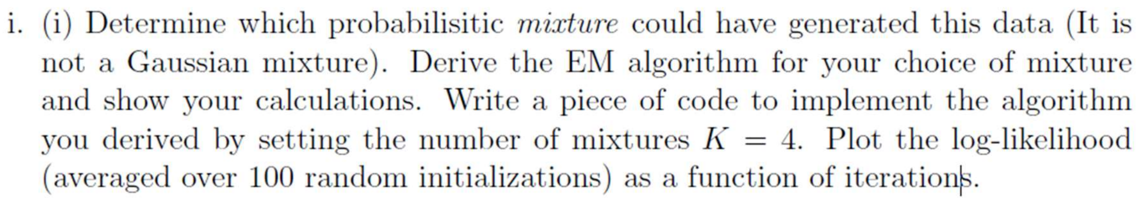
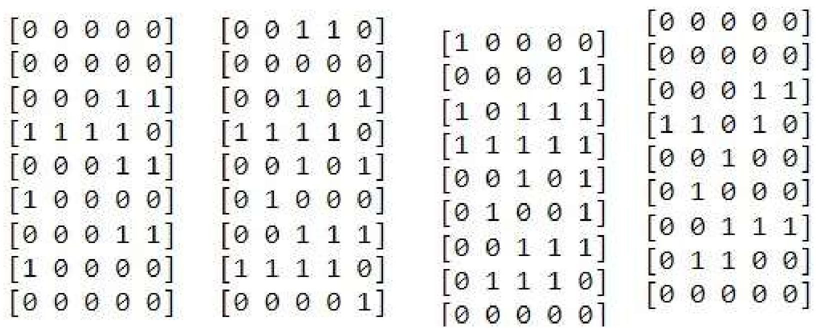
CS-5691 Assignment 2

Question 1:


i. (i) Determine which probabilisitic *mixture* could have generated this data (It is not a Gaussian mixture). Derive the EM algorithm for your choice of mixture and show your calculations. Write a piece of code to implement the algorithm you derived by setting the number of mixtures $K = 4$. Plot the log-likelihood (averaged over 100 random initializations) as a function of iterations.

Visualizing the datapoints (as 10*5 images)


```
[0 0 0 0 0]  [0 0 1 1 0]  [1 0 0 0 0]  [0 0 0 0 0]
[0 0 0 0 0]  [0 0 0 0 0]  [0 0 0 0 1]  [0 0 0 0 0]
[0 0 0 1 1]  [0 0 1 0 1]  [1 0 1 1 1]  [0 0 0 1 1]
[1 1 1 1 0]  [1 1 1 1 0]  [1 1 1 1 1]  [1 1 0 1 0]
[0 0 0 1 1]  [0 0 1 0 1]  [0 0 1 0 1]  [0 0 1 0 0]
[1 0 0 0 0]  [0 1 0 0 0]  [0 1 0 0 1]  [0 1 0 0 0]
[0 0 0 1 1]  [0 0 1 1 1]  [0 0 1 1 1]  [0 0 1 1 1]
[1 0 0 0 0]  [1 1 1 1 0]  [0 1 1 1 0]  [0 1 1 0 0]
[0 0 0 0 0]  [0 0 0 0 1]  [0 0 0 0 0]  [0 0 0 0 0]
```

Answer 1a)
As per the hint given in the question that each data point is a 10*5 matrix, we can find the problem analogous to image classification.

For problems relating to forming image clusters, the latent class analysis model is often preferred. It is an example of the Expectation Maximization algorithm over mixtures of discrete random variables.

The distribution must be a multivariate Bernoulli in d dimensions (here, d = 50).

The data points have 50 features each containing a 0 or a 1, analogous to a problem of coin tossing. A similar distribution is often observed when using Naive Bayes' for classification (spam classification). Each feature represents a coin toss (A Bernoulli Random Variable), and each data point can be expressed as an event of 50-coin tosses occurring simultaneously.

The equation for the multivariate Bernoulli distribution can be given as follows:

$$P(x^{(i)}, k) = \Pi_{i=0}^{n} \left((p_k^i)^{x_d^{(i)}} * (1 - p_k^i)^{(1-x_d^{(i)})} \right)$$

The data points represent images of the shape of 10*5, some of which are printed out below as bitmaps.

$$p_k^d = \frac{\sum_{i=1}^{n} \lambda_k^i f_d^i}{\sum_{i=1}^{n} \lambda_k^i}$$

$$\pi_k = \frac{\sum_{i=1}^{n} \lambda_k^i}{n}$$

$$\lambda_k^i = \frac{\pi_k \prod_{d=1}^{50} (p_k^d)^{f_d^i} (1 - p_k^d)^{1-f_d^i}}{\sum_{l=1}^{k} \pi_l \prod_{d=1}^{50} (p_k^d)^{f_d^i} (1 - p_k^d)^{1-f_d^i}}$$

The Expectation Maximization Algorithm (Dempster Etal ’77) has the following steps:

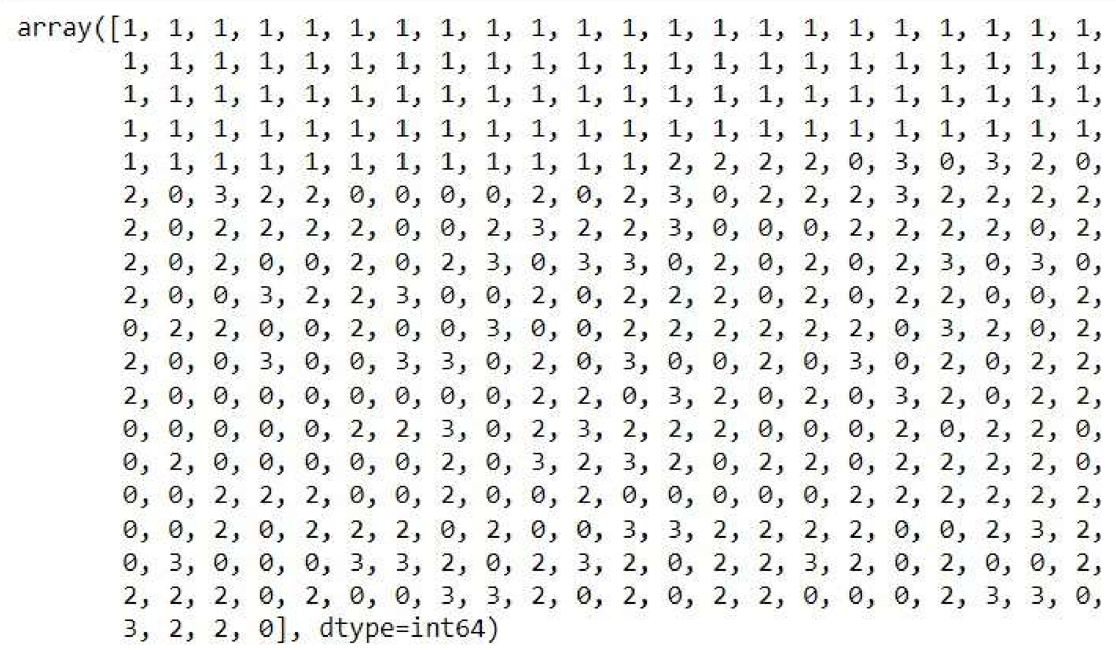
1. Initialize using Lloyd’s algorithm
2. Until Convergence ()
3. Find Expectation of the responsibilities ()
4. Maximize the parameters keeping the responsibilities constant

