HW9_arkam2

April 5, 2021

1 STAT 542: Homework 9

1.1 ARKA MITRA (netid - arkam2)

1.1.1 Spring 2021, by Ruoqing Zhu (rqzhu)

1.1.2 Due: Tuesday, April 6, 11:59 PM CT

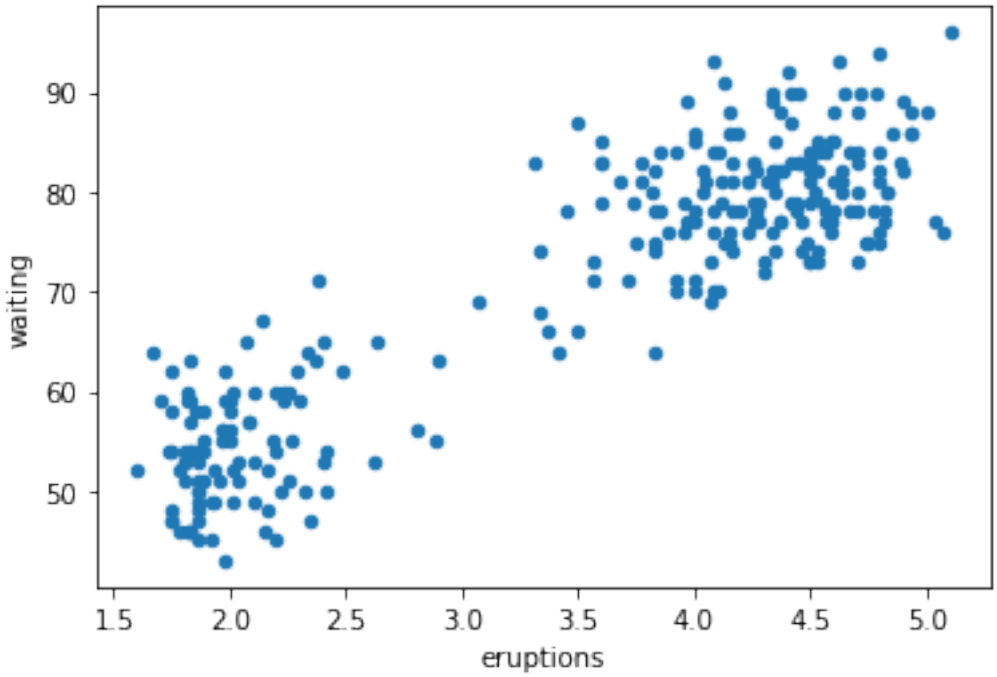
1.2 About HW9

Question 1 [100 Points] A Two-dimensional Gaussian Mixture Model In this homework we will extend the the Gaussian mixture model in the lecture note to a two-dimensional case, where both the mean and variance are unknown. Again, by using the EM algorithm, we face two steps, the E-step that calculates the conditional expectation of the likelihood, and the M-step that update the θ estimates. One nontrivial step is to derive analytic solution of θ in the M-step, which involves some matrix calculation and tricks. Some hints are provided. Finally, we will implement the method using our own code.

1.3 Question 1 [100 Points] A Two-dimensional Gaussian Mixture Model

If you do not use latex to type your answer, you will lose 2 points. We consider another example of the EM algorithm, which fits a Gaussian mixture model to the Old Faithful eruption data. The data is provided at the course website. For a demonstration (and partial solution) of this problem, see the figure provided on Wikipedia. As a result, we will use the formula to implement the EM algorithm and obtain the distribution parameters of the two underlying Gaussian distributions. Here is a visualization of the data:

[2]:
```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('faithful.txt', index_col = False, delim_whitespace=True)
df.plot.scatter(x = 'eruptions', y = 'waiting')
plt.show()
```



We use both variables eruptions and waiting. The plot above shows that there are two eruption patterns (clusters). Hence, we use a hidden Bernoulli random variable $Z_i \sim \text{Bern}(\pi)$ to indicate which pattern an observed eruption falls into. The corresponding distribution of eruptions and waiting can be described by a two-dimensional Gaussian — either $N(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$ or $N(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$ — depending on the outcome of Z_i. Here, the collection of parameters is $\boldsymbol{\theta} = \{\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1, \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2, \pi\}$, and we want to use the EM algorithm to estimate them.

