Homework 3

Michelle Chen (mc4571) Due Date: April 19, 2019

Problem 1 - KMeans

In [27]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from scipy.stats import multivariate_normal
from numpy.linalg import inv
```

In [2]:
```
mu1 = [0, 0]
mu2 = [3, 0]
mu3 = [0, 3]
sig1 = [[1, 0], [0, 1]]
sig2 = [[1, 0], [0, 1]]
sig3 = [[1, 0], [0, 1]]
pi = [0.2, 0.5, 0.3]
```

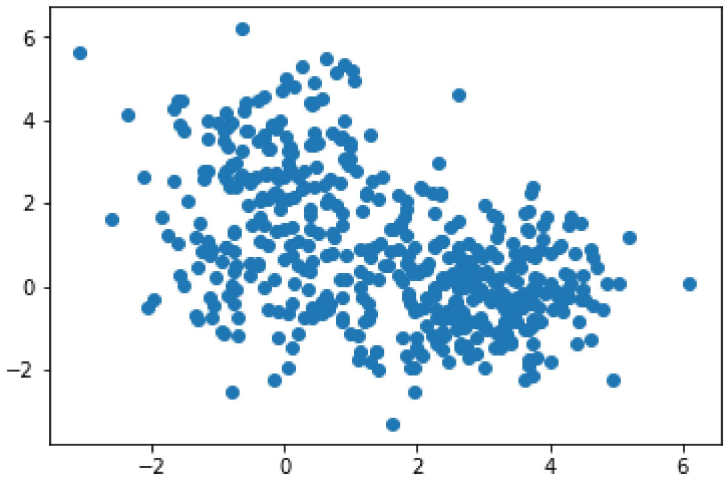
In [3]:
```
np.arange(1,4)
```

Out[3]:
```
array([1, 2, 3])
```

In [4]:
```
# Generate values from weights (pi) distribution
G = []
n = 500
for i in range(n):
    g = np.random.choice(np.arange(1,4), p = pi)
    if g == 1:
        G.append(multivariate_normal.rvs(mu1, sig1))
    elif g == 2:
        G.append(multivariate_normal.rvs(mu2, sig2))
    else:
        G.append(multivariate_normal.rvs(mu3, sig3))
```

In [5]:
```
x = np.transpose(G)[0]
y = np.transpose(G)[1]

plt.scatter(x,y)
```

Out[5]: <matplotlib.collections.PathCollection at 0x1f7577b5b00>



In [6]:
```
coord = list(zip(x,y))
```

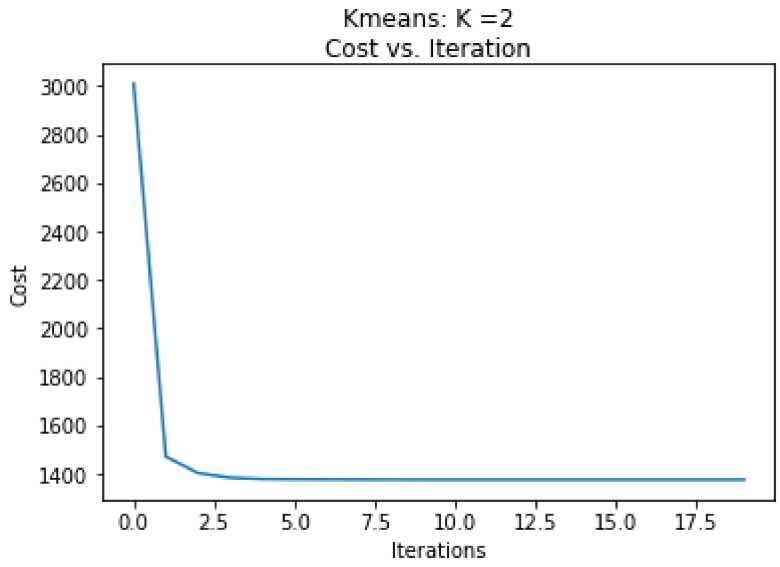
In [7]:
```
def get_dist(list, coord2):
    sum_sq_list = []
    for i in list:
        sum_sq_list.append(np.sum((np.subtract(i, coord2)**2)))
    return sum_sq_list
```

Part 1a, b)

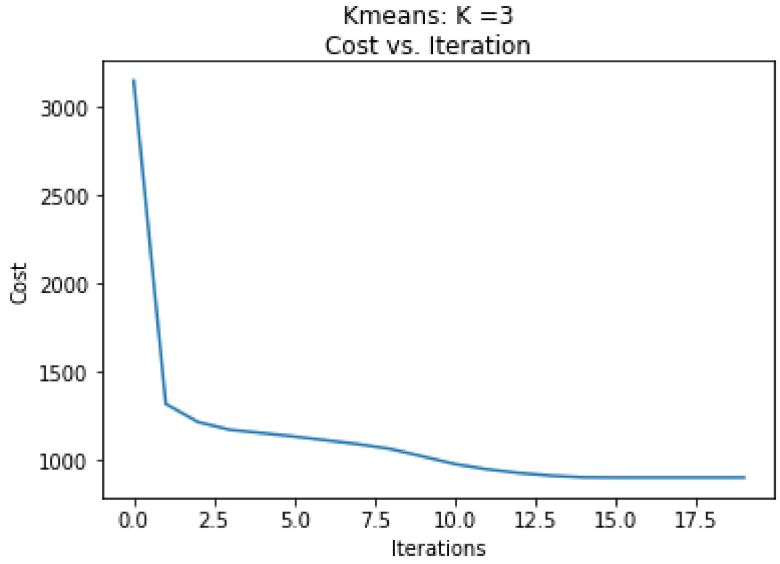
In [8]:
```
def kmeans_plot(num_clusters):
    k = num_clusters

    init_centroid = [coord[i] for i in np.array(np.random.choice(500,k))]

    cluster_labels = np.zeros(len(coord))
    iterations = 20
    d = {}
    total_cost = []

    for i in range(iterations):
        for j in range(k): #go through each k centroid
            d[j] = get_dist(coord, init_centroid[j])

        for index in range(len(coord)):
            cluster_labels[index] = np.argmin(list(zip(*d.values()))[index])

        for j in range(k):
            t = [coord[h] for h in (np.where(cluster_labels == j)[0].tolist
())]
            init_centroid[j]  = [sum(q) / len(q) for q in list(zip(*t))]

        cost = 0
        for j in range(k):
            cost += np.sum([d[j][w] for w in np.where(cluster_labels ==j)[0].t
olist()])

        total_cost.append(cost)


    plt.plot(total_cost)
    plt.xlabel('Iterations')
    plt.ylabel('Cost')
    plt.suptitle('Kmeans: K =' + str(num_clusters) + '\nCost vs. Iteration')
```

In [9]:
```
def kmeans_cluster_labels(num_clusters):
    k = num_clusters

    init_centroid = [coord[i] for i in np.array(np.random.choice(500,k))]

    cluster_labels = np.zeros(len(coord))
    iterations = 20
    d = {}
    total_cost = []

    for i in range(iterations):
        for j in range(k): #go through each k centroid