The parameters for this distribution are pie (4) and the p value (50*4) Here each p is 50 dimensional and there are 4 of each. There are 4 pie values corresponding to the weightage of each mixture.

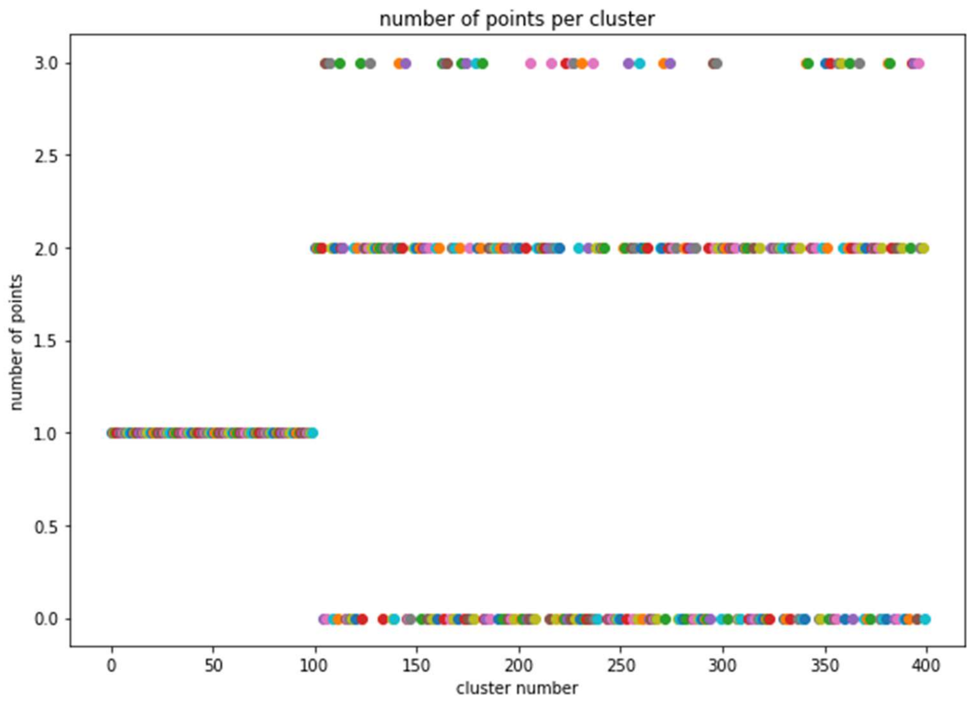
We first calculate the Likelihood of the Mixture of Bernoulli:

Considering the latent variables inside the likelihood:

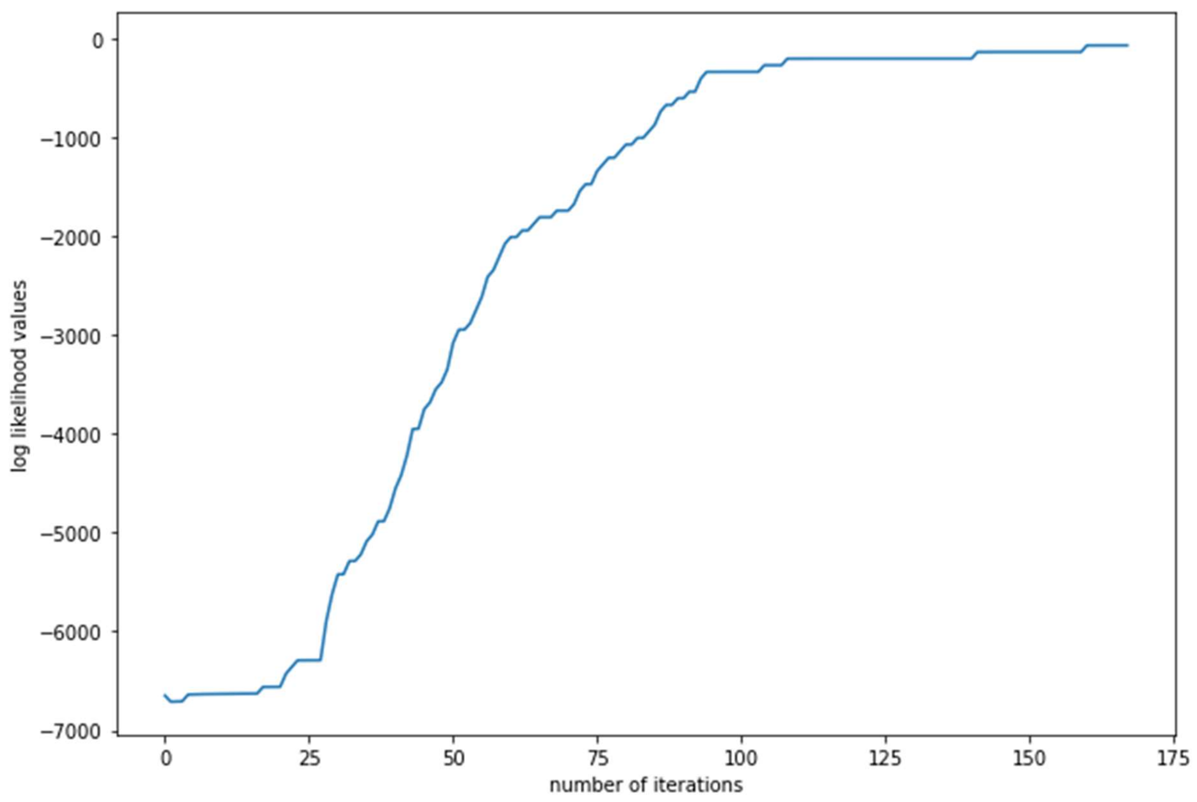
Now, we take the logarithm of the likelihood:



```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 0, 3, 0, 3, 2, 0,
       2, 0, 3, 2, 2, 0, 0, 0, 0, 2, 0, 2, 3, 0, 2, 2, 2, 3, 2, 2, 2, 2,
       2, 0, 2, 2, 2, 2, 0, 0, 2, 3, 2, 2, 3, 0, 0, 0, 2, 2, 2, 2, 0, 2,
       2, 0, 2, 0, 0, 2, 0, 2, 3, 0, 3, 3, 0, 2, 0, 2, 0, 2, 3, 0, 3, 0,
       2, 0, 0, 3, 2, 2, 3, 0, 0, 2, 0, 2, 2, 2, 0, 2, 0, 2, 2, 0, 0, 2,
       0, 2, 2, 0, 0, 2, 0, 0, 3, 0, 0, 2, 2, 2, 2, 2, 2, 0, 3, 2, 0, 2,
       2, 0, 0, 3, 0, 0, 3, 3, 0, 2, 0, 3, 0, 0, 2, 0, 3, 0, 2, 0, 2, 2,
       2, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 0, 3, 2, 0, 2, 0, 3, 2, 0, 2, 2,
       0, 0, 0, 0, 0, 2, 2, 3, 0, 2, 3, 2, 2, 2, 0, 0, 0, 2, 0, 2, 2, 0,
       0, 2, 0, 0, 0, 0, 0, 2, 0, 3, 2, 3, 2, 0, 2, 2, 0, 2, 2, 2, 2, 0,
       0, 0, 2, 2, 2, 0, 0, 2, 0, 0, 2, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2,
       0, 0, 2, 0, 2, 2, 2, 0, 2, 0, 0, 3, 3, 2, 2, 2, 2, 0, 0, 2, 3, 2,
       0, 3, 0, 0, 0, 3, 3, 2, 0, 2, 3, 2, 0, 2, 2, 3, 2, 0, 2, 0, 0, 2,
       2, 2, 2, 0, 2, 0, 0, 3, 3, 2, 0, 2, 0, 2, 2, 0, 0, 0, 2, 3, 3, 0,
       3, 2, 2, 0], dtype=int64)
```