1.4 Part a) (20 Points) The E-Step

Based on the above assumption of eruption patterns, write down the full log-likelihood $\ell(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta})$. In the E-step, we need the conditional expectation

$$g(\boldsymbol{\theta}|\boldsymbol{\theta}^{(k)}) = E_{\mathbf{Z}|\mathbf{x},\boldsymbol{\theta}^{(k)}}[\ell(\mathbf{x}, \mathbf{Z}|\boldsymbol{\theta})].$$

If you do not know where to start, then the answer is already provided on the Wikipedia page. Derive the conditional expectation (p_i) of $\mathbf{Z}$ given $\mathbf{x}$ and $\boldsymbol{\theta}^{(k)}$, using notations in our lecture.

Solution: The likelihood function is given by:

$$L(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta}) = \prod_{i=1}^{n} \left[\phi_{\mu_1,\Sigma_1}(\mathbf{x_i})\right]^{1-z_i} \left[\phi_{\mu_2,\Sigma_2}(\mathbf{x_i})\right]^{z_i} (1-\pi)^{1-z_i} (\pi)^{z_i}$$

where,

$$\phi_{\mu,\Sigma}(\mathbf{x_i}) = -\frac{1}{2}\log|\Sigma_1| - \frac{1}{2}(x_i-\mu_1)^T\Sigma_1^{-1}(x_i-\mu_1)$$

Hence, the log-likelihood is given by:

$$\begin{aligned}\ell(\mathbf{x},\mathbf{z}|\boldsymbol{\theta}) = &\sum_{i=1}^{n}(1-z_i)\left[-\frac{1}{2}\log|\Sigma_1| - \frac{1}{2}(x_i-\mu_1)^T\Sigma_1^{-1}(x_i-\mu_1)\right]\\ &+(z_i)\left[-\frac{1}{2}\log|\Sigma_2| - \frac{1}{2}(x_i-\mu_2)^T\Sigma_2^{-1}(x_i-\mu_2)\right]\\ &+\sum_{i=1}^{n}(1-z_i)\log(1-\pi) + z_i\log(\pi)\end{aligned}$$

Therefore, the E-Step function is :

$$\begin{aligned}g(\boldsymbol{\theta}|\boldsymbol{\theta}^{(k)}) &= E_{\mathbf{Z}|\mathbf{x},\boldsymbol{\theta}^{(k)}}[\ell(\mathbf{x},\mathbf{Z}|\boldsymbol{\theta})]\\ g(\boldsymbol{\theta}|\boldsymbol{\theta}^{(k)}) = &\sum_{i=1}^{n}(1-\hat{p}_i)\left[-\frac{1}{2}\log|\Sigma_1| - \frac{1}{2}(x_i-\mu_1)^T\Sigma_1^{-1}(x_i-\mu_1)\right]\\ &+(\hat{p}_i)\left[-\frac{1}{2}\log|\Sigma_2| - \frac{1}{2}(x_i-\mu_2)^T\Sigma_2^{-1}(x_i-\mu_2)\right]\\ &+\sum_{i=1}^{n}(1-\hat{p}_i)\log(1-\pi) + \hat{p}_i\log(\pi)\end{aligned} \tag{1}$$

where,

$$\hat{p}_i := P(\mathbf{Z}_i = 1|\mathbf{X}_i = \mathbf{x}_i, \boldsymbol{\theta}^{(k)}) = \frac{\pi\phi_{\mu_2,\Sigma_2}(x_i)}{\pi\phi_{\mu_2,\Sigma_2}(x_i) + (1-\pi)\phi_{\mu_1,\Sigma_1}(x_i)} \tag{2}$$

where the multivariate gaussian is $\phi_{\mu,\Sigma}(\cdot)$. PDF and the parameter set $\boldsymbol{\theta} = \{\boldsymbol{\mu}_1, \Sigma_1, \mu_2, \Sigma_2, \pi\}$ is evaluated using the $\boldsymbol{\theta}\{(k)\}$ iteration.

