Assignment 4: Clustering and PCA

In [0]:
```
# imports
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

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.
- Subtract mean from data matrix (zero-mean each column)

In [130]:
```
# read in
gene_expr = pd.read_csv('gene_expression.csv', header=0, index_col=0).to_numpy()

# rescale with mean
gene_expr = np.divide(gene_expr, np.mean(gene_expr, axis=0))

# log_2 transform
gene_expr = np.log2(gene_expr)

# zero-mean
gene_expr -= np.mean(gene_expr, axis = 0)

gene_expr
```

Out[130]:
```
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]])
```

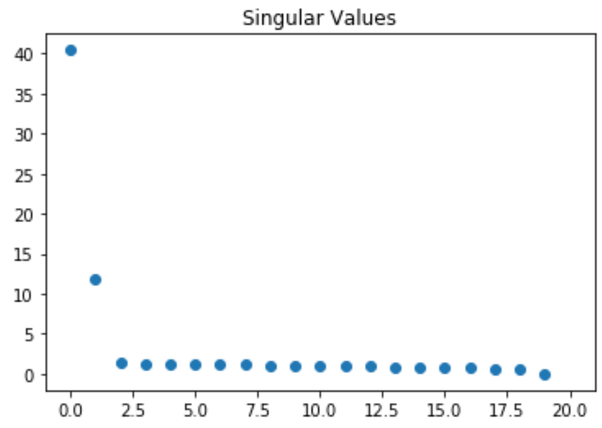
Part 2: PCA

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`.

In [133]:
```
# run svd
u, s, vh = np.linalg.svd(gene_expr)

#plot
plt.scatter(np.arange(0,20), s)
plt.xlim((-1,21))
plt.title("Singular Values")
plt.show()
```



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?

In [0]:
```
d = 2
```

2.3: PCA Derivation/Motivation

Please scan and attach your written work to the submission pdf.

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)

Answer:

- U and V are orthogonal matrices, Σ is a diagonal matrix

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.

Answer:

- $X^T X = V\Sigma^2 V^T$

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.

(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:

- Take the first d vectors from 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).

In [0]:
```
# first d vectors from v
basis = vh[:d]
```

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.apu.edu/~smccathern/PastCourses/F12/LinearAlgebra/LA5_1handout.pdf (https://home.apu.edu/~smccathern/PastCourses/F12/LinearAlgebra/LA5_1handout.pdf) (Theorem 7)

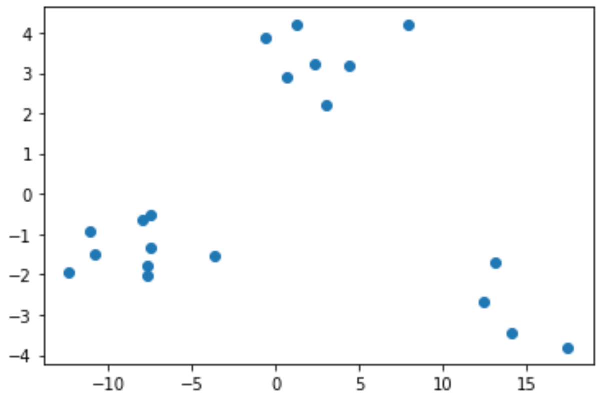
In [0]:
```
def project(v, basis):
    """Takes a vector v and projects it into the space spanned by basis.

    Args:
        v: data vector (1-D numpy array)
        basis: an _orthonormal_ basis of vectors

    Returns:
        a 1-D numpy array, the projected vector
    """
    return np.matmul(v, np.transpose(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.

In [44]:
```
gene_transform = project(gene_expr, basis)
plt.scatter(gene_transform[:,0], gene_transform[:,1])
plt.show()
```



Part 4: Clustering

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

In [0]:
```
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 $\epsilon$ distance, for some small $\epsilon >
0$)

    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 ass
igned 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.

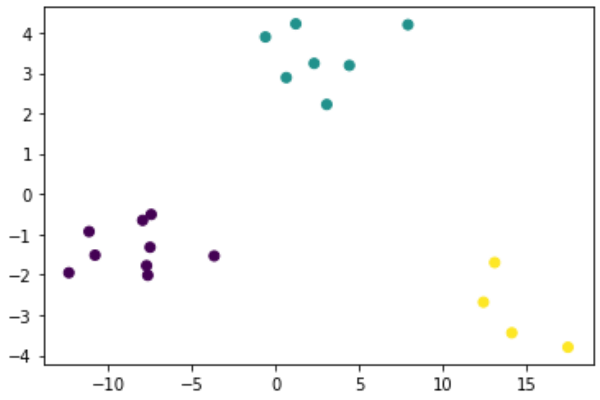
4.2: Implement k-means

In [0]:
```
def kmeans(xs, k):
    """Performs k-means clustering as described above.

    Args:
        xs: list of vectors/data
        k: the number of clusters to form

    Returns:
        ls: list of corresponding labels for each data point, denoting which clu
ster each point belongs to
    """
    eps = 1e-8

    # initialize centroids of clusters
    cs = []  # list of centroids

    for i in range(k):
        point = []
        for dim in np.transpose(xs):
            point.append(np.random.uniform(np.array(dim).min(), np.array(dim).ma
x()))
        cs.append(point)

    ### Slightly cleaner version of above
    #
    # centers = []
    # for dim in np.transpose(xs):
    #     centers.append(np.random.uniform(np.array(dim).min(), np.array(dim).ma
x(), size=k))
    #
    # cs=np.transpose(centers)
    #
    ###

    # Set an old and new centers
    old_cs = np.array(cs)
    new_cs = np.zeros(shape=old_cs.shape)

    # labels
    ls = np.array([0 for i in range(len(xs))])

    # Prevent settingold to new on first loop
    first = True

    # main loop goes here
    while np.linalg.norm(np.subtract(new_cs, old_cs)) > eps:  # TODO: replace Tr
ue condition with condition based on eps
        if not first:
            old_cs = new_cs
        else:
            first = False
        # assign labels
        for i in range(len(xs)):

            # TODO: assign label
            ls[i] = np.argmin(np.linalg.norm(np.subtract(old_cs, xs[i]), axis=
1))

        # recalculate centroids
        for i in range(k):

            # TODO: recalculate centroid position
            points = np.array([xs[cluster_point] for cluster_point in np.arange
```

4.3 Plot projected data with color labels

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

```
In [136]: plt.scatter(t_p[:,0], t_p[:,1],c=kmeans(gene_transform, 3))
          plt.show()
```



Written by Andrew Li for BioE 145 Spring 2020 at UC Berkeley.