Now, the logarithm is a concave function. Therefore, we can make use of the Jensen's inequality to come to the following simplification – (modified log likelihood):

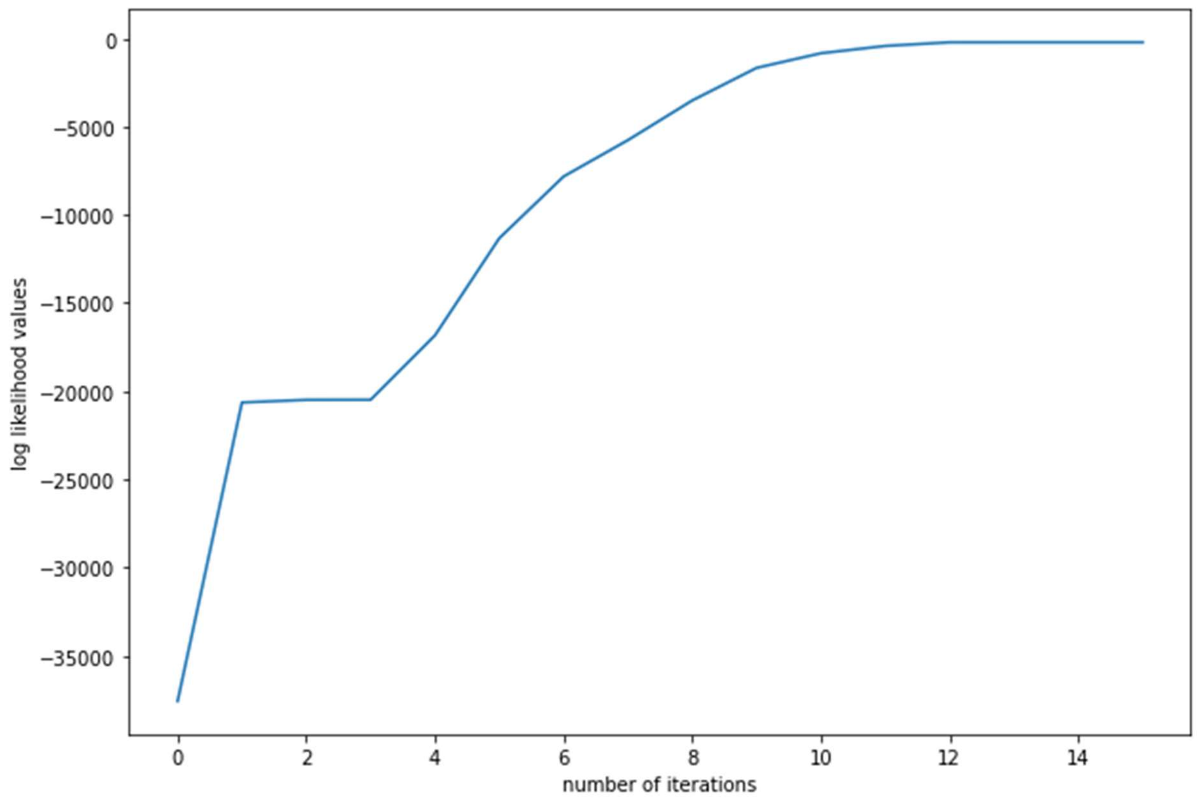


The Expectation step of the EM algorithm for the Multivariate Bernoulli can be derived as follows:

ii. (ii) Assume that the same data was infact generated from a mixture of Gaussians with 4 mixtures. Implement the EM algorithm and plot the log-likelihood (averaged over 100 random initializations of the parameters) as a function of iterations. How does the plot compare with the plot from part (i)? Provide insights that you draw from this experiment.

The update equations for the mean, variance_covariance have to be changed as compared to part (i) where we use the Multivariate Bernoulli distribution.