1.5 Part b) (30 Points) The M-Step

[10 points] Once we have $g(\boldsymbol{\theta}|\boldsymbol{\theta}^{(k)})$, the M-step is to re-calculate the maximum likelihood estimators of $\boldsymbol{\mu}_1$, $\boldsymbol{\Sigma}_1$, $\boldsymbol{\mu}_2$, $\boldsymbol{\Sigma}_2$ and π. Again the answer was already provided on Wikipedia. However, you need to provide a derivation of these estimators. This is essentially taking derivatives of the objective function with respect to the parameters, and then set them to zero to solve for the optimal solution. The derivation involves three tricks:

$\text{Trace}(\beta^T\Sigma^{-1}\beta) = \text{Trace}(\Sigma^{-1}\beta\beta^T)$

$\frac{\partial}{\partial A}\log|A| = A^{-1}$

$\frac{\partial}{\partial A}\text{Trace}(BA) = B^T$

Please note that it is easier to take the derivative with respect to Σ^{-1} rather than Σ.

Solution: Starting with $g(\boldsymbol{\theta}|\boldsymbol{\theta}^{(k)})$, and we want to find the parameter estimates such that:

$$\theta^{(k+1)} = \arg\max_{\theta} g(\boldsymbol{\theta}|\boldsymbol{\theta}^{(k)})$$

Hence, setting $\frac{\partial g}{\partial \theta_i} = 0$ for each $\theta_i \in \boldsymbol{\theta}$, and thus,

For π:

$$\frac{\partial g}{\partial \pi} = 0$$

$$\sum_{i=1}^{n} \frac{\hat{p}_i}{\pi} - \frac{1-\hat{p}_i}{1-\pi} = 0$$

Hence,

$$\pi^{(k+1)} = \frac{1}{n}\sum_{i=1}^{n} \hat{p}_i \tag{3}$$

For μ_1 and μ_2:

$$\frac{\partial g}{\partial \mu_1} = 0$$

$$\sum_{i=1}^{n}(1-\hat{p}_i)\left(-\frac{1}{2}\left(-2\Sigma_1^{-1}(x_i-\mu_1)\right)\right) = 0$$

$$\sum_{i=1}^{n}(1-\hat{p}_i)\Sigma_1^{-1}(x_i-\mu_1) = 0$$

Which gives

$$\mu_1^{(k+1)} = \frac{\sum_{i=1}^{n}(1-\hat{p}_i)x_i}{\sum_{i=1}^{n}(1-\hat{p}_i)}$$

Also,

$$\mu_2^{(k+1)} = \frac{\sum_{i=1}^{n}\hat{p}_i x_i}{\sum_{i=1}^{n}\hat{p}_i} \tag{4}$$

For Σ_1 and Σ_2:

$$\frac{\partial g}{\partial \Sigma_1} = 0$$

$$\sum_{i=1}^{n}(1-\hat{p}_i)\left(-\frac{1}{2}\Sigma_1^{-1} - \frac{1}{2}\Sigma_1^{-1}(x_i-\mu_1)(x_i-\mu_1)^T\Sigma_1^{-1}\right) = 0$$

$$\sum_{i=1}^{n}(1-\hat{p}_i)\left(I - (x_i-\mu_1)(x_i-\mu_1)^T\Sigma_1^{-1}\right) = 0$$

Thus,

$$\Sigma_1^{(k+1)} = \frac{\sum_{i=1}^{n}(1-\hat{p}_i)(x_i-\mu_1)(x_i-\mu_1)^T}{\sum_{i=1}^{n}(1-\hat{p}_i)}$$

Also,

$$\Sigma_2^{(k+1)} = \frac{\sum_{i=1}^{n}\hat{p}_i(x_i-\mu_2)(x_i-\mu_2)^T}{\sum_{i=1}^{n}\hat{p}_i} \tag{5}$$

2 Part c) (50 Points) Implementing the Algorithm

Implement the EM algorithm using the formula you just derived. Make sure that the following are addressed:

[5 Points] You need to give a reasonable initial value such that the algorithm converges.

[10 Points] Make sure that you give proper comment on each step to clearly indicate which quantity the code is calculating.

[5 Points] Set up a convergence criteria under which the iteration stops.

[10 Points] Record the result (all the parameter estimates) for each iteration. Report the final parameter estimates.

[10 Points] Make four plots to demonstrate the fitted model and the updating process: your initial values, the first iteration, the second iteration, and the final results. The plots should intuitively demonstrate the fitted Gaussian distributions. For ideas of the plot, refer to the animation on the Wikipedia page.

You may use other packages to calculate the Gaussian densities. There could be numerical difficulties when you try to calculate them using your own code.

