order from greatest to least. Feel free to use `np.linalg.svd`.

```
import matplotlib.pyplot as plt
#another type of decomposition.
u, d, v = np.linalg.svd(gene_exp_final, full_matrices=True)

#u,d,v --> end up with three matricies
#plt.scatter

dim = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
plt.scatter(dim,d)
```

```
<matplotlib.collections.PathCollection at 0x7fdfe3ea8150>
```



2.2 Choosing *d*:

Based on the values you plotted, what should we set *d* to be for our purposes? Where's the cutoff in your judgement for the small set of useful features?

**Fill out d below by replacing None with a value. It should be an integer. **

```
#d is the number of dimesions in PCA
d = 2
```

2.3: PCA Derivation/Motivation

Let us assume that our data is drawn from a multivariate Gaussian distribution. Suppose we compute the singular value decomposition of our data matrix to be $X = U\Sigma V^\intercal$.

1) What are the major properties of the matrices U, V and Σ? (Describe them like, U is a ____ matrix) (what type of matrix is this?, look up singular value decomposition)

2) What is $X^\intercal X$ in terms of U, V and Σ? Make sure you simplify based on the properties you described in part 1. As a hint, your answer should be a product of three terms.(plug 1 into 2, simplify into another set of matricises, product of three terms)

3) We know that

$$\frac{1}{n} X^\intercal X$$

is the sample covariance matrix if we assume that each X_i is drawn from a Gaussian distribution. This means it describes the joint variance of the features. Given this Gaussian, how may we construct a *d* dimensional basis to project our data? In other words, which vectors from U, V, etc do we take to be our projection basis? Hint: These vectors represent the directions with the greatest variance.

(the middle matrix diagonal gives you the variance of each variable)

(Note 1: we're trying to pick a vector basis with the highest variance because it will give us the most predictive power. Imagine trying to project to a basis where every data point gets projected to a similar or low-varying area. How are you going to classify anything? High variance ensures points are likely to be spread apart and thus more easily classified or clustered into distinct categories.)

(Note 2: additionally, because our basis is orthonormal, we gain the advantage that the features we pick don't "interfere" with each other, and "describe" distinct aspects of the data.)

Answer in a text cell below, or scan and attach written answers to the submission pdf.

1) Σ. is a diagonal matrix, with each member on the diagonal decreasing in magnitude, singular values U is a unitary matrix, also orthogonal, left singular vectors $V^\intercal$ is a unitary matrix, also orthogonal, right singular vectors

2) $(X^\intercal X)$/n = $(V\Sigma U_\intercal)(U\Sigma V_\intercal)$ = $V(\Sigma)$/n$V_\intercal$

3) The first 2 vectors of V

2.4: Collect Basis Vectors

Based on your answer to question 4, collect a basis of vectors that you will be projecting the data onto. The basis should be of length *d* (decided upon in section 2.2).

From 2.1 and 2.3

Replace None below with the correct basis.

```
#first two columns from V
v_basis = v[0:2,:]
v_basis.shape
```

```
(2, 100)
```

Part 3: Projection

3.1: Projection function

Please fill in the function below, which projects a given vector and sends it to the space described by the input basis. Because the basis we're going to be using is orthonormal, we can use the formula described at

https://home.cpu.edu/~smccathern/PastCourses/F13/LinearAlgebra/LA5_1handout.pdf (Theorem

https://home.apu.edu/~smccathern/PastCourses/F12/LinearAlgebra/LA5_Handout.pdf (Theorem 7)

Formula: vector*transpose(basis) * = matrix multiplication (np.matmul)

```
def project(v, basis):
    #Takes a vector v and projects it into the space spanned by basis.
    v_basis = np.matmul(v,np.transpose(basis))

    return v_basis
```

3.2: Project & Plot

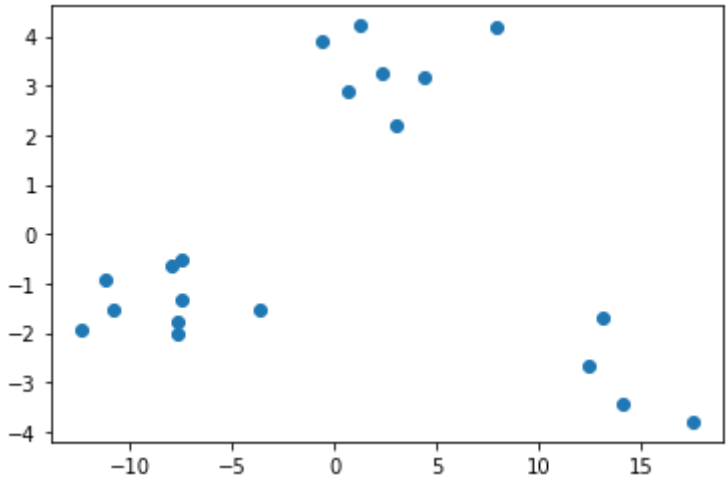
We've chosen the dimension of our projection. We've chosen the vectors that will give us the most descriptive projection subspace. Now we just need to project our data and plot it.

Please use the function in 3.1 and create a scatterplot of your projected data.

```
#project every column by the 2 basis vectors
gene_exp_array_projected = project(gene_exp_final,v_basis)

#Take the first two columnns
plt.scatter(gene_exp_array_projected[:,0],gene_exp_array_projected[:,1])
```

```
<matplotlib.collections.PathCollection at 0x7fdfe3f3f050>
```



Part 4: Clustering

Consider the scatterplot you've made. How many natural clusters do you think the data seems to form?