The omega and lambda (responsibility) value updations will be similar to the previous parts

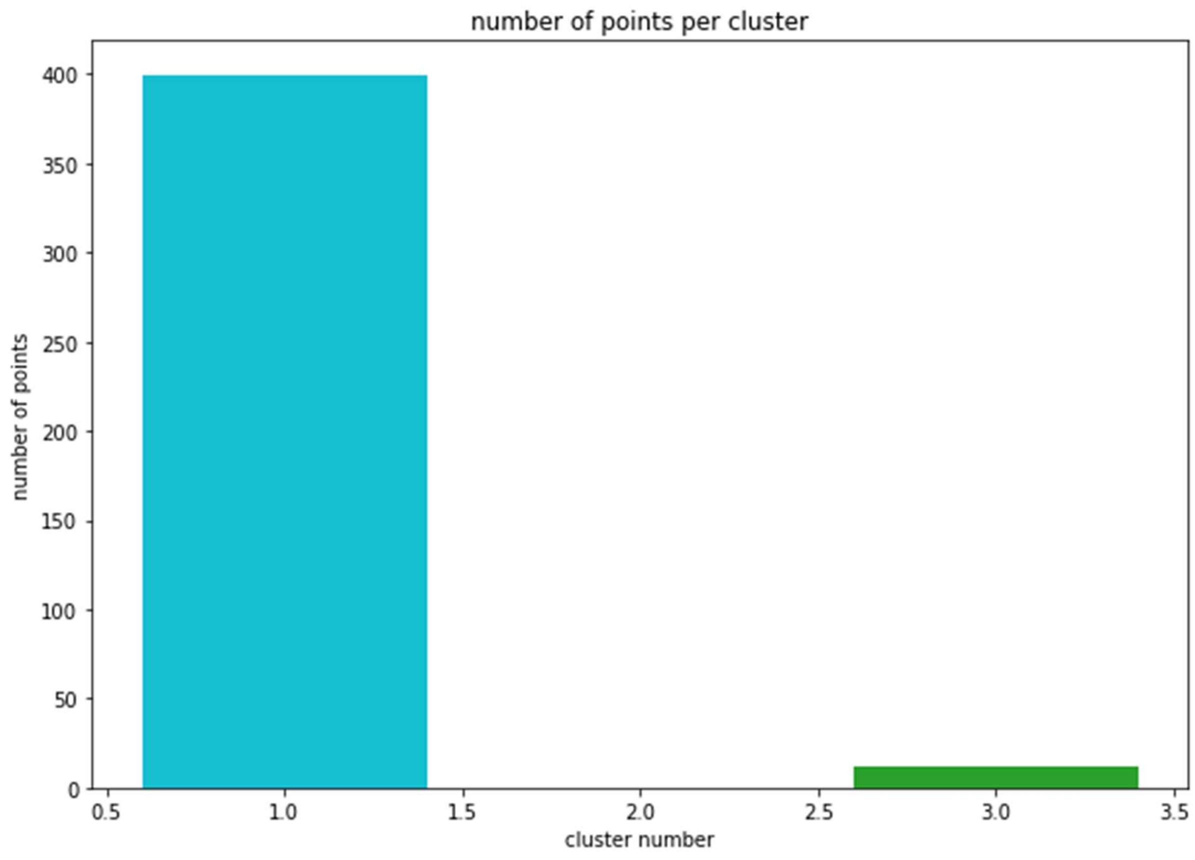


The log likelihood function in here converges a lot quicker than the log likelihood of the Bernoulli.

As we can see from the resultant assignment matrix (hard clustered), after the running the EM algorithm becomes the following:

```
array([1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 3, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1], dtype=int64)
```

The bar graph for the number of points belonging to each cluster is given below:



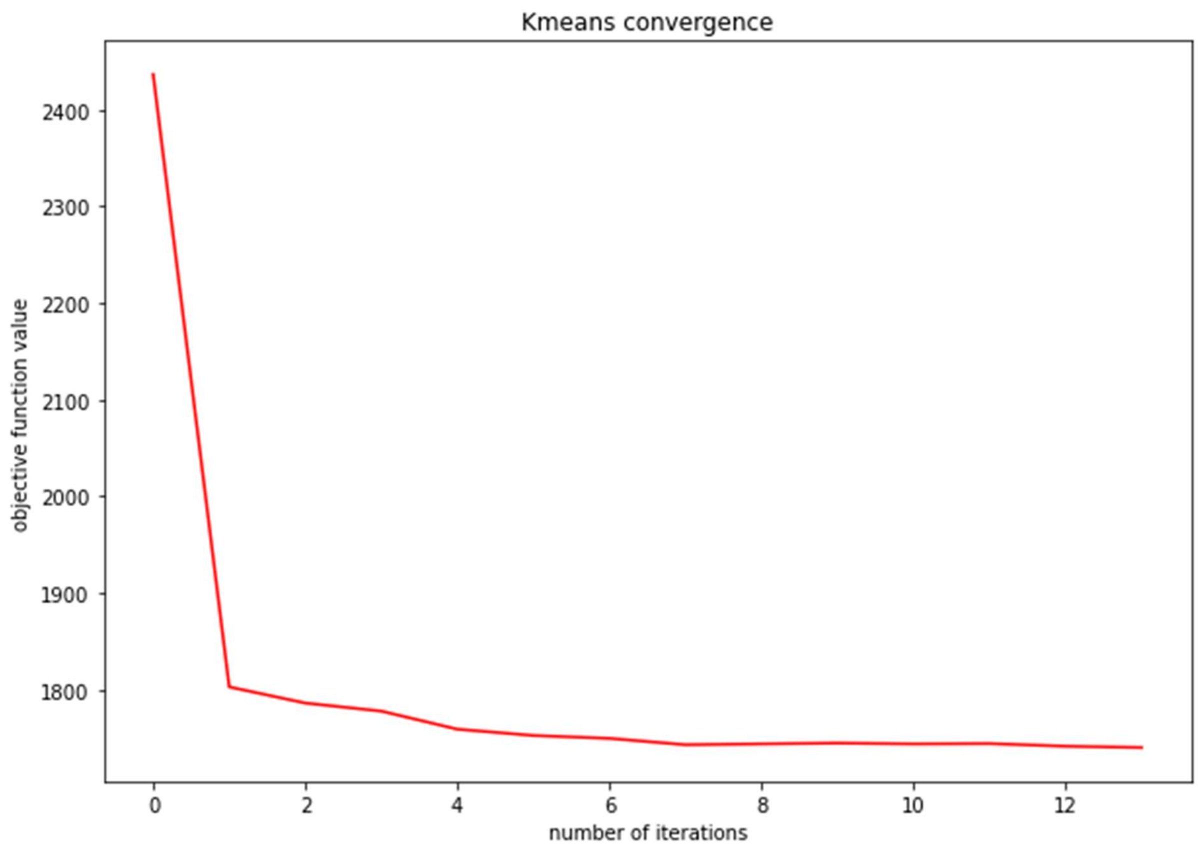
As we can see, a maximum number of points end up being assigned to one of the Gaussians while the others getting almost none to none values.