2.0.1 Solution:

All points are addressed in the code, except point 3. The convergence criterion used is the difference in log-likelihood between two iterations:

$$E_{\mathbf{Z}|\mathbf{x},\boldsymbol{\theta}^{(k)}}[\ell(\mathbf{x},\mathbf{Z}|\boldsymbol{\theta})] \leq E_{\mathbf{Z}|\mathbf{x},\boldsymbol{\theta}^{(k-1)}}[\ell(\mathbf{x},\mathbf{Z}|\boldsymbol{\theta})] + \epsilon$$

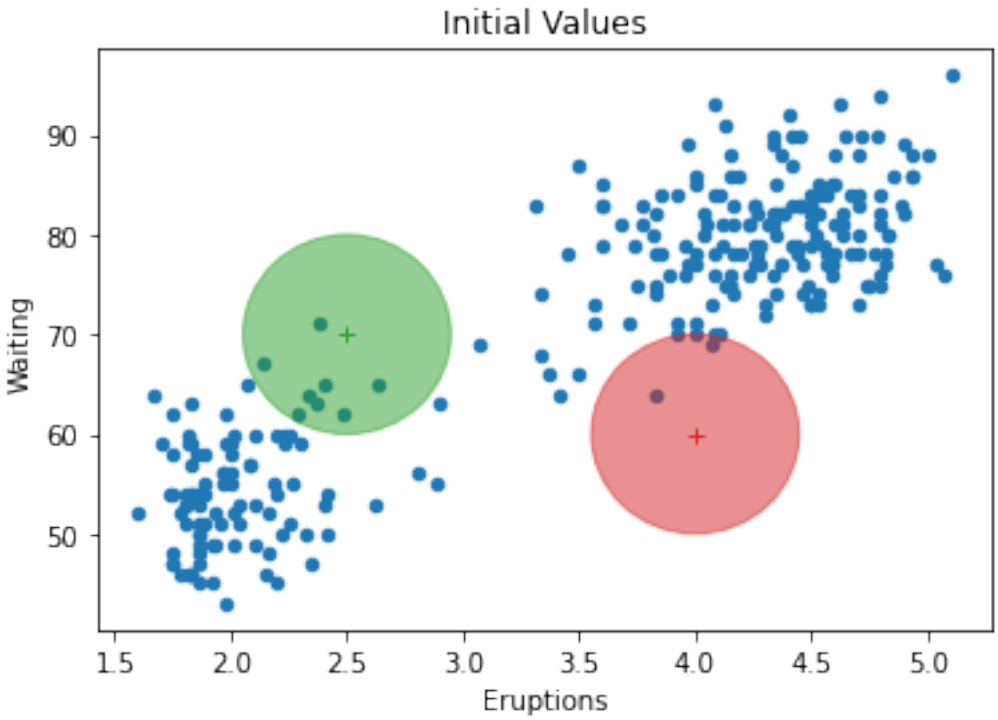
for some pre-defined tolerance ϵ.