Replace None below.

```
#Based off the scatter plot
```

```
k = 3
```

We're going to be employing the k-means algorithm, a form of unsupervised learning. The algorithm will produce an assignment of each data point to one of *k* clusters (coloring your scatterplot *k* colors). A "cluster" is loosely defined by "the collection of points gathered around a center point (centroid)" The reason it's unsupervised is because we're not given any training data or labels, but we still have to infer a label for each data point.

4.1: Understanding k-means Psuedocode

Variables:

k: the number of clusters

$x_1, \ldots, x_n$: the data (vectors)

$l_1, \ldots, l_n$: the corresponding label of each datapoint (scalars, value in interval [1, k])

$c_1, \ldots, c_k$: the centroids of each cluster (vectors)

Algorithm:

> Initialize $c_1, \ldots, c_n$ randomly.
>
> while (centroids still move more than ϵ distance, for some small $\epsilon > 0$) related/is eps, if centroid hasnt changed positions then end run
>
> Assign each l_i label for each point x_i to the nearest centroid's index (if c_3 is the closest to x_i, set l_i to be 3)
>
> Update each centroid c_i to be the average of all the x_i points currently assigned to its cluster

You may notice this functions kind of like the EM algorithm in that we update the centroid (hidden state) and our labels for the data in a loop based on each other. In fact, one might say that k-means is a non-probabilistic ("hard") version of EM specifically for clustering.

4.2: Implement k-means

```
def kmeans(xs, k):

    eps = 1e-8

    # initialize centroids of clusters
    cs = np.random.choice([0], size=(1,2)) # list of centroids
```

```
    for i in range(k):
        # TODO
        # use np.random.uniform() to sample from uniform distribution
        # can use .min() and .max() to get the bounds for above
        rand_location = np.array([[np.random.uniform(xs[:,1].min(),xs[:,1].max()),np.
        cs = np.append(cs,rand_location,axis=0)

    cs_old = cs[1:,:]


    # labels
    # e.g ls = domain -> 0, 1, 2, ... k
    # initially set all the data points to the 0th cluster
    ls = [0 for i in range(len(xs))]

    # Init an empty cs for the first comparison
    # Make sure that in the first comparison you only assign labels
    cs_new = np.random.choice([0], size=(k,2))


    # main loop goes here
    #while difference between this cs and the last cs
    #have an old cs and a new cs to compare to !!!!!!

    count = 0
    while abs(np.mean(cs_new-cs_old)) > eps:  # TODO: replace True condition with con

        print(abs(np.mean(cs_new-cs_old)))

        if count > 0:
          #the new cs has been compared to and is now old
          cs_old = cs_new


        #Assign each  li  label for each point  xi  to the nearest centroid's index (i
        for i in range(len(xs)):
            # TODO: assign label
            distances = []

            #run through all possible centriods
            for j in range(k):
              a = xs[i,:]
              b = cs_old[j,:]

              distances.append(np.linalg.norm(a-b))
              min_value = min(distances)
              ls[i] = distances.index(min_value)

        if count == 0:
            count +=1

        # recalculate centroids
```

```
        for i in range(k):
            # TODO: recalculate centroid position
            # two steps: 1) get all the points that are in cluster i
            points_xy = np.random.choice([0], size=(1,2))

            point_indexs = [j for j, x in enumerate(ls) if x == i]

            for index in point_indexs:
                points_xy = np.append(points_xy,[xs[index,:]],axis=0)

            points_xy = points_xy[1:,:]

            #2) take mean over those points (use np.mean)
            cs_new[i] = np.mean(points_xy, axis=0)

    return ls
```

+ Code + Text

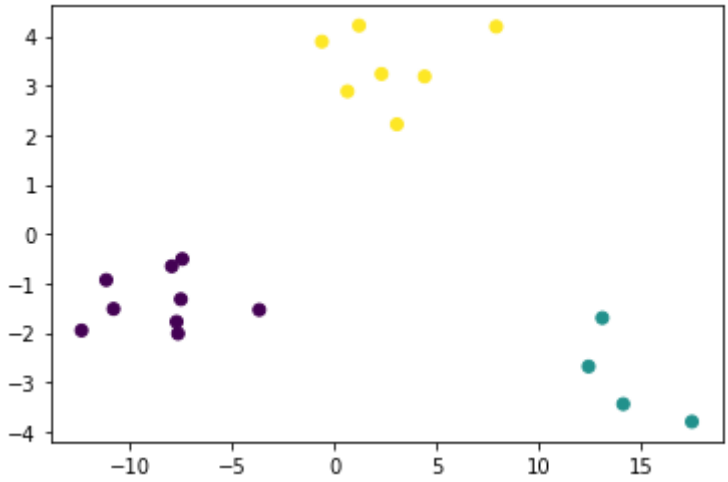
4.3 Plot projected data with color labels

Below, use the 'c' argument in `plt.scatter` and the list of labels you derived to color the scatter plot you created in 3.2.

```
xs = gene_exp_array_projected[:,0:2]

plt.scatter(gene_exp_array_projected[:,0],gene_exp_array_projected[:,1],c=kmeans(xs,k
```

```
0.7384487711856829
0.5948845621476505
<matplotlib.collections.PathCollection at 0x7fdfe3ae3150>
```



✓ 0s completed at 9:10 PM