Thus, the problem converges to saying that all the points in the dataset are generated by one Gaussian distribution. (multivariate).

This is much unlike the Bernoulli where the points go into the clusters they belong.

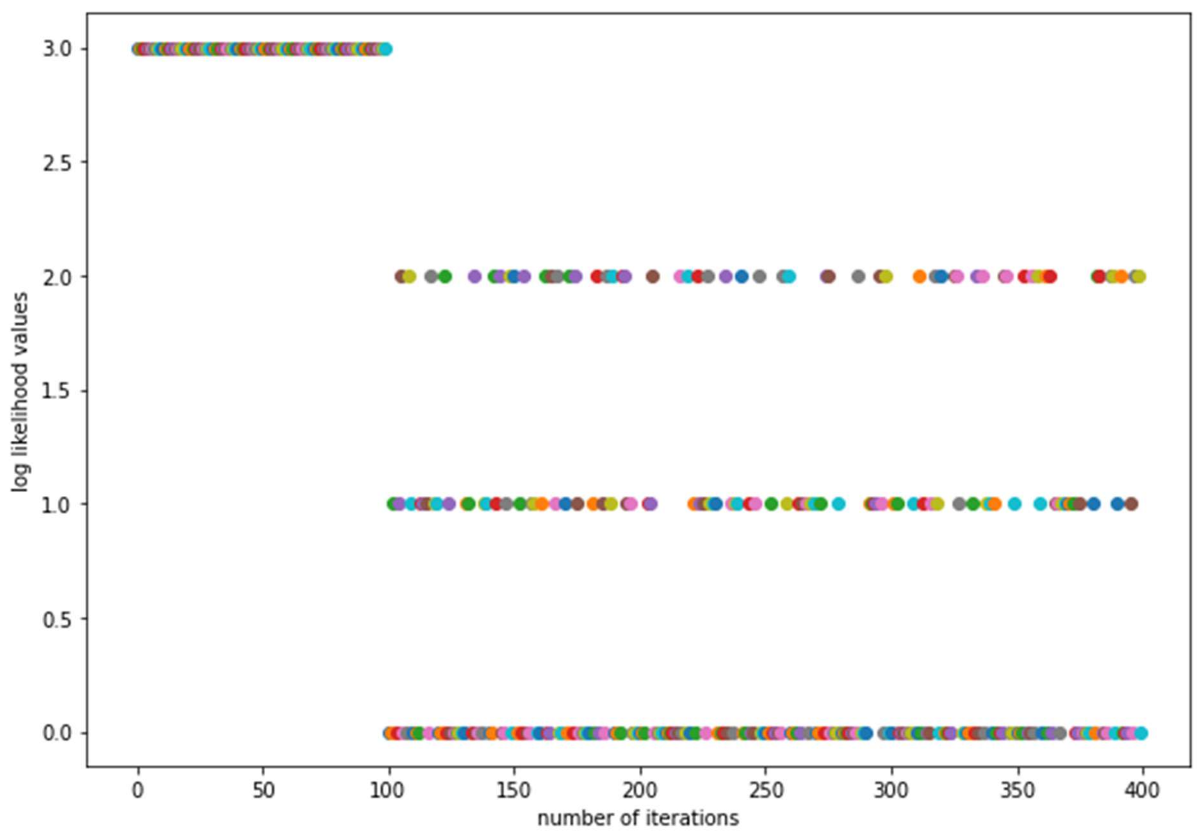
iii. Run the K-means algorithm with $K = 4$ on the same data. Plot the objective of $K - means$ as a function of iterations.

On running the K means algorithm with K = 4 on the data set, it converges on an average within 8-16 iterations.



The cluster assignments upon convergence of the K means can be shown:

```
array([3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 0, 1, 0, 1, 2, 0, 0, 2, 1,
       0, 0, 0, 1, 1, 1, 0, 2, 1, 1, 0, 0, 2, 0, 1, 0, 0, 0, 0, 0, 0, 1,
       1, 0, 2, 0, 0, 0, 1, 1, 0, 0, 2, 1, 2, 0, 0, 1, 2, 0, 2, 0, 1, 0,
       2, 0, 0, 1, 1, 0, 0, 1, 2, 0, 0, 2, 1, 2, 0, 0, 1, 0, 2, 0, 2, 1,
       0, 0, 0, 0, 0, 1, 0, 2, 0, 1, 0, 2, 1, 2, 0, 0, 0, 2, 2, 1, 1, 0,
       0, 0, 0, 0, 0, 1, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 2,
       0, 1, 0, 2, 1, 1, 0, 2, 1, 1, 1, 0, 0, 0, 2, 0, 1, 0, 1, 1, 2, 0,
       0, 1, 0, 0, 1, 2, 0, 0, 0, 0, 1, 0, 0, 0, 0, 2, 1, 2, 0, 0, 0, 1,
       0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 2, 2, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 2, 0, 0, 0, 1, 1, 1, 1, 2, 1, 0, 2, 0, 0, 1, 1, 0, 0, 0, 0, 0,
       0, 1, 0, 2, 0, 1, 0, 0, 1, 2, 1, 0, 2, 0, 0, 0, 0, 2, 2, 1, 0, 0,
       0, 0, 1, 0, 2, 0, 2, 0, 1, 1, 0, 1, 0, 0, 0, 2, 2, 0, 0, 1, 0, 0,
       0, 2, 0, 0, 2, 0, 2, 1, 0, 2, 0, 2, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0,
       0, 1, 0, 0, 0, 0, 1, 0, 2, 2, 0, 0, 0, 2, 2, 0, 1, 2, 0, 0, 0, 1,
       0, 2, 2, 0])
```