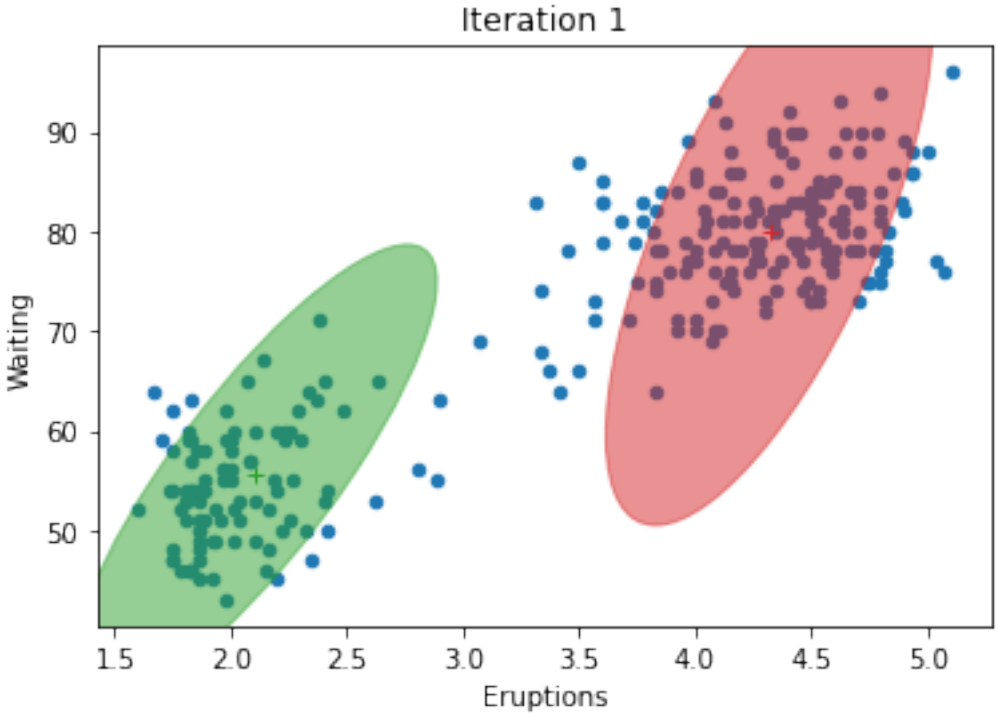
[6]:
```
import math
import itertools
import matplotlib
import numpy as np
from scipy import linalg
from IPython.display import display, Latex
from scipy.stats import multivariate_normal as mvrnorm
df = pd.read_csv('faithful.txt', index_col = False, delim_whitespace=True)

def emAlgo(mu1, mu2, sig1, sig2, pi, x, tol = 1e-05, maxitr = 100):
    def getP_iHat(mu1, mu2, sig1, sig2, pi, x):
        '''
        Equation 2:
        Takes in a vector of points x (n x 2) and returns a vector p (n x 1)
        '''
        num = pi * mvrnorm.pdf(x, mu2, sig2)
        den = num + (1 - pi) * mvrnorm.pdf(x, mu1, sig1)
        return num/den
```

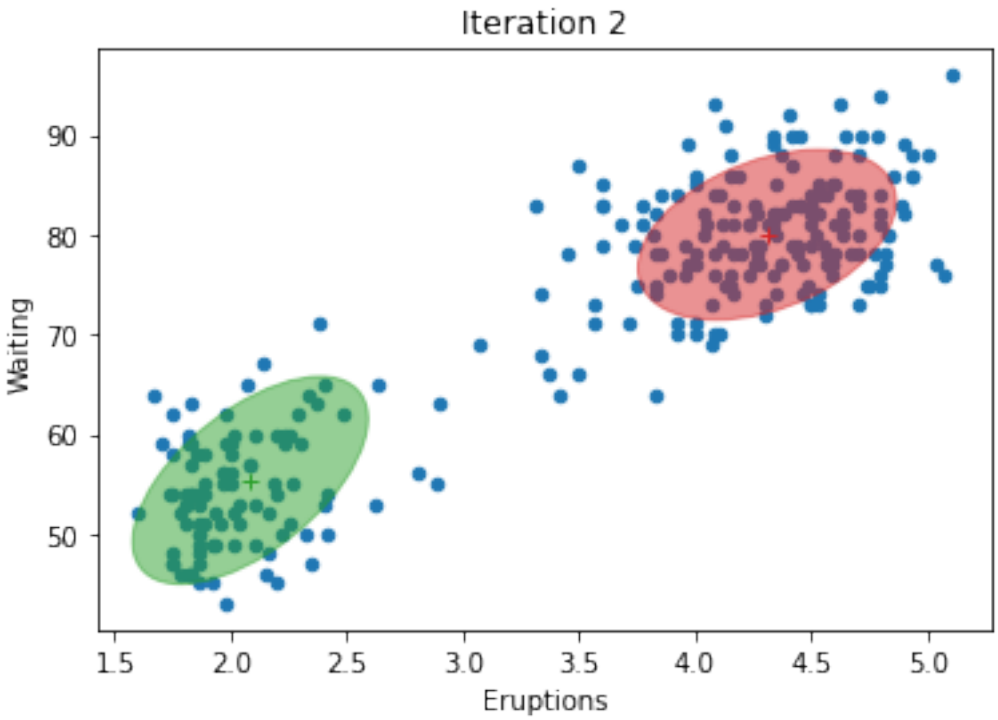
```
    def logL(mu1, mu2, sig1, sig2, pi, x, p_i):
        '''
        Equation 1:
        Takes in a vector of points x (n x 2) and returns the log-likelihood␣
↪(scalar)
        '''
        p_i_dash = 1 - p_i
        sig1_inv = np.linalg.inv(sig1)
        sig2_inv = np.linalg.inv(sig2)
        logsig1 = math.log(np.linalg.det(sig1))
        logsig2 = math.log(np.linalg.det(sig2))