            #Get dist between 500 coord and centroid
            d[j] = get_dist(coord, init_centroid[j])

        for index in range(len(coord)):

            # Zip values together and then do argmin to find index with lowest
dist
            # Assign that cluster # to the respective element for cluster_labe
ls vector
            cluster_labels[index] = np.argmin(list(zip(*d.values()))[index])

        for j in range(k):
            t = [coord[h] for h in (np.where(cluster_labels == j)[0].tolist
())]
            init_centroid[j]  = [sum(q) / len(q) for q in list(zip(*t))]

        cost = 0
        for j in range(k):
            cost += np.sum([d[j][w] for w in np.where(cluster_labels ==j)[0].t
olist()])

        total_cost.append(cost)

    return cluster_labels
```

In [10]:
```
kmeans_plot(2)
```

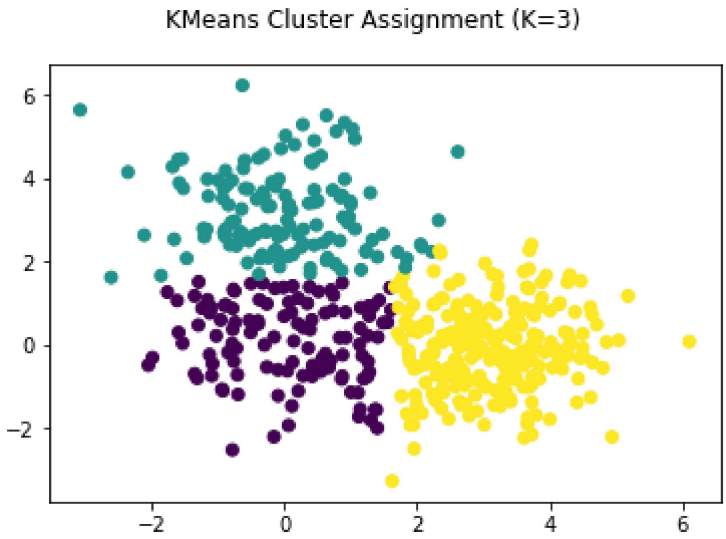


In [11]:
```
kmeans_plot(3)
```

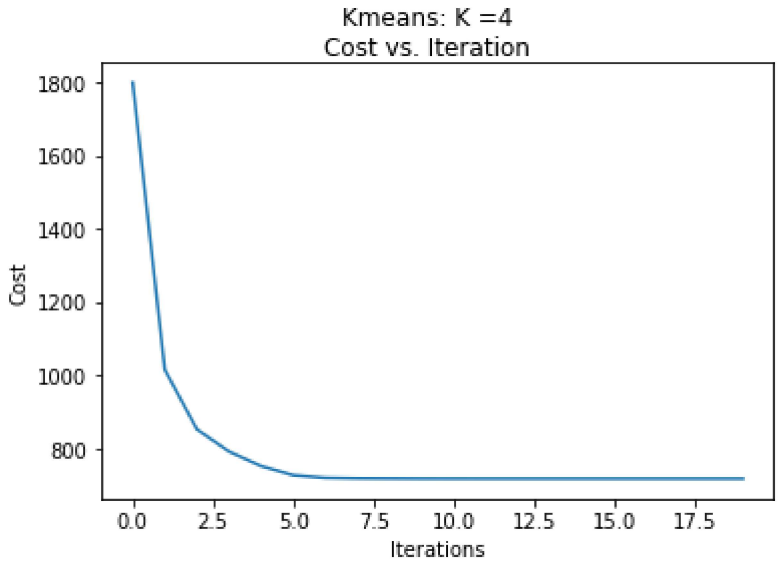


In [12]:
```
k3 = kmeans_cluster_labels(3)
plt.scatter(x,y,c = k3)
plt.suptitle('KMeans Cluster Assignment (K=3)')
```

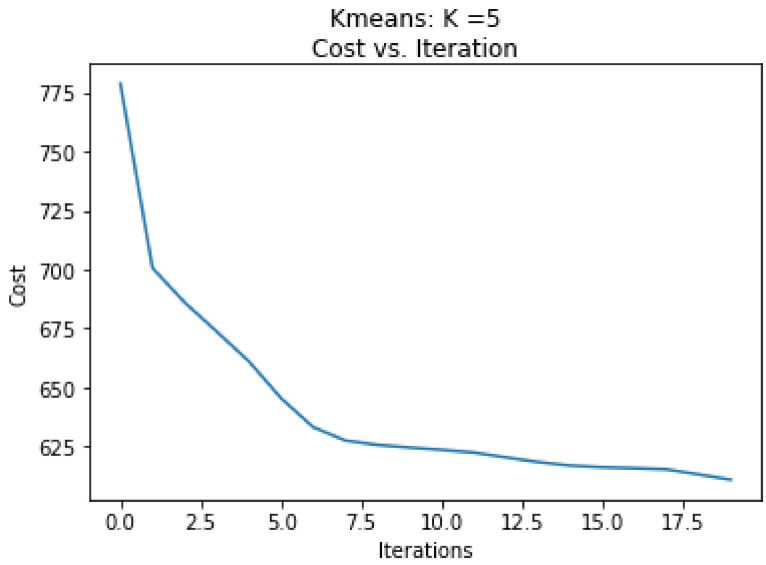
Out[12]: Text(0.5,0.98,'KMeans Cluster Assignment (K=3)')



In [13]:
```
kmeans_plot(4)
```

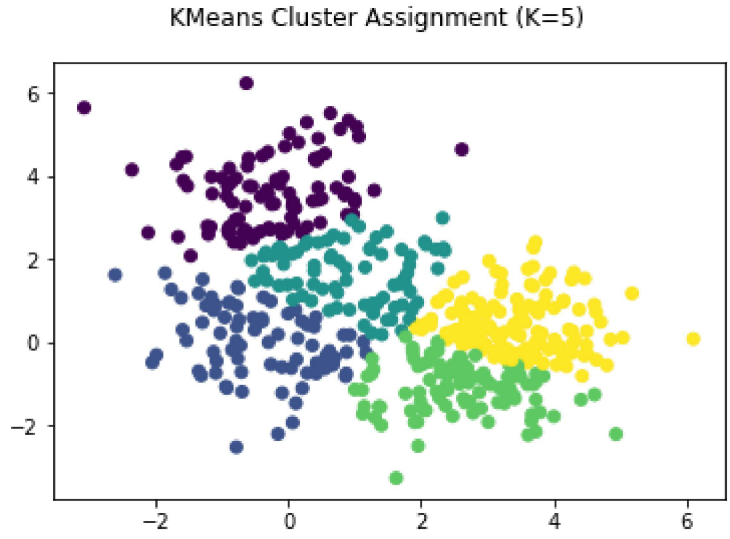


In [14]:
```
kmeans_plot(5)
```



In [15]:
```
k5 = kmeans_cluster_labels(5)
plt.scatter(x,y,c = k5)
plt.suptitle('KMeans Cluster Assignment (K=5)')
```

Out[15]:
```
Text(0.5,0.98,'KMeans Cluster Assignment (K=5)')
```



Problem 2

In [16]:
```
X_train = pd.read_csv("Prob2_Xtrain.csv", header=None)
y_train = pd.read_csv("Prob2_ytrain.csv", header=None)
X_test = pd.read_csv("Prob2_Xtest.csv", header=None)
y_test = pd.read_csv("Prob2_ytest.csv", header=None)

X_train0 = X_train[y_train[0]==0]
X_train1 = X_train[y_train[0]==1]
y_train0 = y_train[y_train[0]==0]
y_train1 = y_train[y_train[0]==1]
```

In [479]:
```
X_train.head()
```

Out[479]:

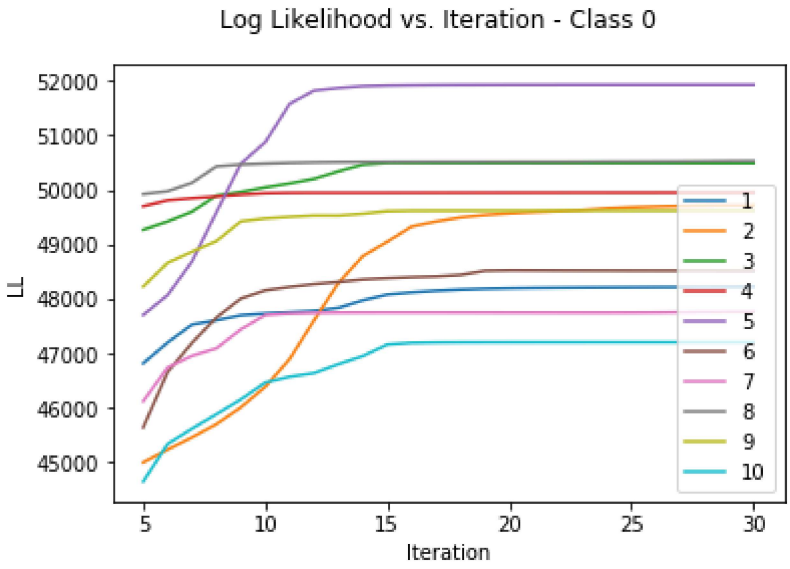
	0	1	2	3	4	5	6	7	
0	-0.042984	0.402080	0.060133	0.008235	-0.011964	0.017788	0.004096	-0.041283	-0.01811
1	-0.021163	0.197970	0.023161	0.000774	-0.000446	0.037280	0.002865	-0.012920	0.02774
2	-0.033537	0.307060	0.048742	0.011210	0.029054	-0.073224	-0.007833	-0.069956	-0.02154
3	-0.033537	0.307060	0.048742	0.011210	0.029054	-0.073224	-0.007833	-0.069956	-0.02154
4	-0.003041	0.028877	-0.000203	-0.003211	-0.005451	0.034991	0.003081	0.005470	0.02607

y_train.head()

Part 2a)

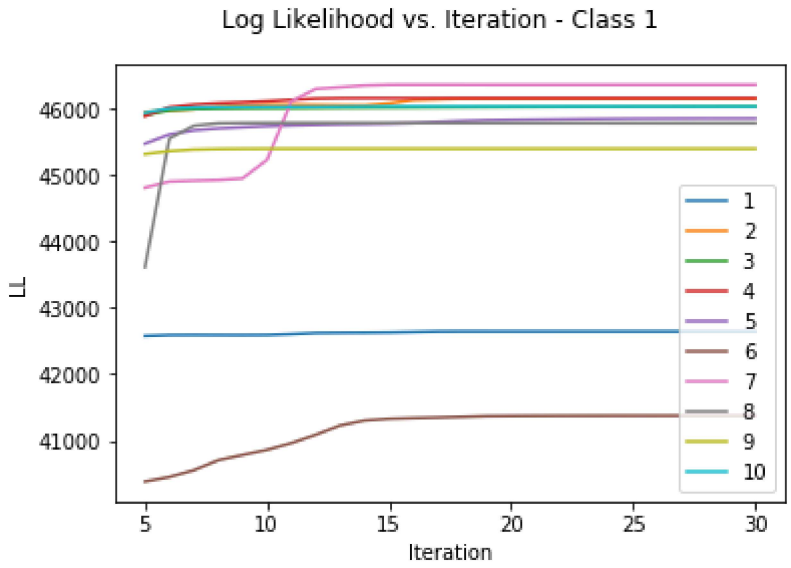
In [642]:
```
from scipy.stats import multivariate_normal

k = 3

log_lik_final_output = []

runs = 10

for i in range(runs):
    sigma_array = []
    sig = np.cov(X_train0, rowvar = False)
    mu = multivariate_normal.rvs(np.mean(X_train0), sig, size = k)


    sig_copy = sig.copy()

    for s in range(k):
        sigma_array.append(sig_copy)

    prob_g = np.array([1/k, 1/k, 1/k])

    log_lik_prev = 0
    log_lik_output = [log_lik_prev]

    for i in range(30):

        ### E-step ###
        phi = np.zeros((X_train0.shape[0],k))
        for index in range(k):
            likelihood = multivariate_normal(mu[index], sigma_array[index]).pd
f(X_train0)
            phi[:,index] = prob_g[index]*likelihood


        def normalize(df):
            return df/np.sum(df)

        phi = np.apply_along_axis(normalize, 1, phi)

        ### M - step ###

        n = X_train0.shape[0]
        pi_k = np.sum(phi, axis = 0) / n

        # Recalculate mu array that maximizes our likelihood
        mu = np.transpose(np.dot(np.transpose(X_train0), phi) / (pi_k * n))

        for s in range(k):
            sigma_array[s] = np.matmul(((X_train0 - mu[s]).T * phi[:,s]), (X_t
rain0 - mu[s]))/ (pi_k*n)[s]

        log_lik = 0

        for p, m, sigma in zip(pi_k, mu, sigma_array):
            log_lik = log_lik + p*multivariate_normal(m, sigma).pdf(X_train0)
        log_lik = np.log(log_lik).sum()
```

```
        log_lik_prev = log_lik
        log_lik_output.append(log_lik_prev)

    log_lik_final_output.append(log_lik_output)
    output = []
    for i in range(len(log_lik_final_output)):
        output.append(log_lik_final_output[i][5:])
    plt.plot(range(5,31), output[i], label = str(i+1))
    plt.suptitle('Log Likelihood vs. Iteration - Class 0')
    plt.xlabel('Iteration')
    plt.ylabel('LL')
    plt.legend()
```



In [643]:
```
k = 3

log_lik_final_output = []

runs = 10

for i in range(runs):
    sigma_array = []
    sig = np.cov(X_train1, rowvar = False)
    mu = multivariate_normal.rvs(np.mean(X_train1), sig, size = k)


    sig_copy = sig.copy()

    for s in range(k):
        sigma_array.append(sig_copy)

    prob_g = np.array([1/k, 1/k, 1/k])

    log_lik_prev = 0
    log_lik_output = [log_lik_prev]

    for i in range(30):

        ### E-step ###
        phi = np.zeros((X_train1.shape[0],k))
        for index in range(k):
            likelihood = multivariate_normal(mu[index], sigma_array[index]).pd
f(X_train1)
            phi[:,index] = prob_g[index]*likelihood


        def normalize(df):
            return df/np.sum(df)

        phi = np.apply_along_axis(normalize, 1, phi)

        ### M - step ###

        n = X_train1.shape[0]
        pi_k = np.sum(phi, axis = 0) / n

        # Recalculate mu array that maximizes our likelihood
        mu = np.transpose(np.dot(np.transpose(X_train1), phi) / (pi_k * n))

        for s in range(k):
            sigma_array[s] = np.matmul(((X_train1 - mu[s]).T * phi[:,s]), (X_t
rain1 - mu[s]))/ (pi_k*n)[s]

        log_lik = 0

        for p, m, sigma in zip(pi_k, mu, sigma_array):
            log_lik = log_lik + p*multivariate_normal(m, sigma).pdf(X_train1)
        log_lik = np.log(log_lik).sum()
        log_lik_prev = log_lik
        log_lik_output.append(log_lik_prev)
```

```
    log_lik_final_output.append(log_lik_output)

    output_1 = []

    for i in range(len(log_lik_final_output)):
        output_1.append(log_lik_final_output[i][5:])
    plt.plot(range(5,31), output_1[i], label = str(i+1))
    plt.suptitle('Log Likelihood vs. Iteration - Class 1')
    plt.xlabel('Iteration')
    plt.ylabel('LL')
    plt.legend()
```



Insights and Observations:

We can see that for Class 1, run # 4, we experience the highest log likelihood (red line). As for class 0, we experience the highest log likelihood with run # 9.

Part 2b)

In [644]:
```
def EM(X_train1, prob_g, mu, sigma_array, k):
    log_lik_prev = 0
    log_lik_output = [log_lik_prev]

    for i in range(30):

        ### E-step ###
        phi = np.zeros((X_train1.shape[0],k))
        for index in range(k):
            likelihood = multivariate_normal(mu[index], sigma_array[index]).pd
f(X_train1)
            phi[:,index] = prob_g[index]*likelihood


        def normalize(df):
            return df/np.sum(df)

        phi = np.apply_along_axis(normalize, 1, phi)

        ### M - step ###

        n = X_train1.shape[0]
        pi_k = np.sum(phi, axis = 0) / n

        # Recalculate mu array that maximizes our likelihood
        mu = np.transpose(np.dot(np.transpose(X_train1), phi) / (pi_k * n))

        for s in range(k):
            sigma_array[s] = np.matmul(((X_train1 - mu[s]).T * phi[:,s]), (X_t
rain1 - mu[s]))/ (pi_k*n)[s]

        log_lik = 0

        for p, m, sigma in zip(pi_k, mu, sigma_array):
            log_lik = log_lik + p*multivariate_normal(m, sigma).pdf(X_train1)
        log_lik = np.log(log_lik).sum()
        log_lik_prev = log_lik
        log_lik_output.append(log_lik_prev)

    #log_lik_final_output.append(log_lik_output)

    return pi_k, mu, sigma_array,log_lik_output
```

In [649]:
```
# Generating final_output for Class  0 (mu, sigma_array, prob_g)

k = 4
final_output_c0 = []
runs = 10
for i in range(runs):

    sigma_array = []

    sig = np.cov(X_train0, rowvar = False)
    mu = multivariate_normal.rvs(np.mean(X_train0), sig, size = k)

    sig_copy = sig.copy()

    for s in range(k):
        sigma_array.append(sig_copy)

    prob_g = np.array([1/k] * k)
    final_output_c0.append(EM(X_train0,prob_g, mu, sigma_array, k))
```

In [ ]:
```
# Generating final_output for Class  0 (mu, sigma_array, prob_g)

k = 4
final_output_c1 = []
runs = 10
for i in range(runs):

    sigma_array = []

    sig = np.cov(X_train1, rowvar = False)
    mu = multivariate_normal.rvs(np.mean(X_train1), sig, size = k)

    sig_copy = sig.copy()

    for s in range(k):
        sigma_array.append(sig_copy)

    prob_g = np.array([1/k] * k)
    final_output_c1.append(EM(X_train1, prob_g, mu, sigma_array, k))
```

In [666]:
```
# Find run with highest log likelihood

c0_loglik = []

for i in range(len(final_output_c0)):
    c0_loglik.append(final_output_c0[i][3])


c1_loglik = []

for j in range(len(final_output_c1)):
    c1_loglik.append(final_output_c1[j][3])

c1_max_run = np.argmax(np.array(list(zip(*c1_loglik))[-1:])[0])
c1_max_run

c0_max_run = np.argmax(np.array(list(zip(*c0_loglik))[-1:])[0])
c0_max_run
```

In [713]:
```
print('Class 0: Run ' + str(c0_max_run) + " has the highest log likelihood")
```

```
Class 0: Run 1 has the highest log likelihood
```

In [714]:
```
print('Class 1: Run ' + str(c1_max_run) + " has the highest log likelihood")
```

```
Class 1: Run 1 has the highest log likelihood
```

In [715]:
```
# We can return our prob_g, mu, sigma_array, and log likelihoods

assert len(final_output_c0[0]) ==4
assert len(final_output_c0[1]) ==4

# across 10 runs

assert len(final_output_c0) == 10
assert len(final_output_c0) == 10
```

In [717]:
```
final_output_c0[4]
```

```
Out[717]: (array([0.26059099, 0.43606593, 0.2207244 , 0.08261867]),
           array([[-0.02738551,  0.24078823,  0.02930267,  0.00912115, -0.05423363,
                   -0.06953077,  0.00996687,  0.01062997,  0.01828967,  0.03489011],
                  [-0.238908  ,  0.06630053, -0.06938206,  0.00148679,  0.00416605,
                    0.006096  ,  0.00113577, -0.01669382,  0.02420616, -0.01058301],
                  [-0.10579372,  0.17028161, -0.28416365,  0.00172393, -0.00896417,
                   -0.00454218,  0.00866929,  0.01121599,  0.07937623,  0.02980961],
                  [-0.19393185,  0.26246713,  0.03202891,  0.00431756,  0.01172561,
                   -0.02561767, -0.02575211,  0.14196639,  0.01362221, -0.0373924 ]]),
           [array([[ 4.27079214e-04, -3.93681888e-03, -6.78435543e-04,
                    -1.80469089e-04, -1.55684414e-03,  1.22887940e-03,
                     3.64986355e-05,  5.53654367e-04,  5.49723030e-04,
                    -5.81812822e-05],
                   [-3.93681888e-03,  3.65413453e-02,  6.46111443e-03,
                     1.64878428e-03,  1.46534979e-02, -1.06055209e-02,
                    -4.64482625e-04, -5.85591098e-03, -5.40669564e-03,
                     7.77350838e-04],
                   [-6.78435543e-04,  6.46111443e-03,  1.33959347e-03,
                     3.18238796e-04,  2.28181535e-03, -1.83855481e-03,
                    -1.67729214e-04, -1.39626329e-03, -1.34768208e-03,
                     3.02255786e-04],
                   [-1.80469089e-04,  1.64878428e-03,  3.18238796e-04,
                     1.29900084e-04,  5.08835610e-04, -1.05401334e-03,
                    -2.95211296e-05, -2.45542746e-04, -1.69283912e-04,
                     9.19255929e-05],
                   [-1.55684414e-03,  1.46534979e-02,  2.28181535e-03,
                     5.08835610e-04,  9.18422617e-02,  7.43561961e-03,
                     7.03568482e-05,  2.44341251e-03,  1.72973878e-03,
                     2.72101440e-03],
                   [ 1.22887940e-03, -1.06055209e-02, -1.83855481e-03,
                    -1.05401334e-03,  7.43561961e-03,  1.39726190e-02,
                     3.32085854e-04,  1.89961018e-03,  1.26827006e-03,
                    -9.23488834e-04],
                   [ 3.64986355e-05, -4.64482625e-04, -1.67729214e-04,
                    -2.95211296e-05,  7.03568482e-05,  3.32085854e-04,
                     2.31772531e-04,  1.18380356e-03,  4.11213265e-04,
                     1.78187079e-04],
                   [ 5.53654367e-04, -5.85591098e-03, -1.39626329e-03,
                    -2.45542746e-04,  2.44341251e-03,  1.89961018e-03,
                     1.18380356e-03,  7.09630442e-03,  2.11502288e-03,
                     9.89972347e-04],
                   [ 5.49723030e-04, -5.40669564e-03, -1.34768208e-03,
                    -1.69283912e-04,  1.72973878e-03,  1.26827006e-03,
                     4.11213265e-04,  2.11502288e-03,  3.80652767e-03,
                     1.36942099e-03],
                   [-5.81812822e-05,  7.77350838e-04,  3.02255786e-04,
                     9.19255929e-05,  2.72101440e-03, -9.23488834e-04,
                     1.78187079e-04,  9.89972347e-04,  1.36942099e-03,
                     9.11844501e-03]]),
            array([[ 3.61275454e-01,  5.92783833e-02, -3.58804691e-02,
                     5.96012680e-04,  4.66261278e-04, -8.76304507e-04,
                     1.19864694e-03, -1.21806778e-03,  9.57252553e-03,
                    -3.11356226e-03],
                   [ 5.92783833e-02,  1.92627847e-02, -7.20882247e-03,
                     4.35448737e-04,  9.54380915e-04, -1.90582469e-03,
                     1.57996883e-04, -2.08851158e-03,  1.57071163e-03,
                    -8.70277717e-04],
```

```
       [-3.58804691e-02, -7.20882247e-03,  1.91018691e-02,
         1.08838440e-04,  2.27711534e-05, -4.43814136e-04,
         5.06584647e-05,  2.15240639e-05,  4.42169603e-04,
         1.48666884e-04],
       [ 5.96012680e-04,  4.35448737e-04,  1.08838440e-04,
         1.97077408e-05,  4.57673863e-05, -1.41959964e-04,
        -4.43390341e-06, -9.63306398e-05,  2.08205856e-05,
        -2.30812864e-05],
       [ 4.66261278e-04,  9.54380915e-04,  2.27711534e-05,
         4.57673863e-05,  1.35374572e-04, -4.65657309e-04,
        -3.70634998e-05, -2.03811822e-04, -2.65729414e-04,
         4.05595565e-05],
       [-8.76304507e-04, -1.90582469e-03, -4.43814136e-04,
        -1.41959964e-04, -4.65657309e-04,  2.08372598e-03,
         1.96133341e-04,  4.64029184e-04,  1.48524888e-03,
        -3.41086263e-04],
       [ 1.19864694e-03,  1.57996883e-04,  5.06584647e-05,
        -4.43390341e-06, -3.70634998e-05,  1.96133341e-04,
         4.44644890e-05,  6.28967324e-05,  2.96750623e-04,
        -8.23983378e-05],
       [-1.21806778e-03, -2.08851158e-03,  2.15240639e-05,
        -9.63306398e-05, -2.03811822e-04,  4.64029184e-04,
         6.28967324e-05,  1.08782444e-03, -8.31657232e-04,
         2.83130928e-04],
       [ 9.57252553e-03,  1.57071163e-03,  4.42169603e-04,
         2.08205856e-05, -2.65729414e-04,  1.48524888e-03,
         2.96750623e-04, -8.31657232e-04,  4.55473205e-03,
        -1.32126969e-03],
       [-3.11356226e-03, -8.70277717e-04,  1.48666884e-04,
        -2.30812864e-05,  4.05595565e-05, -3.41086263e-04,
        -8.23983378e-05,  2.83130928e-04, -1.32126969e-03,
         6.31513682e-04]]),
array([[ 1.75985278e-02, -1.15942058e-03,  1.66169174e-02,
         1.56664537e-04,  9.83824871e-04, -3.66487627e-05,
         2.60286109e-04,  2.74196557e-03, -5.47678471e-04,
         1.89886329e-03],
       [-1.15942058e-03,  1.51347619e-02,  1.25418055e-03,
         4.63894099e-04,  4.48434732e-04, -1.87805861e-03,
         1.58352372e-04,  5.01968447e-04, -3.75221486e-03,
        -3.90856460e-03],
       [ 1.66169174e-02,  1.25418055e-03,  1.38598211e-01,
         1.68403796e-03, -2.86725083e-03, -8.11977362e-06,
         3.88417276e-03,  1.22738324e-03,  3.32374273e-02,
         6.41083432e-03],
       [ 1.56664537e-04,  4.63894099e-04,  1.68403796e-03,
         6.43741447e-05,  6.76867408e-05, -5.57642840e-04,
         9.11524722e-05,  2.93110874e-04,  4.69394156e-04,
         2.99428283e-04],
       [ 9.83824871e-04,  4.48434732e-04, -2.86725083e-03,
         6.76867408e-05,  3.71306164e-03, -9.98371535e-04,
        -2.16818612e-05,  3.52794557e-04,  4.10485836e-04,
         8.89725274e-04],
       [-3.66487627e-05, -1.87805861e-03, -8.11977362e-06,
        -5.57642840e-04, -9.98371535e-04,  9.94308681e-03,
        -5.59681453e-04, -6.44946675e-03,  2.32025202e-03,
        -7.82888771e-03],
       [ 2.60286109e-04,  1.58352372e-04,  3.88417276e-03,
```

```
          9.11524722e-05, -2.16818612e-05, -5.59681453e-04,
          6.04241633e-04,  2.05559863e-03,  2.38176880e-03,
          8.53098909e-04],
        [ 2.74196557e-03,  5.01968447e-04,  1.22738324e-03,
          2.93110874e-04,  3.52794557e-04, -6.44946675e-03,
          2.05559863e-03,  1.68715612e-02, -5.74601929e-03,
          6.24095046e-03],
        [-5.47678471e-04, -3.75221486e-03,  3.32374273e-02,
          4.69394156e-04,  4.10485836e-04,  2.32025202e-03,
          2.38176880e-03, -5.74601929e-03,  4.01435274e-02,
          2.53209411e-03],
        [ 1.89886329e-03, -3.90856460e-03,  6.41083432e-03,
          2.99428283e-04,  8.89725274e-04, -7.82888771e-03,
          8.53098909e-04,  6.24095046e-03,  2.53209411e-03,
          4.40837660e-02]]),
 array([[ 7.77786946e-02,  4.59772995e-03, -7.07822237e-03,
         -7.49730532e-04, -1.79752756e-03,  6.99657245e-03,
         -8.10111672e-03,  7.29764588e-03,  5.62051332e-03,
          2.58794104e-03],
        [ 4.59772995e-03,  6.71022045e-02,  1.11540673e-02,
          2.42139744e-03,  5.55725337e-03, -6.83726781e-03,
          8.10510205e-03, -1.07281844e-03, -1.11447568e-02,
          2.03860488e-03],
        [-7.07822237e-03,  1.11540673e-02,  3.84672085e-03,
          2.42250186e-04,  7.61503931e-04,  1.66729979e-03,
         -4.04171595e-04, -5.94067207e-03, -3.89331154e-03,
          2.27559802e-03],
        [-7.49730532e-04,  2.42139744e-03,  2.42250186e-04,
          4.38881196e-04,  8.12935129e-04, -4.09760650e-03,
          2.84657141e-03,  1.31692112e-03,  5.11533320e-05,
         -1.25518477e-03],
        [-1.79752756e-03,  5.55725337e-03,  7.61503931e-04,
          8.12935129e-04,  1.85339837e-03, -9.34826517e-03,
          2.35297784e-03,  4.22508055e-04, -2.77054618e-04,
         -2.08618518e-03],
        [ 6.99657245e-03, -6.83726781e-03,  1.66729979e-03,
         -4.09760650e-03, -9.34826517e-03,  6.64845173e-02,
         -2.93867666e-03, -2.26357683e-02, -7.57561249e-03,
          2.06945643e-02],
        [-8.10111672e-03,  8.10510205e-03, -4.04171595e-04,
          2.84657141e-03,  2.35297784e-03, -2.93867666e-03,
          5.35978333e-02,  1.47478289e-02, -7.33686134e-04,
         -6.20901411e-03],
        [ 7.29764588e-03, -1.07281844e-03, -5.94067207e-03,
          1.31692112e-03,  4.22508055e-04, -2.26357683e-02,
          1.47478289e-02,  7.30277740e-02,  1.33358402e-02,
         -2.25462185e-02],
        [ 5.62051332e-03, -1.11447568e-02, -3.89331154e-03,
          5.11533320e-05, -2.77054618e-04, -7.57561249e-03,
         -7.33686134e-04,  1.33358402e-02,  7.18309727e-03,
         -5.93944110e-03],
        [ 2.58794104e-03,  2.03860488e-03,  2.27559802e-03,
         -1.25518477e-03, -2.08618518e-03,  2.06945643e-02,
         -6.20901411e-03, -2.25462185e-02, -5.93944110e-03,
          1.13659273e-02]])],
[0,
 39468.95804957945,
```

```
44837.01082199905,
47010.95873679289,
47958.72677709899,
48235.10187089043,
48386.01343435364,
48523.034049841066,
48714.02099929041,
48897.199301744164,
48990.15682669195,
49085.84582747561,
49391.664448002186,
49798.92202471259,
50585.50407082659,
51215.6865426749,
51675.42794988315,
51823.62434526112,
51851.64418682888,
51854.55655772227,
51852.44048834001,
51848.9781290768,
51844.93747634007,
51840.724403358865,
51836.62918777441,
51832.771134259354,
51829.194845130216,
51825.89204634563,
51822.81851267195,
51819.91650032896,
51817.11247952479])
```

In [724]:
```
def nb_predict(df, k, final_output_best_0, final_output_best_1):

    prior_c1 = y_train[y_train[0]==1].shape[0] / y_train.shape[0]
    prior_c0 = 1- prior_c1

    prob_g0 = final_output_best_0[0]
    prob_g1 = final_output_best_1[0]

    mu0 = final_output_best_0[1]
    mu1 = final_output_best_1[1]

    sigma_array0 = final_output_best_0[2]
    sigma_array1 = final_output_best_1[2]

    p0 = 0
    p1 = 0
    for i in range(k):
        p0  = p0 + multivariate_normal(mu0[i], sigma_array0[i]).pdf(df)  * pro
b_g0[i]
        p1 = p1 + multivariate_normal(mu1[i], sigma_array1[i]).pdf(df) * prob_
g1[i]
    p0 = p0 * prior_c1
    p1 = p1 * prior_c0

    if p0 < p1:
        return 1
    else:
        return 0
```

In [647]:
```
X_test.shape
```

Out[647]: (460, 10)

Results: Gaussian GMM for 1,2,3,4 mixture models

In [731]:
```
for num_mix in range(1,5):

    k = num_mix
    X_test_n = X_test.shape[0]
    y_pred = X_test.apply(lambda x: nb_predict(x, k, final_output_c0[c0_max_ru
n], final_output_c1[c1_max_run]), axis=1)
    y_pred_n = len(y_pred)

    confusion_matrix = [[0,0],[0,0]]
    for i in range(y_pred_n):
        pred = y_pred[i]
        ytest = y_test[0].values[i]
        confusion_matrix[pred][ytest] += 1

    print('\n For Gaussian GMM = ' + str(num_mix) + ":")
    print("\nAccuracy: ", (confusion_matrix[0][0] +  confusion_matrix[1][1]) /
X_test_n)
    print(confusion_matrix[0], '\n',  confusion_matrix[1])
    print('\n####################')
```

```
 For Gaussian GMM = 1:

Accuracy:  0.7630434782608696
[204, 35]
 [74, 147]

####################

 For Gaussian GMM = 2:

Accuracy:  0.7695652173913043
[199, 27]
 [79, 155]

####################

 For Gaussian GMM = 3:

Accuracy:  0.8043478260869565
[200, 12]
 [78, 170]

####################

 For Gaussian GMM = 4:

Accuracy:  0.8565217391304348
[221, 9]
 [57, 173]

####################
```

Problem 3

In [364]:
```
ratings = pd.read_csv('Prob3_ratings.csv', header = None)
ratings_test = pd.read_csv('Prob3_ratings_test.csv', header = None)
```

In [365]:
```
ratings.head()
```

Out[365]:

	0	1	2
0	196	242	-0.53039
1	186	302	-0.53039
2	244	51	-1.53040
3	166	346	-2.53040
4	298	474	0.46961

In [366]:
```
ratings = ratings.rename(index=str, columns={0: "user_id", 1: "movie_id", 2:
'rating'})
ratings.head()
```

Out[366]:

	user_id	movie_id	rating
0	196	242	-0.53039
1	186	302	-0.53039
2	244	51	-1.53040
3	166	346	-2.53040
4	298	474	0.46961

In [367]:
```
ratings_test = ratings_test.rename(index=str, columns={0: "user_id", 1: "movie
_id", 2: 'rating'})
ratings_test.head()
```

Out[367]:

	user_id	movie_id	rating
0	617	590	-2.53040
1	836	12	1.46960
2	933	239	-0.53039
3	85	622	-0.53039
4	886	204	-0.53039

In [368]:
```
# Look at unique user_id
print('There are ' + str(len(ratings['user_id'].unique())) + ' unique users in
Train')
```

```
There are 943 unique users in Train
```

In [369]:
```
print('There are ' + str(len(ratings_test['user_id'].unique())) + ' unique use
rs in Test')
```

```
There are 838 unique users in Test
```

In [370]:
```
# Look at unique movie ids
print('There are ' + str(len(ratings['movie_id'].unique())) + ' unique movies
 in Train')
```

```
There are 1676 unique movies in Train
```

In [371]:
```
# Look at unique movie ids
print('There are ' + str(len(ratings_test['movie_id'].unique())) + ' unique mo
vies in Test')
```

```
There are 1062 unique movies in Test
```

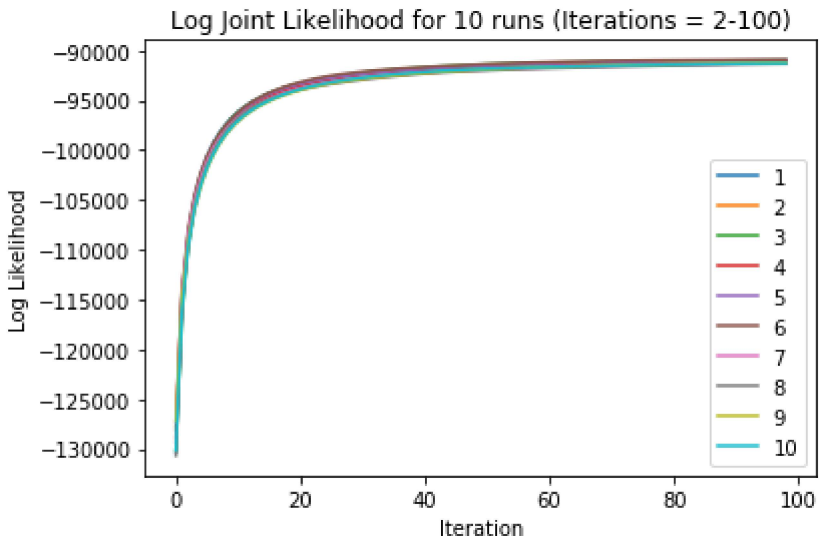
In [ ]:
```
# Initialize U, V

runs = 10
num_users = ratings['user_id'].max()
num_movies = ratings['movie_id'].max()

U = np.random.multivariate_normal(np.zeros(runs), np.identity(runs), num_users
)
V = np.random.multivariate_normal(np.zeros(runs), np.identity(runs), num_movie
s)

num_users, num_movies

r_np= np.zeros((ratings['user_id'].max(),ratings['movie_id'].max()))
r_np_test = np.zeros((ratings['user_id'].max(),ratings['movie_id'].max()))

for g in ratings.iterrows():

    r_np[int(g[1]['user_id']-1), int(g[1]['movie_id']-1)] = g[1]['rating']

for g in ratings_test.iterrows():

    r_np_test[int(g[1]['user_id']-1), int(g[1]['movie_id']-1)] = g[1]['rating'
]
```

In [28]:
```
rmse_output = []
L_final_output = []

iteration = 100
runs = 10
sigma2 = 0.25

for r in range(runs):
    L_array = []
    U = np.random.multivariate_normal(np.zeros(runs), np.identity(runs), num_u
sers)
    V = np.random.multivariate_normal(np.zeros(runs), np.identity(runs), num_m
ovies)
    for i in range(iteration):
        for user_index in range(r_np.shape[0]):
            nz_user_index = np.where(r_np[user_index]!=0)[0]
            vv = np.zeros((runs,runs))
            mv = np.zeros((runs,1))

            for j in nz_user_index:
                vv = vv + (V[j].reshape(10,1)).dot(V[j].reshape(10,1).T)
                mv = mv + r_np[user_index][j]*(V[j].reshape(10,1))
            U[user_index] = inv(np.identity(runs) * sifgma2 + vv).dot(mv).flat
ten()

        for movie_index in range(r_np.shape[1]):
            nz_movie_index = np.where(r_np[:,movie_index]!=0)[0]
            uu = np.zeros((runs, runs))
            mu = np.zeros((runs,1))

            for i in nz_movie_index:
                uu = uu + (U[i].reshape(10,1)).dot(U[i].reshape(10,1).T)
                mu = mu + r_np[:,movie_index][i]*(U[i].reshape(10,1))
            V[movie_index] = inv(np.identity(runs) * sigma2 + uu).dot(mu).flat
ten()


        lam = 1

        # First part
        Lsum1 = 0
        pairs1 = list(zip(np.where(r_np!=0)[0], np.where(r_np!=0)[1]))
        for i,j in pairs1:
            Lsum1 = Lsum1 + (0.5 * (1/(sigma2))) * np.power((r_np[i,j] - U[i].
dot(V[j])),2)

        # Second part
        Lsum2 = 0
        n1 = r_np.shape[0]
        for i in range(n1):
            Lsum2 = Lsum2 + (lam/2) * (U[i].dot(U[i]))

        # Third part
        Lsum3 = 0
        n2 = r_np.shape[1]
        for j in range(n2):
```

```
                Lsum3 = Lsum3 + (lam/2) * (V[j].dot(V[j]))

            # --> Combining each of the three log loss sums
            L = -Lsum1 - Lsum2 - Lsum3
            L_array.append(L)

        L_final_output.append(L_array)

        # Predict using U*V
        rating_pred = U.dot(V.transpose())
        diff = (rating_pred[np.where(r_np_test!=0)] - r_np_test[np.where(r_np_test
!=0)])
        rmse = np.sqrt((diff**2).sum()/r_np_test.shape[0])
        rmse_output.append(rmse)
```

In [31]:
```
for i in range(10):
    plt.plot(L_final_output[i][1:], label= str(i+1))
plt.legend()
plt.title("Log Joint Likelihood for 10 runs (Iterations = 2-100)")
plt.xlabel("Iteration")
plt.ylabel("Log Likelihood")
```

Out[31]: Text(0,0.5,'Log Likelihood')



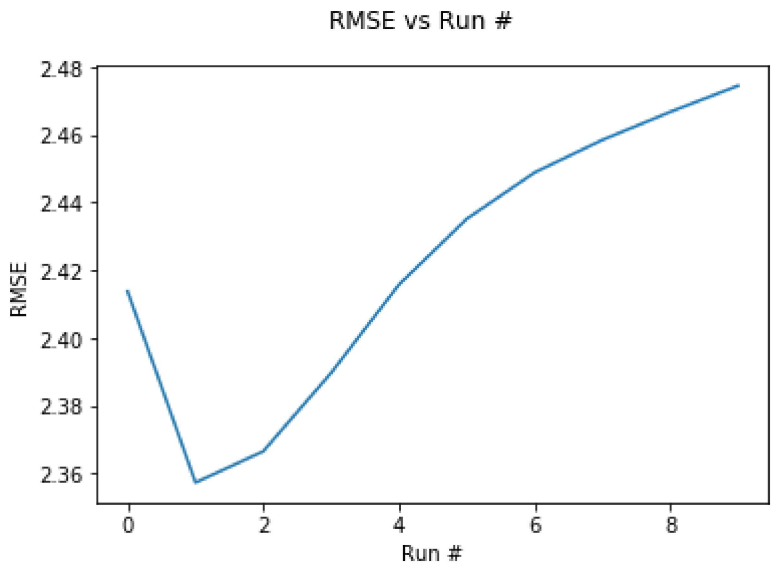
In [35]:
```
df = pd.DataFrame({'Log Likelihood': L_final_output.T[-1], 'RMSE': rmse_output
})
sorted_table = df.sort_values(by = 'Log Likelihood', ascending = False)
sorted_table
```

Out[35]:

	Log Likelihood	RMSE
5	-90910.144212	3.492820
4	-90964.549318	3.593479
2	-90966.880956	3.583208
3	-91083.746859	3.521931
0	-91137.578768	3.516827
1	-91152.197636	3.589980
8	-91221.234030	3.547038
6	-91238.507469	3.500046
9	-91277.909418	3.576665
7	-91327.081096	3.521905

In [806]:
```
#test rmse

plt.plot(rmse_output)
plt.xlabel('Run #')
plt.ylabel('RMSE')
plt.suptitle('RMSE vs Run #')
```

Out[806]: Text(0.5,0.98,'RMSE vs Run #')



Part b)

In [86]:
```
titles = pd.read_table('Prob3_movies.txt', header=None)
titles = titles.rename(columns ={0: 'Titles'})
titles.head()
```

Out[86]:

	Titles
0	Toy Story (1995)
1	GoldenEye (1995)
2	Four Rooms (1995)
3	Get Shorty (1995)
4	Copycat (1995)

In [87]:
```
M = pd.DataFrame(V)
M_merged =pd.concat([titles, M], axis = 1)
```

In [90]:
```
# Get rid of years

M_merged['Titles'] = M_merged['Titles'].apply(lambda x: x.split(' (')[0])
M_merged_new = M_merged.set_index('Titles')
```

In [91]:
```
M_merged_new.head()
```

Out[91]:

Titles	0	1	2	3	4	5	6	7
Toy Story	0.140680	-0.046656	-0.191644	-0.001840	-0.426929	0.412328	0.148938	0.453114
GoldenEye	0.145842	0.047146	-0.396591	-0.338986	-0.314637	0.558856	-0.106475	0.816120
Four Rooms	-1.365709	0.253187	-0.346226	0.289881	0.208765	0.484893	-0.011512	1.250325
Get Shorty	-0.197821	0.053881	0.505892	0.332693	-0.572958	0.688231	-0.020377	0.426246
Copycat	-0.228182	0.047736	0.005653	0.110104	0.562182	-0.260106	0.020295	0.746319

In [96]:
```
given_movies = ['Star Wars', 'My Fair Lady', 'GoodFellas']
subset_rows = M_merged_new.loc[given_movies,:]
subset_rows
```

Out[96]:

Titles	0	1	2	3	4	5	6	7
Star Wars	-0.206968	-0.288278	0.425634	0.290666	-0.591603	0.635059	0.081499	-0.362376
My Fair Lady	-0.117464	-0.020187	-0.118161	-0.416416	-0.781749	-0.186341	-0.335801	0.832058
GoodFellas	0.060822	0.679786	-0.275898	0.230171	-1.023493	0.222953	0.422239	-0.041731

Star wars 10 closest movies

In [114]:
```
star_wars = subset_rows.iloc[0,:]
M_c = M_merged_new.copy()
M_c['L2'] = M_c.apply(lambda x: np.linalg.norm(x - star_wars), axis = 1)
output_sw = M_c['L2'].sort_values(ascending = True)[1:11]
pd.DataFrame(output_sw)
```

Out[114]:

Titles	L2
Empire Strikes Back, The	0.308051
Return of the Jedi	0.664561
Raiders of the Lost Ark	0.697484
Manon of the Spring	0.820811
Blues Brothers, The	0.871707
Close Shave, A	0.890216
L.A. Confidential	0.915165
Wrong Trousers, The	0.918327
Indiana Jones and the Last Crusade	0.926887
Terminator, The	0.934029

My fair lady 10 closest movies

In [113]:
```
my_fair_lady = subset_rows.iloc[1,:]
M_c = M_merged_new.copy()
M_c['L2'] = M_c.apply(lambda x: np.linalg.norm(x - my_fair_lady), axis = 1)
output_mp = M_c['L2'].sort_values(ascending = True)[1:11]

pd.DataFrame(output_mp)
```

Out[113]:

Titles	L2
Mary Poppins	0.618227
Wizard of Oz, The	0.948103
Miracle on 34th Street	0.985554
Sabrina	1.002924
Babe	1.015618
Sound of Music, The	1.033339
It's a Wonderful Life	1.055874
Paradise Road	1.056174
Singin' in the Rain	1.062661
Cinderella	1.081797

Good fellas 10 closest movies

In [112]:
```
good_fellas = subset_rows.iloc[2,:]
M_c = M_merged_new.copy()
M_c['L2'] = M_c.apply(lambda x: np.linalg.norm(x - good_fellas), axis = 1)
output_gf = M_c['L2'].sort_values(ascending = True)[1:11]

pd.DataFrame(output_gf)
```

Out[112]:

Titles	L2
Casino	0.644632
Good, The Bad and The Ugly, The	0.651373
Once Upon a Time in the West	0.667882
Full Metal Jacket	0.669690
Godfather: Part II, The	0.682719
Short Cuts	0.800816
Apocalypse Now	0.828561
Cool Hand Luke	0.961795
2001: A Space Odyssey	1.020255
Deer Hunter, The	1.027391