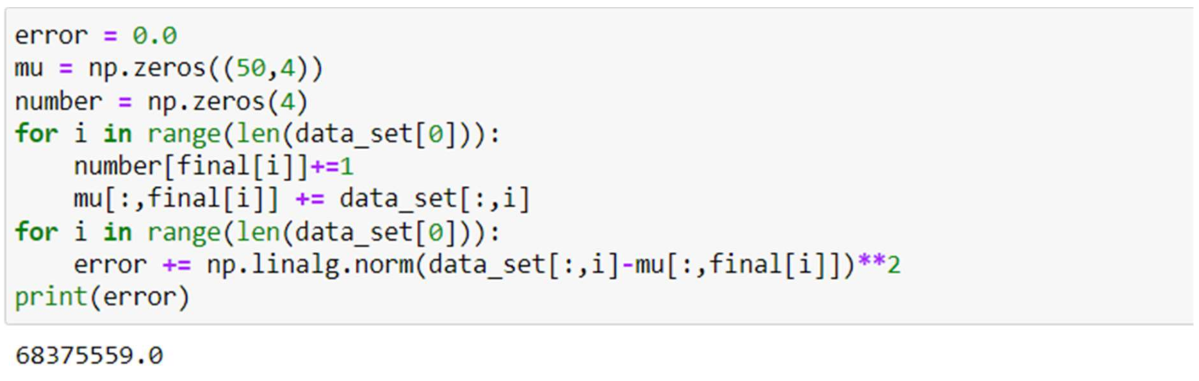


iv. Among the three different algorithms implemented above, which do you think you would choose to for this dataset and why?

We design objective functions to compare the quality of the clusters formed. We use the same objective function as was used in Lloyd's algorithm to get an estimate.

The code for the calculations are appended to the bottom of the .ipynb files and their images are shown below:

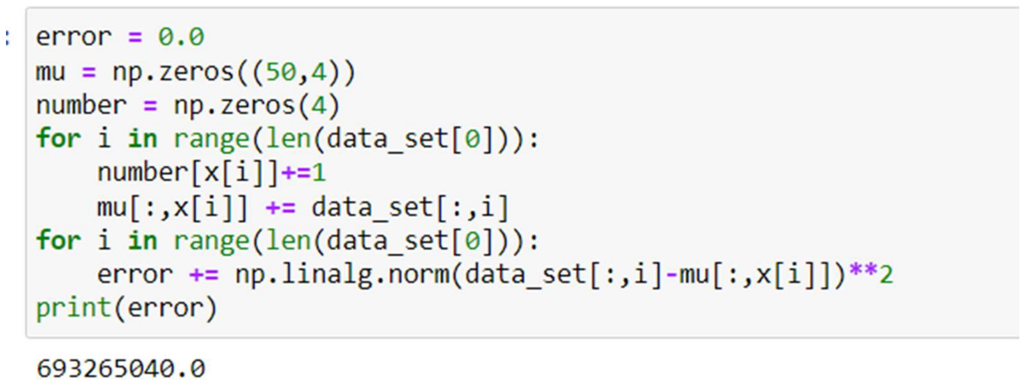
For the mixture of Bernoulli's:



```
error = 0.0
mu = np.zeros((50,4))
number = np.zeros(4)
for i in range(len(data_set[0])):
    number[final[i]]+=1
    mu[:,final[i]] += data_set[:,i]
for i in range(len(data_set[0])):
    error += np.linalg.norm(data_set[:,i]-mu[:,final[i]])**2
print(error)
```

```
68375559.0
```

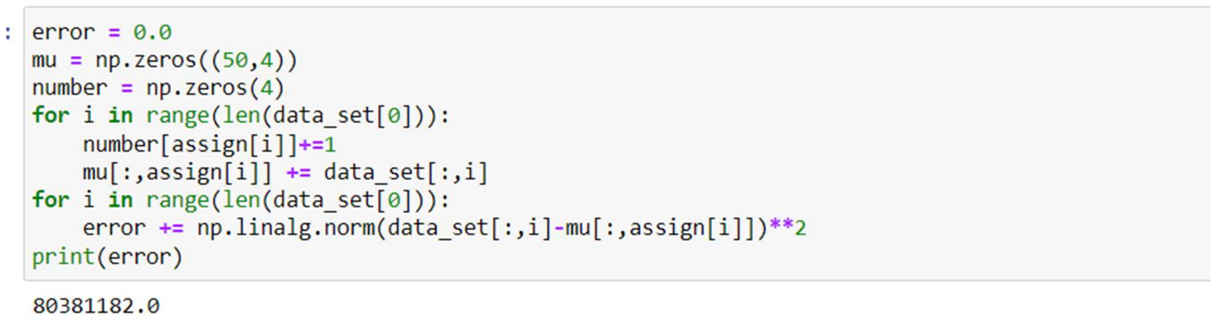
For the mixture of Gaussian's:



```
error = 0.0
mu = np.zeros((50,4))
number = np.zeros(4)
for i in range(len(data_set[0])):
    number[x[i]]+=1
    mu[:,x[i]] += data_set[:,i]
for i in range(len(data_set[0])):
    error += np.linalg.norm(data_set[:,i]-mu[:,x[i]])**2
print(error)
```

```
693265040.0
```

For the K means:



```
error = 0.0
mu = np.zeros((50,4))
number = np.zeros(4)
for i in range(len(data_set[0])):
    number[assign[i]]+=1
    mu[:,assign[i]] += data_set[:,i]
for i in range(len(data_set[0])):
    error += np.linalg.norm(data_set[:,i]-mu[:,assign[i]])**2
print(error)
```

```
80381182.0
```

The mixture of Bernoulli's gives the lowest objective value, followed by the K means and then followed by the mixture of Gaussians.

Therefore the mixture of Bernoullis gives us the best results in terms of clustering on this data set.

Answer 2:

i. Obtain the least squares solution $\mathbf{w}_{ML}$ to the regression problem using the analytical solution.

Here, we can use the closed form solution for calculating the optimal set of weights and solve Linear Regression.

We obtain the equation by taking Gradient of the objective function and equating it to 0. If we do so, we get the following formula:

The equation for the value of the actual Wml is:

ii. Code the gradient descent algorithm with suitable step size to solve the least squares algorithms and plot $\|\mathbf{w}^t - \mathbf{w}_{ML}\|_2$ as a function of t. What do you observe?