        sum1 = np.sum([p_i_dash[j] * (-0.5*logsig1 - 0.5*(x[j] -mu1).
↪dot(sig1_inv.dot(x[j]-mu1))) for j in range(len(x))])
        sum2 = np.sum([p_i[j] * (-0.5*logsig2 - 0.5*(x[j] -mu2).dot(sig2_inv.
↪dot(x[j]-mu2))) for j in range(len(x))])
        sum3 = np.sum(math.log(1 - pi)*p_i_dash)
        sum4 = np.sum(math.log(pi)*p_i)
        return sum1 + sum2 + sum3 + sum4

    def getMuUpdate(p_i, x):
        '''
        mu update based on Equation 4
        '''
        return np.sum(np.multiply(p_i.reshape(-1,1), x), axis = 0) / np.sum(p_i)

    def getSigmaUpdate(p_i, x, mu):
        '''
        Does the following:
        (1) sigma update based on equation 5
        (2) E-step:
        Calculates the conditional distribution of z
        (3) M-Step:
        Parameter Updates
        '''
        # Sigma update
        s = [p_i[j] * (x[j] - mu).reshape(-1,1).dot((x[j] - mu).reshape(1,-1))␣
↪for j in range(len(x))]
        return np.sum(s, axis = 0) / np.sum(p_i)

    results = {}
    for i in range(maxitr):
        results[i] = (mu1, mu2, sig1, sig2, pi)
        '''

        '''
        # (2) E-Step
        p_i = getP_iHat(mu1, mu2, sig1, sig2, pi, x)
```

