Applied Machine Learning
Homework 10

Daniel Gonzalez
Colton Piper
15^{th} of November, 2018

1 Results

For this assignment, we used the `GaussianMixture` package from SciKitLearn for Python to call the EM algorithm.

1.1 Table

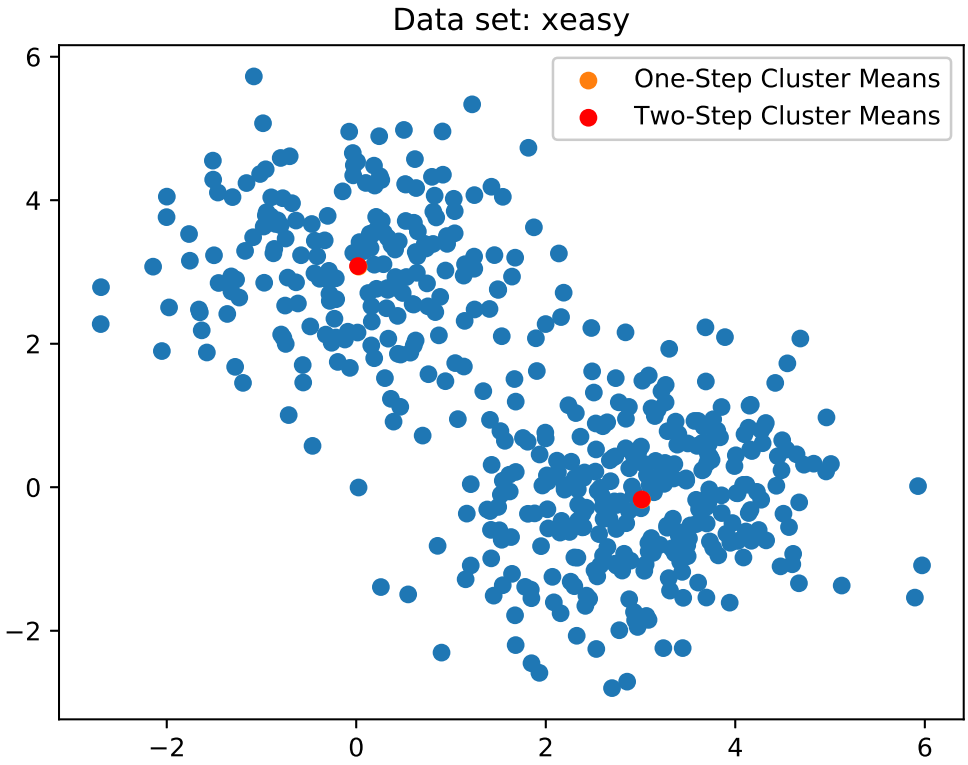
Table 1: One-Step EM Algorithm

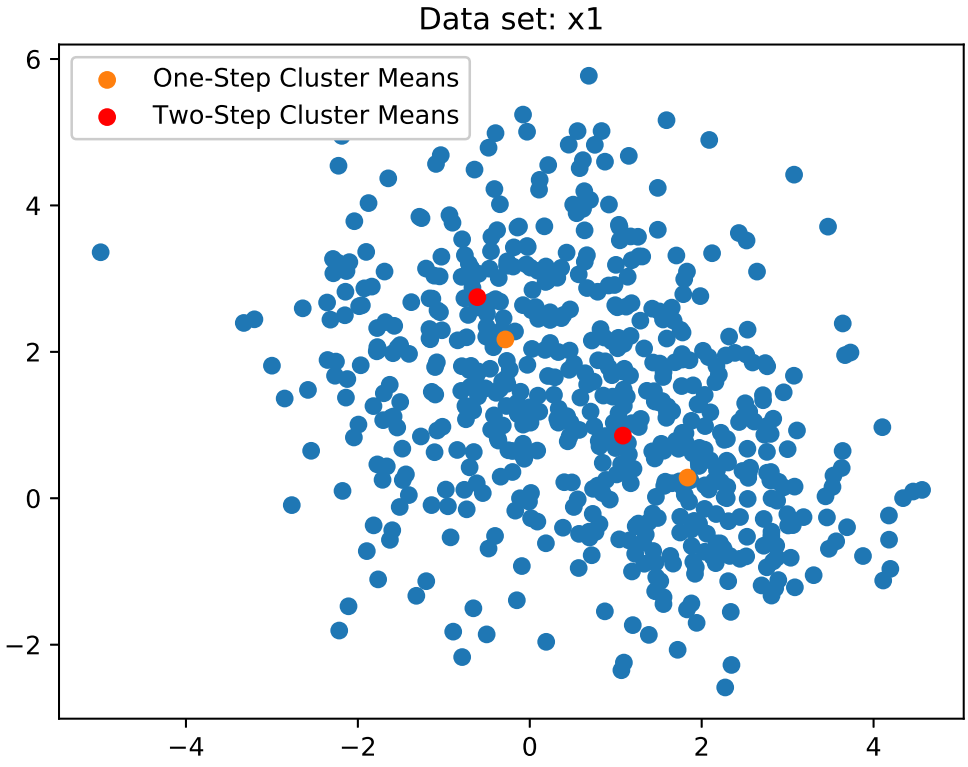
Data Set	k	π_k	μ_k	Σ_k
`xeasy`	1	0.407871	$(0.02478088, 3.07460079)$	$\begin{pmatrix} 1.01363993 & -0.05311694 \\ -0.05311694 & 0.94863522 \end{pmatrix}$
	2	0.592129	$(3.01650103, -0.17460086)$	$\begin{pmatrix} 1.00808335 & 0.15670429 \\ 0.15670429 & 0.94521241 \end{pmatrix}$
`x1`	1	0.55565978	$(-0.28366821, 2.16960597)$	$\begin{pmatrix} 1.59256984 & 0.35809924 \\ 0.35809924 & 2.07505554 \end{pmatrix}$
	2	0.44434022	$(1.83617659, 0.28263171)$	$\begin{pmatrix} 1.12412731 & -0.09771768 \\ -0.09771768 & 1.22889663 \end{pmatrix}$
`x2`	1	0.56150159	$(0.03484, -0.03989854)$	$\begin{pmatrix} 1.31365667 & 0.10374308 \\ 0.10374308 & 0.97577869 \end{pmatrix}$
	2	0.43849841	$(0.17023148, -0.14255449)$	$\begin{pmatrix} 9.86732223 & 0.79962458 \\ 0.79962458 & 10.26300564 \end{pmatrix}$

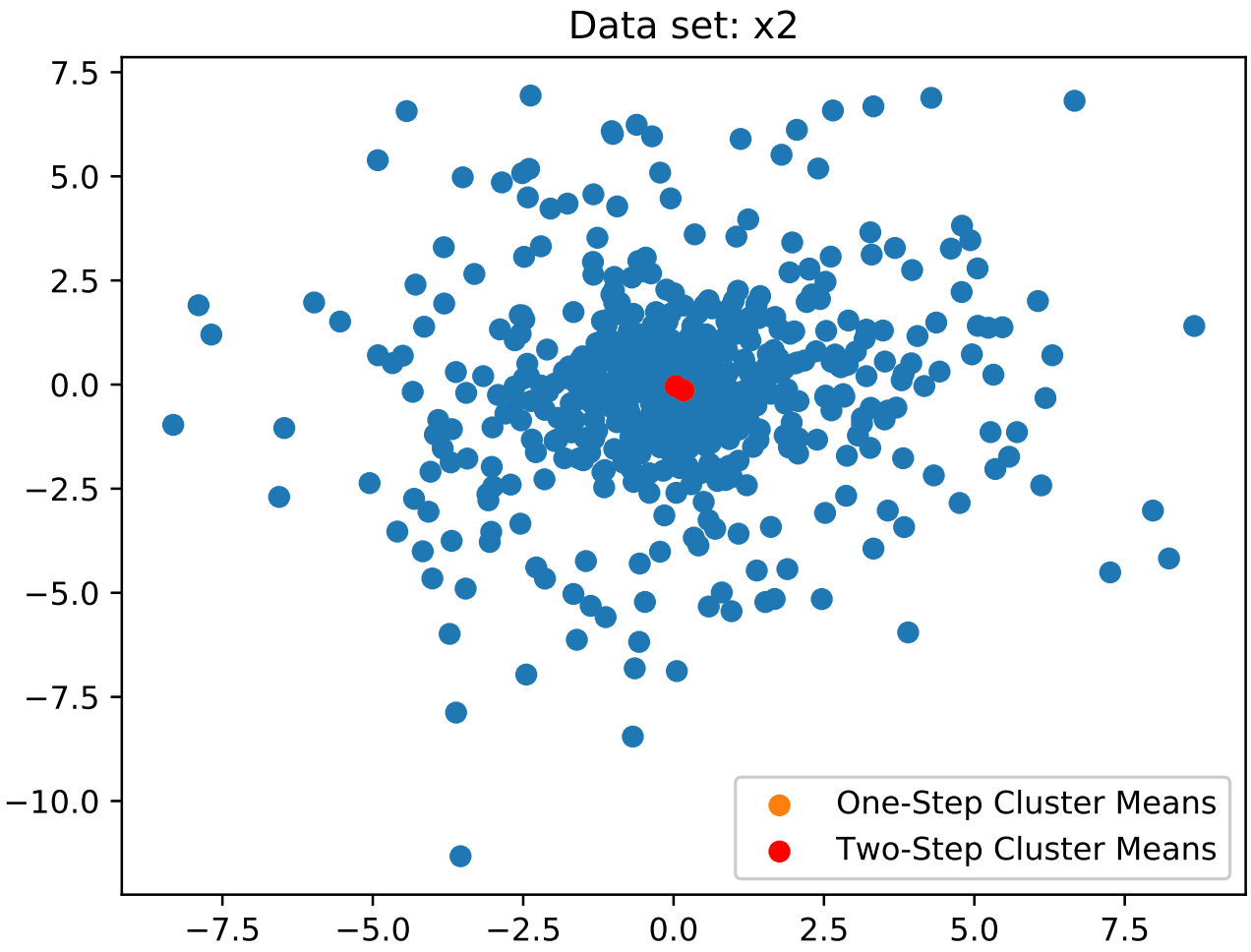
Table 2: Two-Step (Provable) EM Algorithm

Data Set	k	π_k	μ_k	Σ_k
`xeasy`	1	0.59388969	$(3.01183881, -0.17000872)$	$\begin{pmatrix} 1.01430655 & 0.14999363 \\ 0.14999363 & 0.95097821 \end{pmatrix}$
	2	0.40611031	$(0.01862825, 3.08197222)$	$\begin{pmatrix} 1.00648418 & -0.0442361 \\ -0.0442361 & 0.93800178 \end{pmatrix}$
`x1`	1	0.74863867	$(1.08395907, 0.85578155)$	$\begin{pmatrix} 2.1600946 & -0.51568872 \\ -0.51568872 & 1.99120546 \end{pmatrix}$
	2	0.25136133	$(-0.6095988, 2.74694693)$	$\begin{pmatrix} 1.34102811 & 0.62338099 \\ 0.62338099 & 1.64899629 \end{pmatrix}$
`x2`	1	0.54930196	$(0.03588655, -0.03956158)$	$\begin{pmatrix} 1.28017758 & 0.10341123 \\ 0.10341123 & 0.94265068 \end{pmatrix}$
	2	0.45069804	$(0.16529115, -0.14018644)$	$\begin{pmatrix} 9.67740853 & 0.78075241 \\ 0.78075241 & 10.05218714 \end{pmatrix}$

2 Figures


Data set: xeasy
One-Step Cluster Means
Two-Step Cluster Means
6
4
2
0
−2
−2
0
2
4
6


Data set: x1
One-Step Cluster Means
Two-Step Cluster Means
6
4
2
0
−2
−4
−2
0
2
4



3 Appendix: Code

If the code looks too small, please zoom in on the pdf. The screenshots are `.png` images, so you should be able to zoom in and read at whatever is a comfortable size for you. The first two screenshots are of the actual EM code, while the last screenshot is the python code for graphing to verify our results.

```
...assignment10.py | ...t10 — λ — -bash | ...vim report.tex | ...xmk -pdf -pvc ...
```

```python
# Daniel Gonzalez, FSU Mathematics PhD
# Colton Piper, FSU Mathematics PhD
# Applied Machine Learning Assignment 10

from __future__ import print_function
import random
import numpy as np
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture

#PARAMETERS FOR THE SCIKITLEARN GAUSSIAN MIXTURE MODEL
#---N_COMPONENTS:------NUMBER OF CLUSTERS
 | DEFAULT: 1
#---COVARIANCE_TYPE:---'full' MEANS EVERY CLUSTER HAS ITS OWN INDEPENDENT COVARIANCE MATRIX
 | DEFAULT: 'full'
#---TOL:---------------TOLERANCE FOR CONVERGENCE
 | DEFAULT: 1e-3
#---MAX_ITER:----------EM ALGORITHM WILL RUN FOR max_itr ITERATIONS UNTIL CONVERGENCE
 | DEFAULT: 100
#---N_INIT:------------NUMBER OF INITIALIZATIONS TO PERFORM, KEEPING ONLY THE BEST RESULTS
 | DEFAULT: 1
#---INIT_PARAMS:-------HOW TO INITIALIZE THE WEIGHTS, MEANS, AND VARIANCES
 | DEFAULT: 'kmeans'
#---WEIGHTS_INIT:------USER PROVIDES WEIGHTS FOR EACH CLUSTER MANUALLY
 | DEFAULT: init_params method
#---MEANS_INIT:--------USER PROVIDES MEANS FOR EACH CLUSTER MANUALLY
 | DEFAULT: init_params method
#---PRECISIONS_INIT:---USER PROVIDES INVERSES FOR THE COVARIANCE MATRICES MANUALLY
 | DEFAULT: init_params method

#PROVABLE (TWO-STEP) EM ALGORITHM
def provable_EM(data, k, l):
    #INITIALIZING PARAMETERS
    weights = [1./l, 1./l, 1./l, 1./l]
    means = data[np.random.choice(data.shape[0], l)]
    sigma = []
    cov = []
    for i in range(0, l):
        sigma.append(np.amin([np.linalg.norm(means[i] - x) for x in np.reshape(means[means
== means[i]], (l-1, 2))]))
    for i in range(0, l):
        cov.append(np.identity(2)*(1./sigma[i]))

    #FIRST PASS OF EM ALGORITHM
    GM = GaussianMixture(n_components=l, n_init=5, weights_init=weights, means_init=means,
precisions_init=cov).fit(data)
    weights = GM.weights_
    means = GM.means_
    prec = GM.precisions_

    #PRUNING
    pruned = [i for i in range(0, l) if weights[i] < 1./(4*l)]
    weights = np.delete(weights, pruned, 0)
    means = np.delete(means, pruned, 0)
    prec = np.delete(prec, pruned, 0)

    #COMPUTE THE NEW INITIAL VALUES
    S_weights = np.array([weights[0]])
    S_means = np.array([means[0]])
    S_prec = np.array([prec[0]])
    means = means = np.delete(means, 0, 0)
    while S_means.shape[0] < k:
        d = 0
```

```
"assignment10.py" 118L, 4757C                                  1,1           Top
```

```
...assignment10.py | ...t10 — λ — -bash | ...vim report.tex | ...xmk -pdf -pvc ...
```

```
                curr = mu
        index = np.where(means==mu)[0][0]
        S_weights = np.append(S_weights, [weights[index]], axis=0)
        S_means = np.append(S_means, [means[index]], axis=0)
        S_prec = np.append(S_prec, [prec[index]], axis=0)
        means = np.delete(means, index, 0)

    #NORMALIZE THE WEIGHTS
    S_weights = S_weights/np.sum(S_weights)

    #SECOND PASS OF EM ALGORITHM
    return GaussianMixture(n_components=k, n_init=5, weights_init=S_weights, means_init=S_means, precisions_init=S_prec).fit(data)

#MAIN BLOCK
def main():
    with open('./output/results.txt', 'w') as f:
        xeasy = np.load('./data_parsed/xeasy.data.npy')
        x1 = np.load('./data_parsed/x1.data.npy')
        x2 = np.load('./data_parsed/x2.data.npy')

        #ONE-STEP EM ALGORITHM
        print('ONE-STEP EM:', file=f)
        GMeasy = GaussianMixture(n_components=2, n_init=5).fit(xeasy)
        GM1 = GaussianMixture(n_components=2, n_init=5).fit(x1)
        GM2 = GaussianMixture(n_components=2, n_init=5).fit(x2)
        print_results(GMeasy, GM1, GM2, f)

        print('========================================', file=f)
        #TWO-STEP EM ALGORITHM
        print('TWO-STEP EM:', file=f)
        GMeasy = provable_EM(xeasy, 2, 4)
        GM1 = provable_EM(x1, 2, 4)
        GM2 = provable_EM(x2, 2, 4)
        print_results(GMeasy, GM1, GM2, f)

def print_results(GMeasy, GM1, GM2, f):
    print('xeasy', file=f)
    print('pi: ', file=f)
    print(GMeasy.weights_, file=f)
    print('mu: ', file=f)
    print(GMeasy.means_, file=f)
    print('sigma: ', file=f)
    print(GMeasy.covariances_, file=f)
    print('----------------------------------------', file=f)
    print('x1', file=f)
    print('pi: ', file=f)
    print(GM1.weights_, file=f)
    print('mu: ', file=f)
    print(GM1.means_, file=f)
    print('sigma: ', file=f)
    print(GM1.covariances_, file=f)
    print('----------------------------------------', file=f)
    print('x2', file=f)
    print('pi: ', file=f)
    print(GM2.weights_, file=f)
    print('mu: ', file=f)
    print(GM2.means_, file=f)
    print('sigma: ', file=f)
    print(GM2.covariances_, file=f)

#EXECUTE
main()
```

```
                                                    118,1         Bot
```

```python
# Daniel Gonzalez, FSU Mathematics PhD
# Colton Piper, FSU Mathematics PhD
# Applied Machine Learning Assignment 10

import numpy as np
import matplotlib.pyplot as plt

#GRAPH THE LOSS VS ITERATIONS
def graph(data, means, filename):
    #x_axis = range(1, 301)
    fig, ax = plt.subplots()
    ax.scatter(data[:,0], data[:,1])
    ax.scatter(means[:,0], means[:,1], label='Cluster Means')
    legend = ax.legend(loc='best')
    plt.title('Data set: ' + filename)
    plt.savefig('./report/figures/' + filename + '.eps', format='eps', dpi=1000, bbox_inches='tight')

def main():
    xeasy = np.load('./data_parsed/xeasy.data.npy')
    x1 = np.load('./data_parsed/x1.data.npy')
    x2 = np.load('./data_parsed/x2.data.npy')
    xeasy_mean = np.array([[0.02478088, 3.07460079], [3.01650103, -0.17460086]])
    x1_mean = np.array([[-0.28366821, 2.16960597], [1.83617659, 0.28263171]])
    x2_mean = np.array([[0.03484, -0.03989854], [0.17023148, -0.14255449]])

    graph(xeasy, xeasy_mean, 'xeasy')
    graph(x1, x1_mean, 'x1')
    graph(x2, x2_mean, 'x2')

main()
```