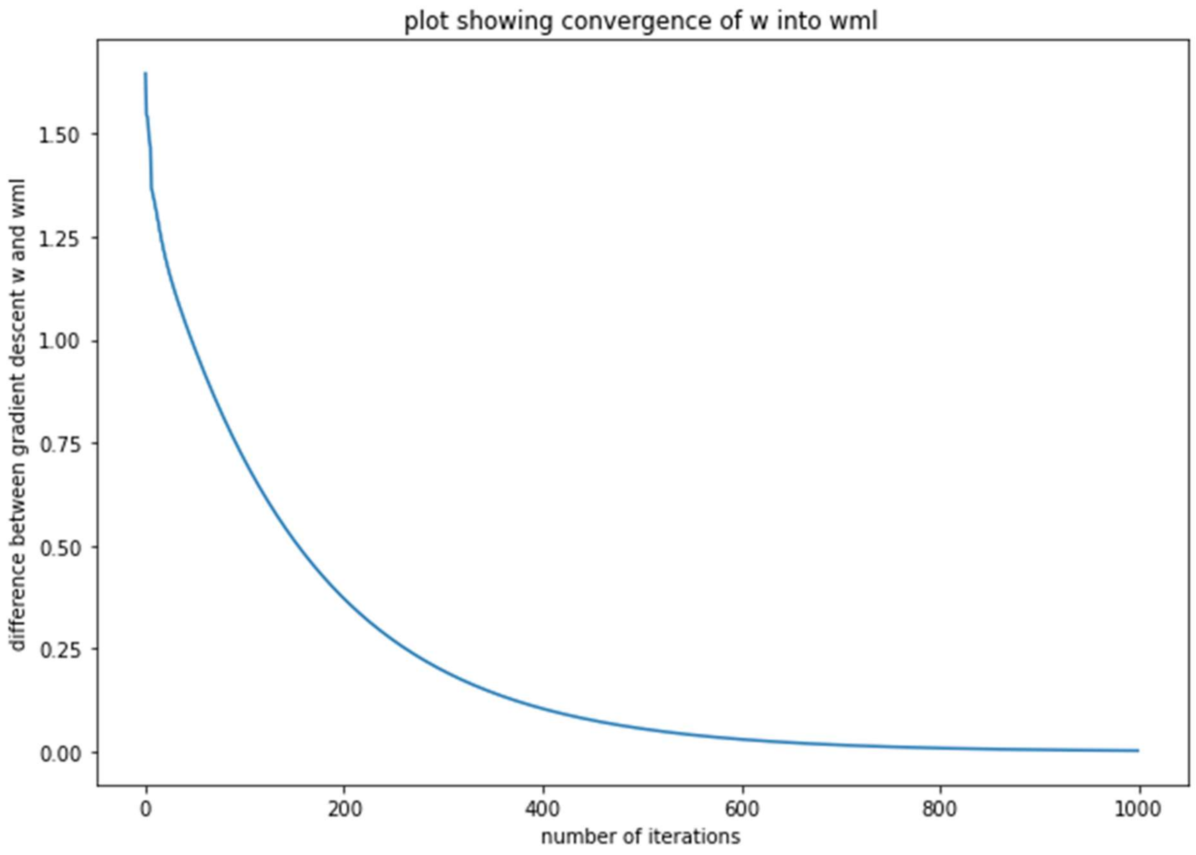
The gradient descent is an optimization method used to obtain minima in cases where equating the gradiant of a function to 0 may not have a closed form solution.

The steps for the algorithm are as follows:

1. Initialize a random w0
2. For every value of t in [1,T]
3. Take a gradient step against the gradient of the equation
4. End

Here, in addition to taking just a step size in the direction opposite to the gradient, we make the step size aware of the gradient's value, so that it doesn't overstep. We multiply the step size by 1/norm(gradient).

This way, the gradient descent does not have rubbish values.



iii. Code the stochastic gradient descent algorithm using batch size of 100 and plot $\|\mathbf{w}^t - \mathbf{w}_{ML}\|_2$ as a function of t. What are your observations?

The stochastic Gradient Descent is an optimization technique for reducing the computational requirements of the Gradient Descent algorithm.

It does so by randomly sampling small batches of data and calculating the gradients while assuming this small batch is the entire data set.

The Stochastic Gradient Descent we use is a variant of the standard version.

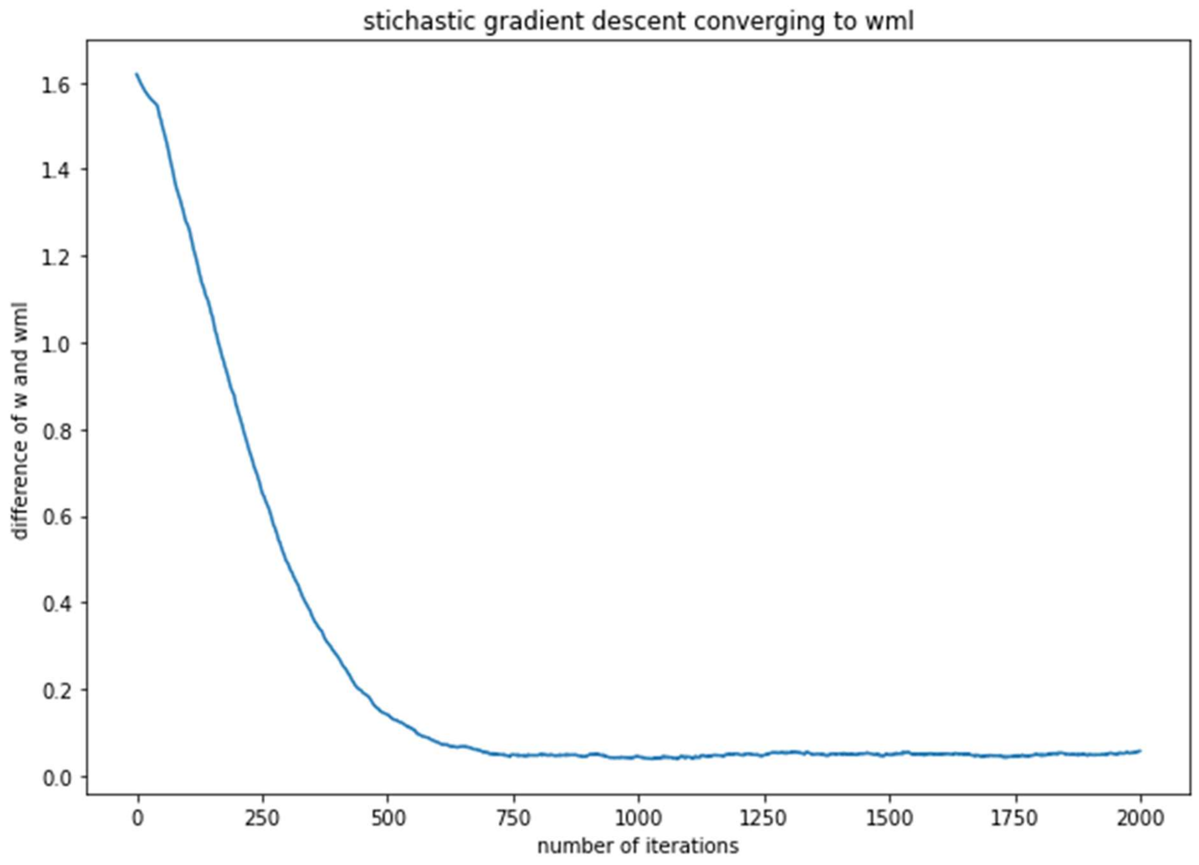
Here, the Algorithm is as follows:

1. For every value of t in [1, T]:
   a. At each step, sample a batch size of data points uniformly at random from the set of all points
   b. Assume this is the entire data set and take a gradient step with respect to it to it.

   End

2. After the T rounds, average formed in every iteration and return this value.

The algorithm is guaranteed to converge to an optima with high probability.



iv. Code the gradient descent algorithm for ridge regression. Cross-validate for various choices of λ and plot the error in the validation set as a function of λ. For the best λ chosen, obtain $\mathbf{w}_R$. Compare the test error (for the test data in the file A2Q2Data_test.csv) of $\mathbf{w}_R$ with $\mathbf{w}_{ML}$. Which is better and why?

In ridge regression, we promote a concept called regularisation. Regularisation checks if loss can be reduced further (calculated by the loss function) by increasing the length of the weights vector w, slightly.

Regularisation puts more weights on some features than other.

The new objective function for this purpose is as follows:

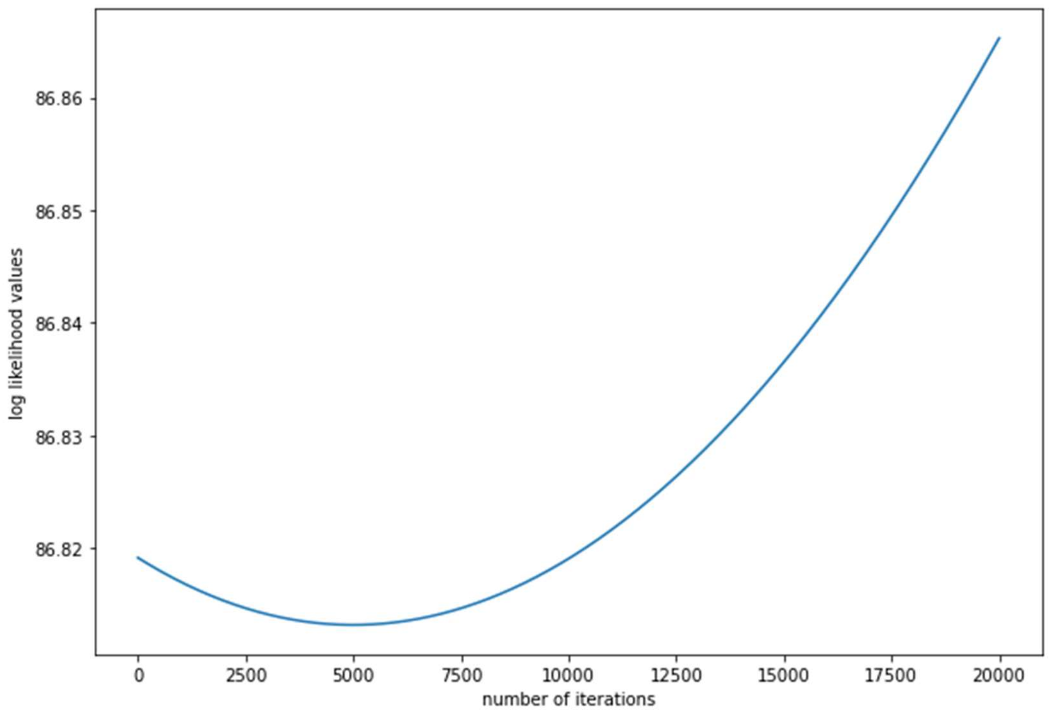
$$\begin{aligned}\hat{\beta}^{\text{ridge}} &= \underset{\beta\in\mathbb{R}^p}{\operatorname{argmin}} \sum_{i=1}^{n}(y_i - x_i^T\beta)^2 + \lambda\sum_{j=1}^{p}\beta_j^2 \\ &= \underset{\beta\in\mathbb{R}^p}{\operatorname{argmin}} \underbrace{\|y - X\beta\|_2^2}_{\text{Loss}} + \lambda \underbrace{\|\beta\|_2^2}_{\text{Penalty}}\end{aligned}$$

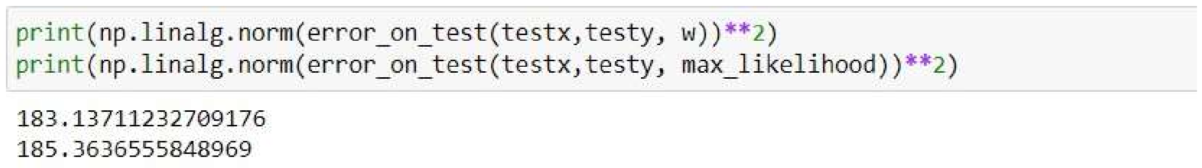
The above can be viewed as a constraint optimization problem where the same loss function is constrained to a norm 2 ball.

The amount we want to increase w by is denoted by lambda. This value is obtained by taking it as a hyper parameter and cross validating. Here, we shuffle the data set to introduce randomness, although an optimal way to do this would be through K fold cross validation.

The steps for the algorithm are as follows:

1. Take the gradient of the given ridge regression objective.
2. Perform gradient descent and obtain a w
3. Use this value of w to calculate the error.
4. Repeat steps 1 to 3 for over values of lambda in a decided range with decided accuracy.
5. The minimum value of lambda and the corresponding value of w should give an error with the original loss function either lesser than or equal to the above.





```
print(np.linalg.norm(error_on_test(testx,testy, w))**2)
print(np.linalg.norm(error_on_test(testx,testy, max_likelihood))**2)
```

```
183.13711232709176
185.3636555848969
```

The above shows the lower error values when using ridge as compared to when using normal regression.