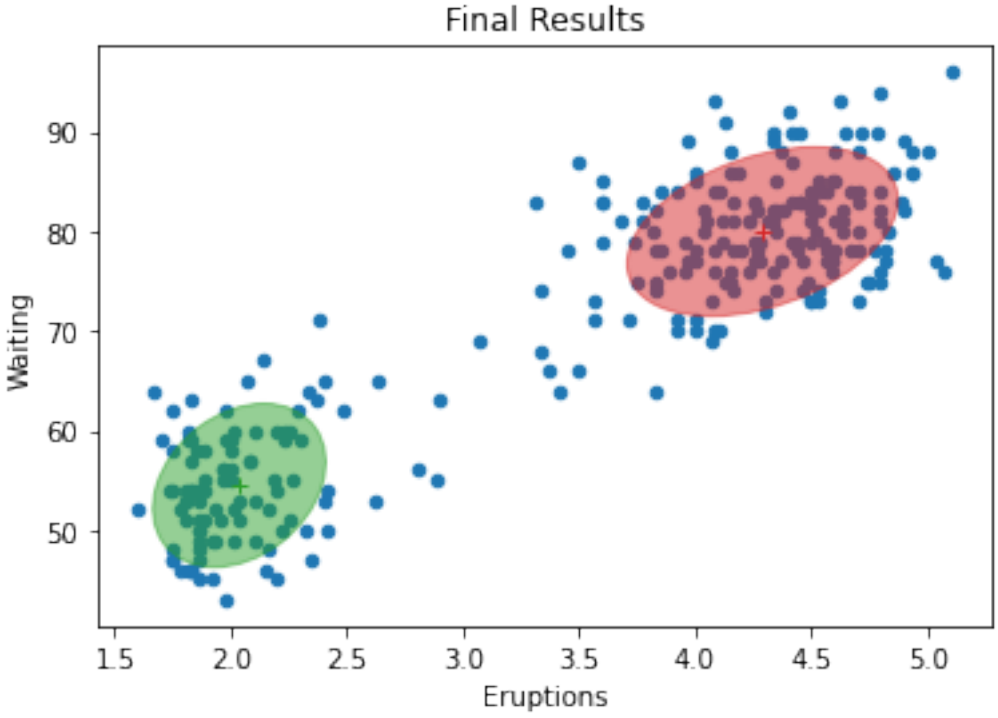
```
        # (3) M-Step
        pi_new             = np.mean(p_i)
        mu1_new, mu2_new   = getMuUpdate(1 - p_i, x) , getMuUpdate(p_i, x)
        sig1_new, sig2_new = getSigmaUpdate(1 - p_i, x, mu1) ,␣
↪getSigmaUpdate(p_i, x, mu2)

        if abs(logL(mu1, mu2, sig1, sig2, pi, x, p_i) - \
               logL(mu1_new, mu2_new, sig1_new, sig2_new, pi_new, x, p_i)) < tol:
            break
        else:
            pi = pi_new
            mu1, mu2 = mu1_new, mu2_new
            sig1, sig2 = sig1_new, sig2_new

    return results

def plot_results(X, means, covariances, title):
    '''
     Make four plots to demonstrate the fitted model
     and the updating process: your initial values,
     the first iteration, the second iteration, and
     the final results.
    '''
    color_iter = itertools.cycle(['tab:green','tab:red'])

    fig = plt.subplot(1,1,1)
    plt.scatter(X[:,0], X[:,1], s = 20)
    for i, (mean, covar, color) in enumerate(zip(means, covariances,␣
↪color_iter)):
        v, w = linalg.eigh(covar)
        v = 2. * np.sqrt(2.) * np.sqrt(v)
        u = w[0] / linalg.norm(w[0])
        plt.plot(means[i,0],means[i,1],'+', color = color)

        # Plot an ellipse to show the Gaussian component
        angle = np.arctan(u[1] / u[0])
        angle = 180. * angle / np.pi  # convert to degrees
        ellipse = matplotlib.patches.Ellipse(mean, v[0], v[1], 180. + angle,␣
↪color=color)
        ellipse.set_clip_box(fig.bbox)
        ellipse.set_alpha(0.5)
        fig.add_artist(ellipse)

    plt.xlabel ('Eruptions')
    plt.ylabel ('Waiting')
    fig = matplotlib.pyplot.gcf()
```

```
    fig.set_size_inches(6,4)
    plt.title(title)
    plt.show()

#Initial Values
mu1 = np.array([2.5, 70])
mu2 = np.array([4, 60])
sig1 = np.array([[0.1, 0],[0, 50]])
sig2 = np.array([[0.1, 0],[0, 50]])
df = df.drop(['index'],axis=1)
x = df.values
pi = 0.5

resultsDict = emAlgo(mu1, mu2, sig1, sig2, pi, x)
finalItr = list(resultsDict.keys())[-1]
#Iterations whose results you want to plot
plotIdx = [0, 1, 2, finalItr]
titles = ['Initial Values'] + ['Iteration '+str(k) for k in␣
 ↪range(1,len(plotIdx)-1)] + ['Final Results']

for idx, title in zip(plotIdx, titles):
    currItr = resultsDict[idx]
    means = np.append(currItr[0].reshape(1,-1), currItr[1].reshape(1,-1), axis =␣
 ↪0)
    covariances = np.append(currItr[2].reshape(2,-2,), currItr[3].
 ↪reshape(2,-2,), axis = 0).reshape(2,2,2)
    plot_results(x, means, covariances, title)
```


Initial Values
Waiting
Eruptions
90
80
70
60
50
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0


Iteration 1
Waiting
Eruptions
90
80
70
60
50
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0


Iteration 2
Waiting
Eruptions
90
80
70
60
50
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0


Final Results
Waiting
Eruptions
90
80
70
60
50
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

[50]:
```
print('Reporting final parameter estimates :')
display(Latex(r'$\mu_1$ = '))
print(resultsDict[7][0])
display(Latex(r'$\Sigma_1$ = '))
print(resultsDict[7][2])
display(Latex(r'$\mu_2$ = '))
print(resultsDict[7][1])
display(Latex(r'$\Sigma_1$ = '))
print(resultsDict[7][3])
```

```
Reporting final parameter estimates :
```

μ_1 =

```
[ 2.03641548 54.47878845]
```

Σ_1 =

```
[[ 0.06918915  0.4353919 ]
 [ 0.4353919  33.69881466]]
```

μ_2 =

```
[ 4.28968588 79.96840421]
```

Σ_1 =

```
[[ 0.1699381   0.94022358]
 [ 0.94022358 36.04187046]]
```

[ ]: