CS5691: Pattern Recognition and Machine Learning

Assignment 2

Report

Course Instructor: Arun Rajkumar

Submitted By: Vennapareddy Abhigna

Roll Number: ME19B059

Date: 06/04/2022

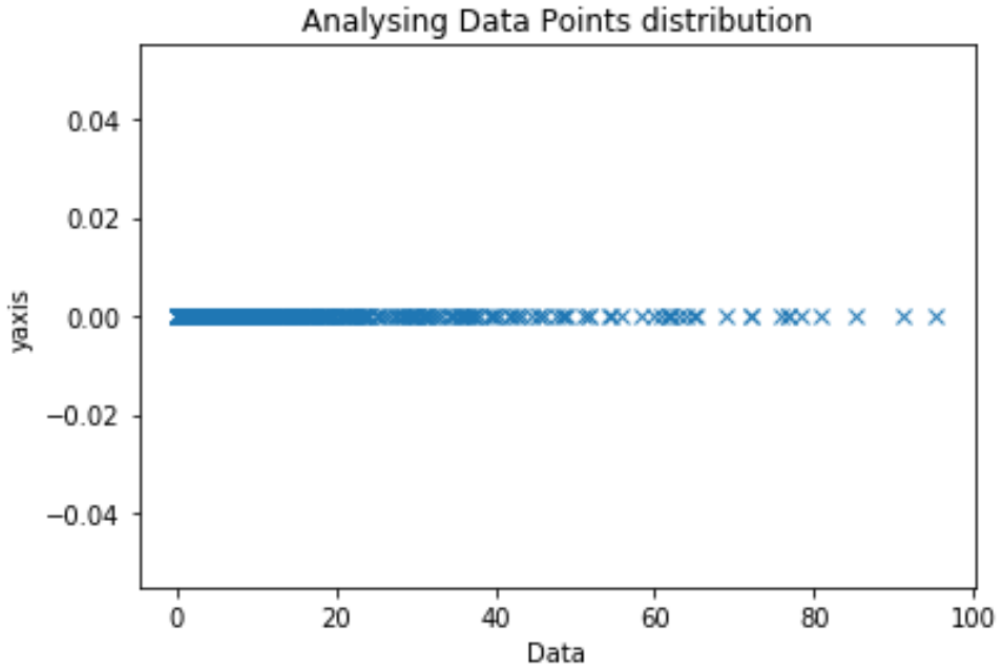
Indian Institute of Technology Madras

Chennai 600036, India

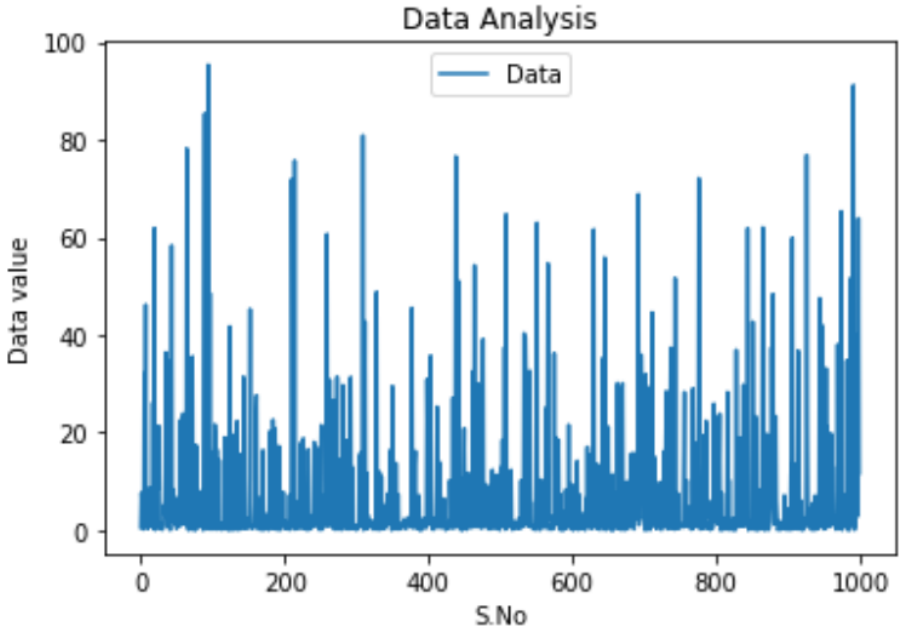
Questions and Solutions

1. Given a dataset with 1000 data points generated from a mixture of some distribution.

Data file 'A2Q1.csv' is observed and plotted on the y=0 line to observe the distribution of points on the line as shown below. Values given in the dataset range from 0 to ~100.



Data value of each row is observed in the graph below.



Before proceeding to the main problem, a few functions and common processes are first discussed below to make you familiar with the process later.

For the EM algorithm (we will discuss this below), we need initial parameters (called as theta which might contain means, variance, probability or pi based on the distribution mixture chosen) in order to run the algorithm. These initial parameters can be either obtained by performing k-means or by randomly choosing values. By observing both the processes (attached both the code files with the assignment), similar output is obtained. So, we will discuss k-means initialisation process here.

Process to get initial parameters through kmeans goes as follows (k-means clustering process - detailed explanation will be given in question 1.iii):

- We first assume a data point from the given data set as one cluster mean, and we obtain the remaining k-1 cluster means by calculating the distance of remaining points from the means and choosing the far placed data point as the mean. This way we find the k cluster means given an initial point. Function *'assumemean'* is coded for this process in the code file which returns k cluster means by taking one data point (cluster mean) as the input.

- Then we cluster the whole data based on these obtained cluster means by calculating the distance of each data point from the means and clustering it to the nearest mean cluster. Function *'cluster'* is coded for this clustering process in the code file which returns the cluster value of each data point by taking data and means as the input. It also returns an error which is the sum of the distance of all data points from their corresponding cluster means.

- After forming clusters, we now find the new cluster mean using the function '*newmean'* which calculates the distance of each point from its cluster old mean and averages it to get a new mean value. And then we give new clusters again with '*cluster*' function, and this process continues until cluster distribution converges to the same output in consecutive iterations. Function '*final*' is coded for this process and it outputs total number of iterations, final clusters, final cluster means and error value in each iteration as the output.

This is the process of k-means clustering and the final parameters obtained here are considered as initial parameters for the EM algorithm.

In the questions, we were asked to average over 100 random initializations of the parameters and then observe the log-likelihood. So we first find the averaged parameters by 100 random initial *'assumemean'* values. Hence we get average means, average variances and average pi value arrays for the initial step of the EM algorithm. We sort the data in ascending order and also corresponding all parameter values so that we won't face any problem in averaging as cluster numbers may not be the same all the time for the same cluster position. We store these clusters in the '*cluster_initial'* column in the data frame to call them later wherever required.

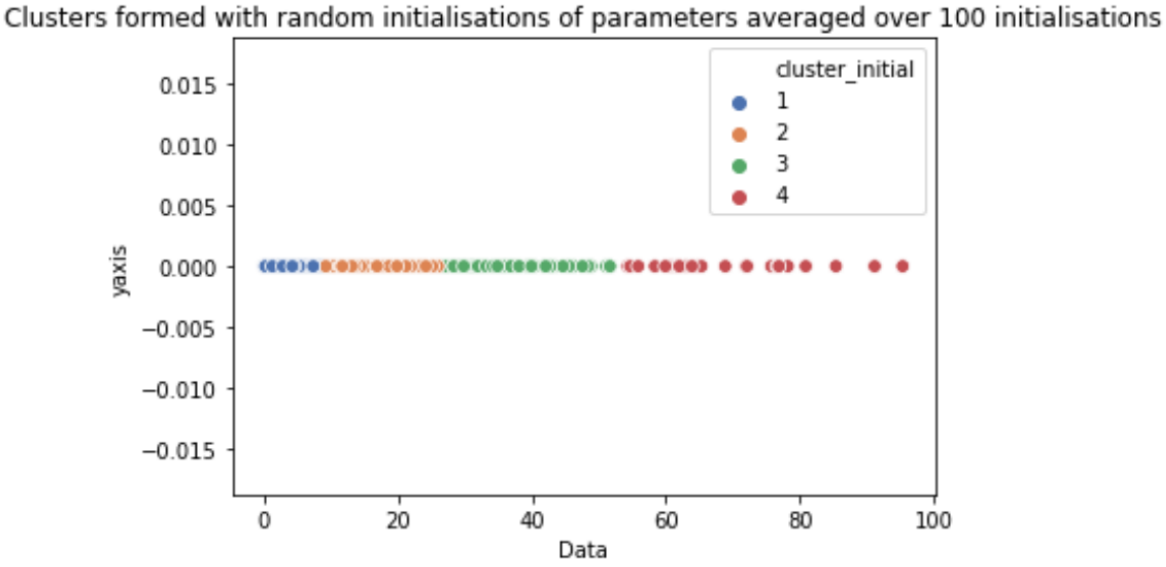
We were given to assume 4 number of mixtures i.e., K=4. Therefore, by performing the above 100 initializations averaged process, we obtain the following for 4 clusters:

- ➔ average means = [1.72009404, 16.20715899, 36.41782813, 68.84516]
- ➔ average variance = [4.28596673, 20.70951944, 46.87140761, 120.85994381]
- ➔ average probability (π) = [0.772, 0.139, 0.064, 0.025]

Here pi is the probability of a point to belong to a particular cluster which is calculated as:

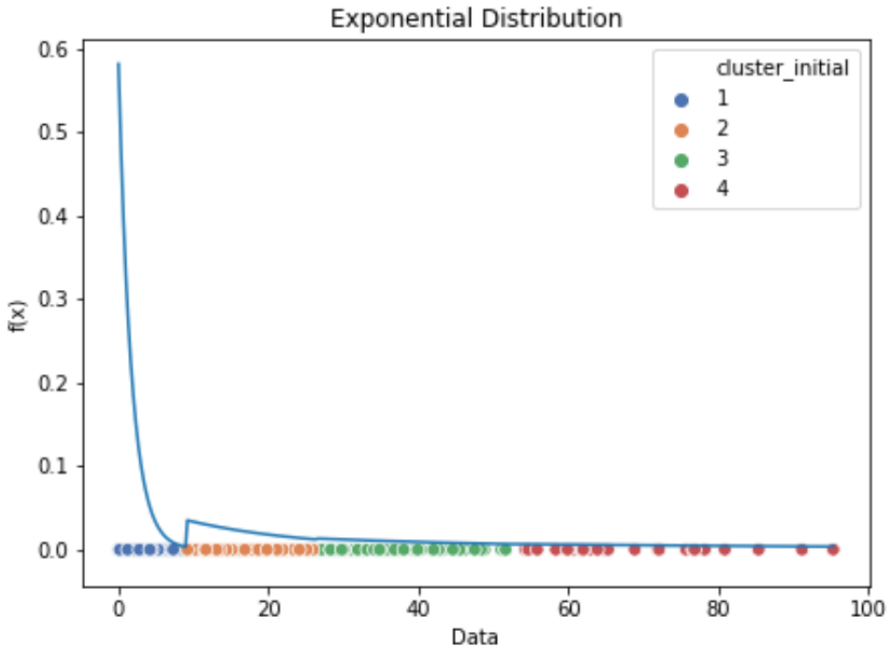
$$P(x_i = z) = \frac{No.\ of\ data\ points\ in\ cluster\ z}{Total\ no.\ of\ points} = \pi_z$$

On our given data set, the number of data points are divided as 772, 139, 64 and 25 in 4 clusters. The clusters formed are as shown below.



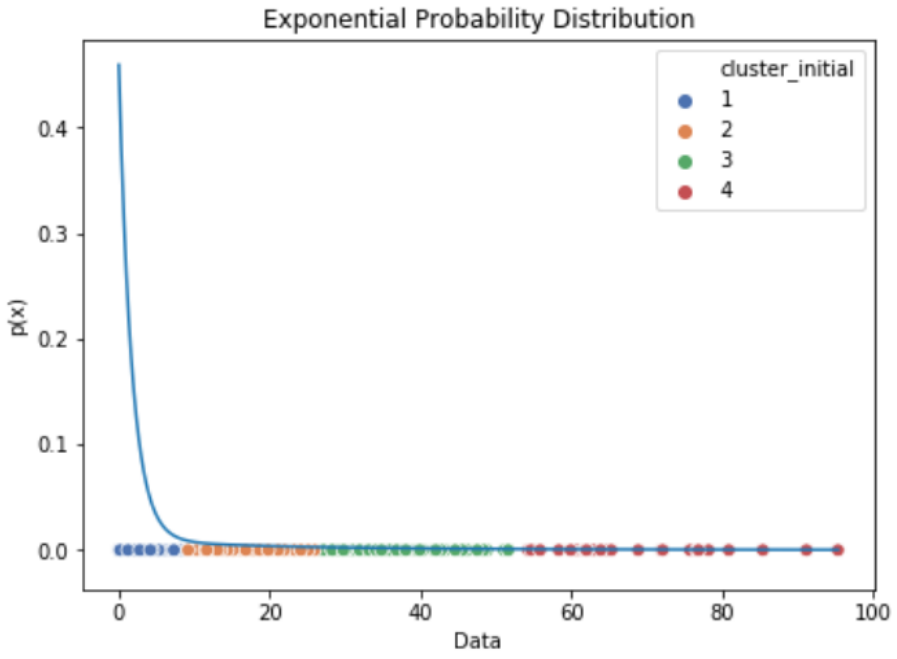
I. Exponential Distribution with K=4 mixtures:

Data could have been generated from the exponential distribution mixture, $f(x) = \lambda e^{-\lambda x}$. For this distribution, mean is obtained as $\int_{-\infty}^{+\infty} x f(x)\, dx = \frac{1}{\lambda}$. Therefore, with the initial averaged means, we get λ of each cluster using which the function has been plotted as below. $f(x) = \frac{e^{-x/\mu_k}}{\mu_k}$



Exponential probability distribution p(x) for the initial parameters is as shown below.

$$p(x_i) = \sum_{l=1}^{K} \pi_l \frac{e^{-x_i/\mu_l}}{\mu_l}$$



Maximum Likelihood for exponential distribution is given as:

$$L = \prod_{i=1}^{n} f(x_i; \mu_1,...., \mu_k; \pi_1,...., \pi_k)$$

$$L(\theta) = \prod_{i=1}^{n} [\sum_{k=1}^{K} \pi_k f(x_i; \mu_k)] \text{ , where } \theta \text{ is known for parameters } \mu \textit{ and } \pi$$

$$L(\theta) = \prod_{i=1}^{n} [\sum_{k=1}^{K} \pi_k \frac{e^{-x_i/\mu_k}}{\mu_k}]$$

Log Likelihood function is then written as:

$$\log L(\theta) = \sum_{i=1}^{n} \log (\sum_{k=1}^{K} \pi_k \frac{e^{-x_i/\mu_k}}{\mu_k})$$

It is not possible to solve this equation analytically by differentiating and equating it to zero. So, we use *Jensen's Inequality* to solve the log likelihood function.

By Jensen's Inequality, $f(\sum_{k=1}^{K} \lambda_k a_k) \geq \sum_{k=1}^{K} \lambda_k f(a_k)$

To implement this on our log likelihood, we first modify our log likelihood by introducing parameter λ.

For every data point i, take the parameters $\{\lambda_1^i, \lambda_2^i,...., \lambda_k^i\}$ such that $\forall$ i $\sum_{k=1}^{K} \lambda_k^i = 1, 0 \leq \lambda_k^i \leq 1 \ \forall$ i, k.

$$\text{modified\_log } L(\theta) = \sum_{i=1}^{n} \log \Big(\sum_{k=1}^{K} \lambda_k^{\,i} \Big(\frac{\pi_k \frac{e^{-x_i/\mu_k}}{\mu_k}}{\lambda_k^{\,i}} \Big) \Big).$$

$$\text{modified\_log } L(\theta) = \sum_{i=1}^{n} \sum_{k=1}^{K} \lambda_k^{\,i} \log \Big(\frac{\pi_k \frac{e^{-x_i/\mu_k}}{\mu_k}}{\lambda_k^{\,i}} \Big) \quad \text{['By jensen's inequality']}$$

Again by Jensen's Inequality, log L(θ) ≥ modified_log L(θ, λ) which means modified log likelihood gives the lower bound for true log likelihood. So, if we fix λ, it is easy to maximize with respect to θ and if we fix θ, it is easy to maximize with respect to λ.

➔ Fixing λ and maximizing θ:

$$max_{\Theta} \sum_{i=1}^{n} \sum_{k=1}^{K} \lambda_k^{\,i} \log \Big(\frac{\pi_k \frac{e^{-x_i/\mu_k}}{\mu_k}}{\lambda_k^{\,i}} \Big)$$

$$max_{\Theta} \sum_{i=1}^{n} \sum_{k=1}^{K} [\lambda_k^{\,i} log\, \pi_k - \lambda_k^{\,i} \frac{x_i}{\mu_k} - \lambda_k^{\,i} log\, \mu_k - \lambda_k^{\,i} log\, \lambda_k^{\,i}]$$

Taking derivative with respect to μ, we get $\hat{\mu}_k^{MML} = \frac{\sum_{i=1}^{n} \lambda_k^{\,i} x_i}{\sum_{i=1}^{n} \lambda_k^{\,i}}$.

$$max_{\pi_1,\ldots,\pi_k} \sum_{i=1}^{n} \sum_{k=1}^{K} \lambda_k^{\,i} log\, \pi_k \text{ s.t} \sum_k \pi_k = 1;\ \pi_k \geq 0$$

Using Lagrange multipliers, we get $\hat{\pi}_k^{MML} = \frac{\sum_{i=1}^{n} \lambda_k^{\,i}}{n}$

➔ Fixing θ and maximizing λ:

$$max_{\lambda} \sum_{i=1}^{n} \sum_{k=1}^{K} \lambda_k^{\,i} \log \Big(\frac{\pi_k \frac{e^{-x_i/\mu_k}}{\mu_k}}{\lambda_k^{\,i}} \Big)$$

$$max_{\lambda} \sum_{i=1}^{n} \sum_{k=1}^{K} [\lambda_k^{\,i} \log \Big(\pi_k \frac{e^{-x_i/\mu_k}}{\mu_k} \Big) - \lambda_k^{\,i} \log \lambda_k^{\,i}]$$

We can maximize λ for every i, therefore we fix i to get $\lambda_k^{\,i}$

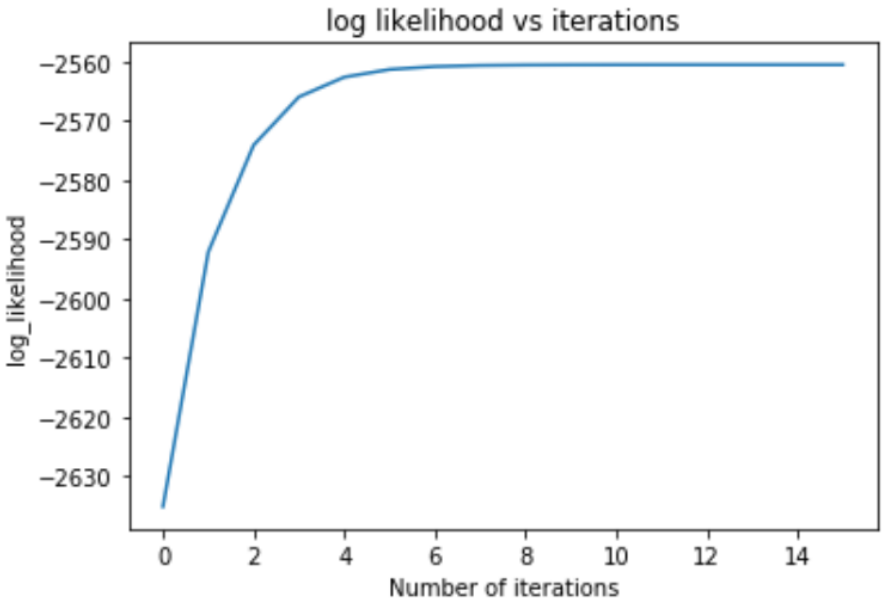
$$max_{\lambda_1^i,\ldots,\lambda_k^i} \sum_{k=1}^{K} [\lambda_k^i \log (\pi_k \frac{e^{-x_i/\mu_k}}{\mu_k}) - \lambda_k^i \log \lambda_k^i] \text{ s.t } \sum_{k=1}^{K} \lambda_k^i = 1,\ 0 \leq \lambda_k^i \leq 1$$

Using Lagrange multipliers, we get $\hat{\lambda}_k^i = \frac{\pi_k \frac{e^{-x_i/\mu_k}}{\mu_k}}{\sum_{l=1}^{k} (\pi_l \frac{e^{-x_i/\mu_l}}{\mu_l})} = P(z_i = k/x_i) = \frac{P(z_i=k)P(x_i/z_i=k)}{P(x_i)}$

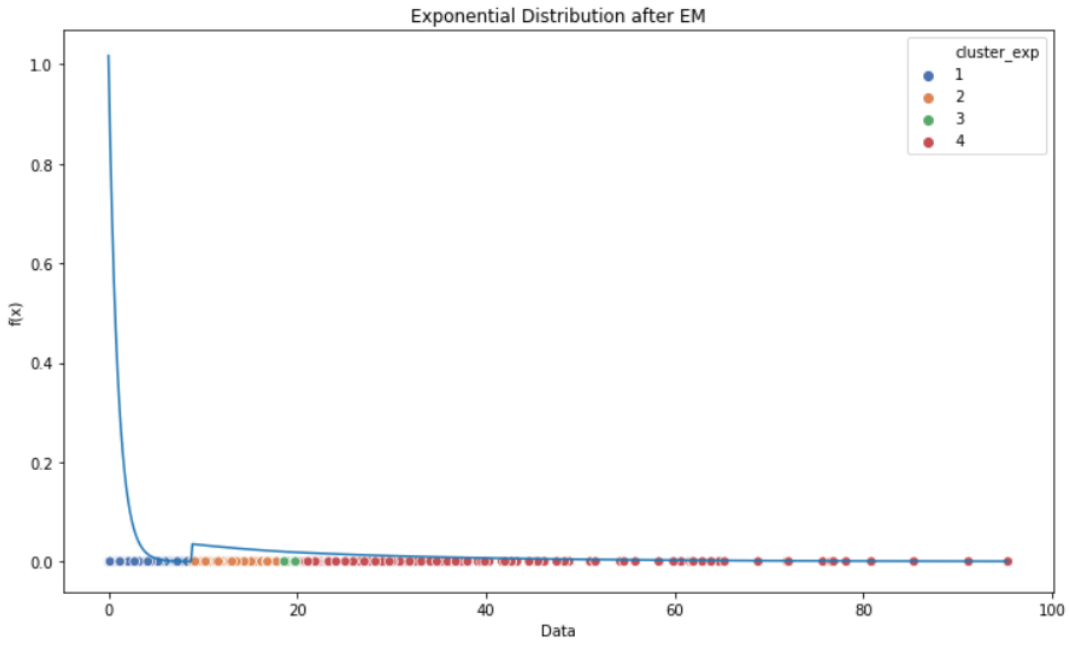
Process of EM algorithm:

- ➔ Initialize $\theta^0 = \{\mu_1^0, \ldots, \mu_k^0; \pi_1^0, \ldots \pi_k^0\}$. We have obtained this at the beginning by averaging 100 random initializations.
- ➔ Until convergence of $||\theta^{t+1} - \theta^t|| \leq \epsilon$ for some tolerance, we update λ and θ.
  - ◆ $\lambda^{t+1} = arg\ max_\lambda\ modified\ log\ L(\theta^t, \lambda)$ → Expectation step
  - ◆ $\theta^{t+1} = arg\ max_\theta\ modified\ log\ L(\theta, \lambda^{t+1})$ → Maximization step
- ➔ end

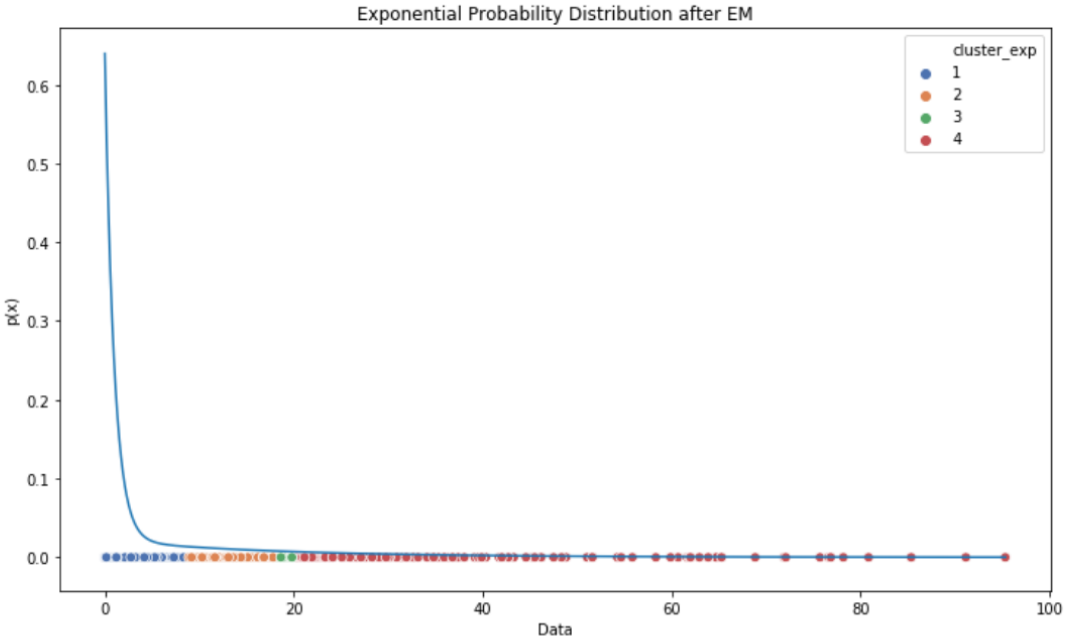
Performing EM algorithm on our data, the graph of log likelihood as a function of number of iterations is as shown below which illustrates that log likelihood increases with increase in number of iterations as parameters move towards convergence.



With the final θ^T obtained after convergence, we get μ_k through which we can form and analyze final clusters and the exponential function as shown below.



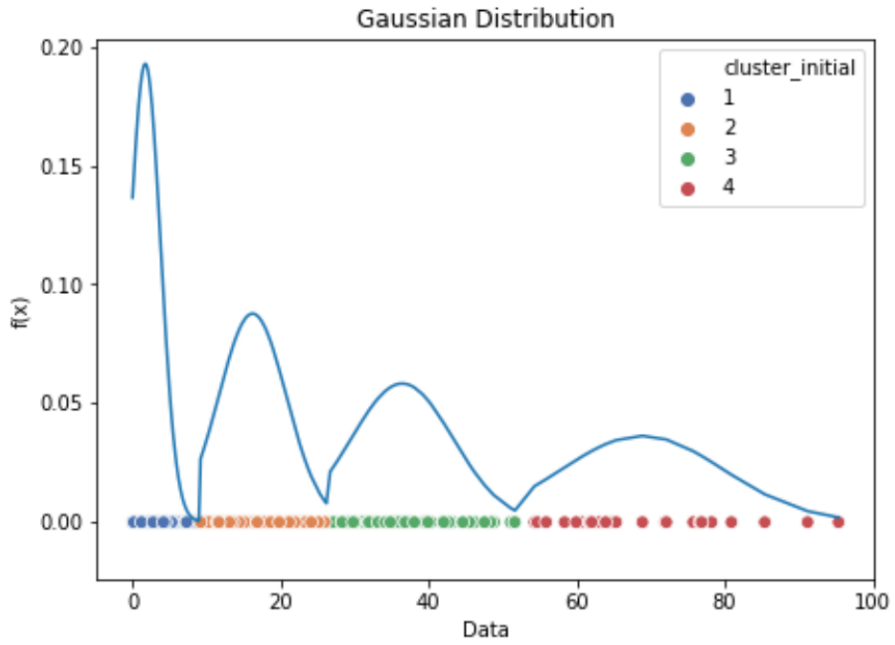
With final converged parameters, Exponential probability distribution p(x) is as shown below.



II. Gaussian Distribution with K=4 mixtures:

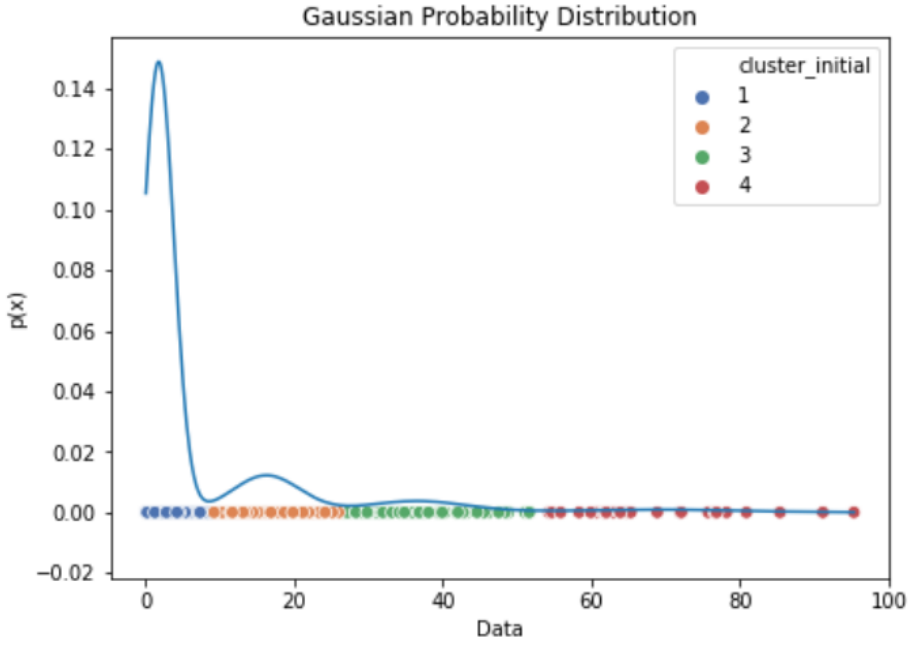
$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

Initial parameters are obtained above by averaging 100 random initializations, using those initial μ *and* σ, the clusters and the corresponding function is as shown below.



Gaussian probability distribution p(x) for the initial parameters is as shown below.

$$p(x_i) = \sum_{l=1}^{k} \left(\pi_l \frac{1}{\sqrt{2\pi\sigma_l^2}} e^{\frac{-(x_i-\mu_l)^2}{2\sigma_l^2}}\right)$$



Maximum Likelihood for exponential distribution is given as:

$$L = \prod_{i=1}^{n} f(x_i; \mu_1, \ldots, \mu_k; \sigma_1^2, \ldots, \sigma_k^2, \pi_1, \ldots, \pi_k)$$

$$L(\theta) = \prod_{i=1}^{n} [\sum_{k=1}^{K} \pi_k f(x_i; \mu_k, \sigma_k^2]$$, where θ is known for parameters μ, σ *and* π

$$L(\theta) = \prod_{i=1}^{n} [\sum_{k=1}^{K} \pi_k \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{\frac{-(x_i-\mu_k)^2}{2\sigma_k^2}}]$$

Log Likelihood function is then written as:

$$\log L(\theta) = \sum_{i=1}^{n} \log (\sum_{k=1}^{K} \pi_k \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{\frac{-(x_i-\mu_k)^2}{2\sigma_k^2}})$$

It is not possible to solve this equation analytically by differentiating and equating it to zero. So, we use *Jensen's Inequality* to solve the log likelihood function.

By Jensen's Inequality, $f(\sum_{k=1}^{K} \lambda_k a_k) \geq \sum_{k=1}^{K} \lambda_k f(a_k)$

To implement this on our log likelihood, we first modify our log likelihood by introducing parameter λ.

For every data point i, take the parameters $\{\lambda_1^i, \lambda_2^i, ..., \lambda_k^i\}$ such that $\forall$ i $\sum_{k=1}^{K} \lambda_k^i = 1, 0 \leq \lambda_k^i \leq 1 \ \forall$ i, k.

$$\text{modified\_log } L(\theta) = \sum_{i=1}^{n} \log (\sum_{k=1}^{K} \lambda_k^i (\frac{\pi_k \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{\frac{-(x_i-\mu_k)^2}{2\sigma_k^2}}}{\lambda_k^i})).$$

$$\text{modified\_log } L(\theta) = \sum_{i=1}^{n} \sum_{k=1}^{K} \lambda_k^i \log (\frac{\pi_k \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{\frac{-(x_i-\mu_k)^2}{2\sigma_k^2}}}{\lambda_k^i}) \text{ ['By jensen's inequality']}$$

Again by Jensen's Inequality, log L(θ) ≥ modified_log L(θ, λ) which means modified log likelihood gives the lower bound for true log likelihood. So, if we fix λ, it is easy to maximize with respect to θ and if we fix θ, it is easy to maximize with respect to λ.

➔ Fixing λ and maximizing θ:

$$max_\Theta \sum_{i=1}^{n} \sum_{k=1}^{K} \lambda_k^i \log (\frac{\pi_k \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{\frac{-(x_i-\mu_k)^2}{2\sigma_k^2}}}{\lambda_k^i})$$

$$max_\Theta \sum_{i=1}^{n} \sum_{k=1}^{K} [\lambda_k^i \log \pi_k - \lambda_k^i \frac{(x_i-\mu_k)^2}{2\sigma_k^2} - \lambda_k^i \log \sqrt{2\pi\sigma_k^2} - \lambda_k^i \log \lambda_k^i]$$

Taking derivative with respect to μ, we get $\hat{\mu}_k^{MML} = \frac{\sum_{i=1}^{n} \lambda_k^i x_i}{\sum_{i=1}^{n} \lambda_k^i}$.

Taking derivative with respect to σ , we get $\hat{\sigma}_k^{2\,MML} = \frac{\sum_{i=1}^{n} \lambda_k^i (x_i - \hat{\mu}_k^{MML})^2}{\sum_{i=1}^{n} \lambda_k^i}$

$$max_{\pi_1,\ldots,\pi_k} \sum_{i=1}^{n}\sum_{k=1}^{K} \lambda_k^i \log \pi_k \text{ s.t} \sum_k \pi_k = 1;\ \pi_k \geq 0$$

Using Lagrange multipliers, we get $\hat{\pi}_k^{MML} = \frac{\sum_{i=1}^{n} \lambda_k^i}{n}$

➔ Fixing θ and maximizing λ:

$$max_{\lambda} \sum_{i=1}^{n}\sum_{k=1}^{K} \lambda_k^i \log \left(\frac{\pi_k \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{\frac{-(x_i-\mu_k)^2}{2\sigma_k^2}}}{\lambda_k^i}\right)$$

$$max_{\lambda} \sum_{i=1}^{n}\sum_{k=1}^{K} \left[\lambda_k^i \log \left(\pi_k \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{\frac{-(x_i-\mu_k)^2}{2\sigma_k^2}}\right) - \lambda_k^i \log \lambda_k^i\right]$$

We can maximize λ for every i, therefore we fix i to get λ_k^i

$$max_{\lambda_1^i,\ldots,\lambda_k^i} \sum_{k=1}^{K} \left[\lambda_k^i \log \left(\pi_k \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{\frac{-(x_i-\mu_k)^2}{2\sigma_k^2}}\right) - \lambda_k^i \log \lambda_k^i\right] \text{ s.t } \sum_{k=1}^{K} \lambda_k^i = 1,\ 0 \leq \lambda_k^i \leq 1$$

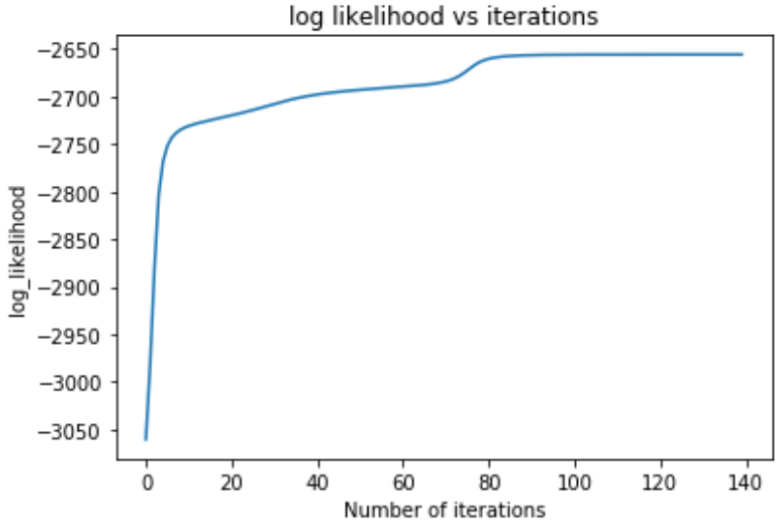
Using Lagrange multipliers, we get

$$\hat{\lambda}_k^i = \frac{\pi_k \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{\frac{-(x_i-\mu_k)^2}{2\sigma_k^2}}}{\sum_{l=1}^{k} \left(\pi_l \frac{1}{\sqrt{2\pi\sigma_l^2}} e^{\frac{-(x_i-\mu_l)^2}{2\sigma_l^2}}\right)} = P(z_i = k/x_i) = \frac{P(z_i=k)P(x_i/z_i=k)}{P(x_i)}$$

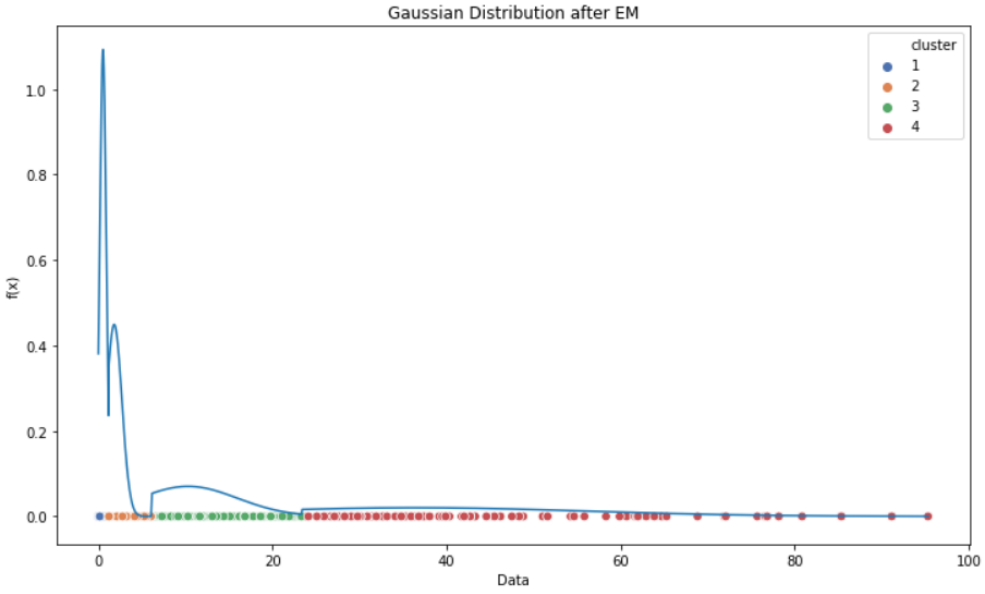
Process of EM algorithm:

- ➔ Initialize $\theta^0 = \{\mu_1^0, \ldots, \mu_k^0; \sigma_1^{2^0}, \ldots, \sigma_k^{2^0}, \pi_1^0, \ldots \pi_k^0\}$. We have obtained this at the beginning by averaging 100 random initializations.
- ➔ Until convergence of $||\theta^{t+1} - \theta^t|| \leq \epsilon$ for some tolerance, we update λ and θ.
  - ◆ $\lambda^{t+1} = arg\,max_\lambda\, modified\, log\, L(\theta^t, \lambda)$ → Expectation step
  - ◆ $\theta^{t+1} = arg\,max_\theta\, modified\, log\, L(\theta, \lambda^{t+1})$ → Maximization step
- ➔ end

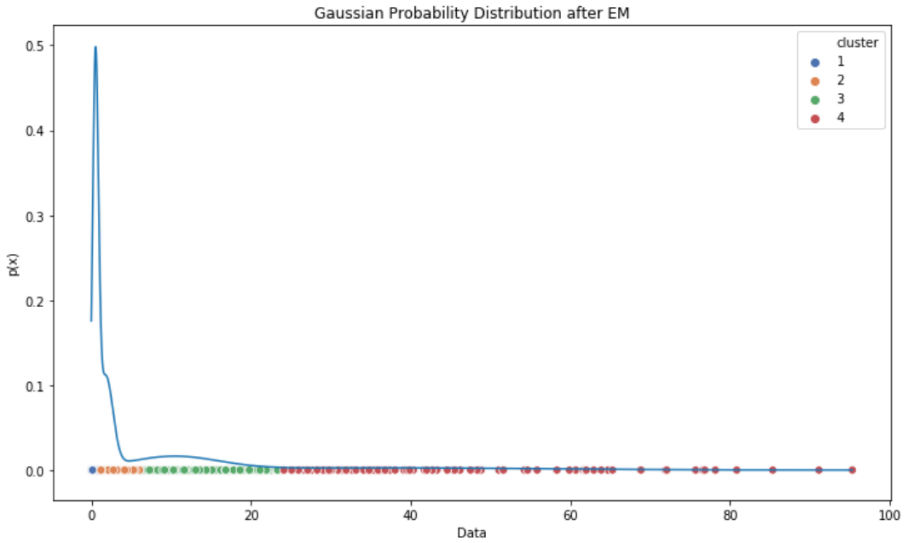
Performing EM algorithm on our data, the graph of log likelihood as a function of number of iterations is as shown below which illustrates that log likelihood increases with increase in number of iterations as parameters move towards convergence.



With the final θ^T obtained after convergence, we get μ_k *and* σ_k^2 through which we can form and analyze final clusters and the gaussian function as shown below.



With final converged parameters, Gaussian probability distribution p(x) is as shown below.



By comparing log likelihood as a function of iterations for exponential and gaussian distribution, we can derive the following observations:

- ➔ Both the plots try to maximize the log likelihood.
- ➔ Exponential distribution maximized to ~-2560 where as gaussian distribution maximized to ~-2650. Final Exponential log likelihood > Final Gaussian log likelihood.
- ➔ Exponential distribution log likelihood converged for less number of iterations whereas Gaussian distribution log likelihood converged for far more number of iterations and that too to a value less than that obtained in exponential distribution.
- ➔ Exponential distribution best explains the data distribution compared to gaussian distribution.

III. K-means algorithm with K=4:

We have already explained the functions used in the k-means algorithm in the beginning of this question. Process goes as follows:

- ➔ Assumed the first mean $\{\mu_1^0\}$ to be the 200th data point and used the '*assumemean*' function to get other 3 means which are obtained as [[1.2941], [95.346], [48.363], [71.925]].

  '*assumemean*' function chooses μ_l^0 probabilistically according to score,

  $S(x) = min_{j=1,..l-1} ||x - \mu_j^0||_2^2$. Each successive mean is chosen which maximizes this S(x).

- ➔ Now we use these means to give initial clusters (z) to the data points.

$$z_i = arg\ min_k ||x_i - \mu_k||_2^2 \quad \forall \text{ i, where z } \epsilon \{1,...,k\}$$

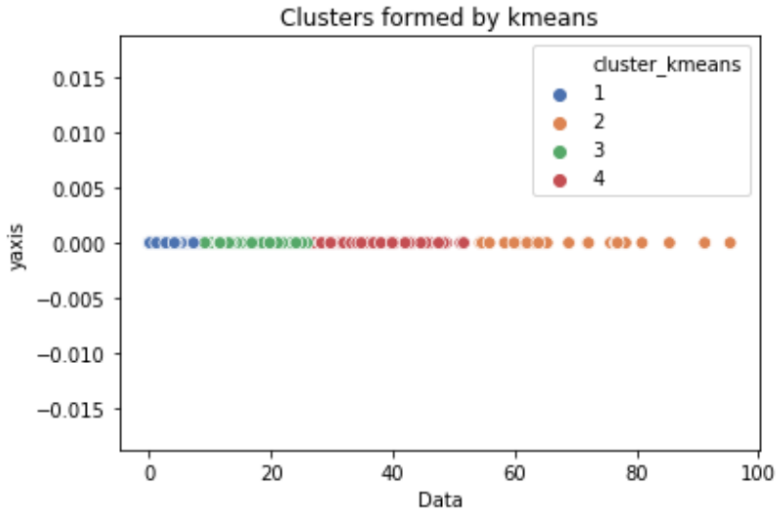
- ➔ Until convergence (i.e., till $z_i^{t+1} = z_i$):
  - ◆ Mean of cluster k at iteration t:

$$\mu_k^{\,t} = \frac{\sum_{i=1}^{n} x_i \, \Pi(z_i^{\,t} = k)}{\sum_{i=1}^{n} \Pi(z_i^{\,t} = k)} \quad \text{where } \Pi(z_i = k) = 1 \; only \; when \; z_i = k, \; and \; 0 \; otherwise$$

- Re-assignment step:

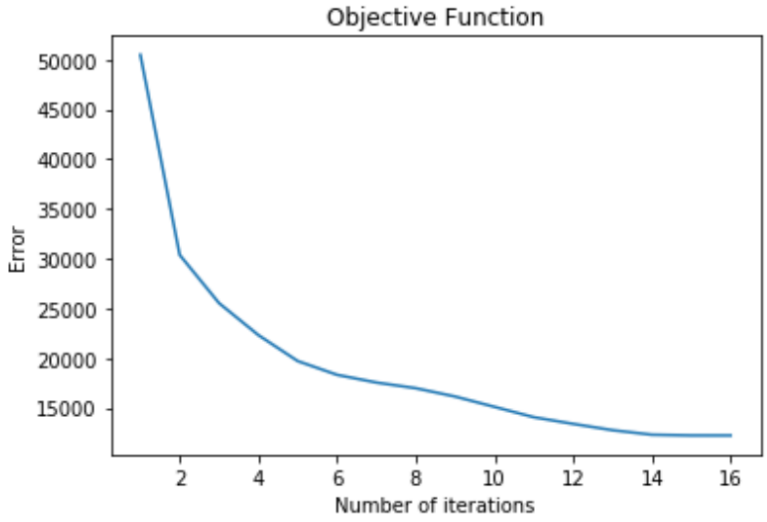
$$z_i^{\,t+1} = arg\,min_k \, ||x_i - \mu_k^{\,t}||_2^{\,2} \quad \forall \text{ i, where z } \epsilon \{1,\ldots,k\}$$

Function '*final*' is used for this process and the final means obtained for our data are {[1.72009404, 68.84516 , 16.20715899, 36.41782812]}. With the final cluster 'z', data forms the clusters as shown below.



➔ end

We get objective function value also from the function '*final*' which is $\sum_{i=1}^{n} ||x_i - \mu_k||_2^{\,2}$ where $k = z_i$. Objective of k-means as a function of iterations is as shown below.



As the iterations increase to converge, objective function decreases drastically to give the best possible clusters with obtained means.

IV. k-means doesn't remember the data, it just clusters the given data. And for the EM algorithm in both exponential and gaussian distributions, we initialized parameters using k-means itself. So k-means works better in terms of clustering the existing given data.

In the case of the EM algorithm, we try to estimate and play with probabilities of the points to go to a particular cluster which helps us in predicting the cluster of a new data point. We use k-means to analyze and remember the data through probability observations to predict the new output.

Hence, for the case where we have to deal with new data points, I would choose the EM algorithm and in case where we need to cluster given data points, I would choose k-means.

For this data distribution in the EM algorithm, I would choose exponential distribution as it increased the likelihood function more than gaussian in less number of iterations.

2. Given a dataset with 10000 data points in (R^{100}, R) - Each row corresponds to a datapoint where the first 100 components are features and the last component is the associated y value.

Dataset 'A2Q2Data_train.csv' is observed and the first 100 columns are assigned to features x1, x2,... x100, and the last column is assigned to y.

I. Least Square Solution w_{ML} using analytical approach:

In the regression problem, we have an input $(x_1, x_2, \ldots x_n) \in R^d$ with $(y_1, y_2, \ldots y_n) \in$ R and we learn to perform ,

($x_1, x_2, \ldots x_n$ here is different from the column feature names that we gave above)

$$H_{linear} = \{h_w : R^d \rightarrow \text{R such that } h_w(\text{x}) = w^T x \text{ , } w \in R^d\}.$$

Our goal is to minimize the squared error,

$$min_{h_w \in H_{linear}} \sum_{i=1}^{n} (h_w(x_i) - y_i)^2$$

$$min_{w \in R^d} \sum_{i=1}^{n} (w^T x_i - y_i)^2$$

This is also known as linear regression;

$$min_{w \in R^d} \sum_{i=1}^{n} (w^T x_i - y_i)^2 = ||X^T W - Y||_2^2 ,$$

where $X \in R^{dxn}$ and $Y \in R^{nx1}$ and $W \in R^{dx1}$

$$min_{w \in R^d} \sum_{i=1}^{n} (w^T x_i - y_i)^2 = min_{w \in R^d} (X^T W - Y)^T (X^T W - Y)$$

This is an unconstrained optimization problem, quadratic in W. Therefore we can directly find the W that minimizes the error by derivative and equating it to zero.

$$\text{Let f(W)} = (X^T W - Y)^T (X^T W - Y)$$

$$\text{f(W)} = W^T (XX^T) W - W^T XY - Y^T X^T W + Y^T Y$$

$$\text{delta f(W)} = 2(XX^T) W - 2(XY) = 0$$

$$(XX^T) w^* = XY$$

$$w^* = (XX^T)^\dagger XY = w_{ML}$$

Hence, we get our least square solution w_{ML} which is the pseudo inverse of XX^T times XY. The above solution is coded in PRML_A2Q2.ipynb for the given data and w_{ML} is obtained where $w_{ML} \in R^d$.

Least Square Error obtained by w_{ML} can be calculated as,

$$||X^T w_{ML} - Y||_2^2 = (X^T w_{ML} - Y)^T (X^T w_{ML} - Y)$$

For our data, least square error with the obtained w_{ML} is calculated to be 396.8644186272516.

II. Gradient Descent Algorithm:

In this algorithm, we solve least square algorithm in steps by updating w_t in each step i.e., w_t is updated to w_{t+1} such that w_{t+1} tries converge to the optimal w.

$$\text{We already have f(W)} = (X^T W - Y)^T (X^T W - Y)$$

$$\text{f(W)} = W^T (XX^T) W - W^T XY - Y^T X^T W + Y^T Y$$

$$\text{delta f(W)} = 2(XX^T) W - 2(XY)$$

Gradient Descent update step is given as,

$$w_{t+1} = w_t - \eta_t \, [\text{delta f}(w_t)]$$

$$w_{t+1} = w_t - \eta_t \, [2(XX^T) w_t - 2(XY)]$$

Here η_t is the step size and - delta f(w_t) gives the direction in which w_t should be updated in order to reach the optimal position. Hence $w_t + \eta_t$ [-delta f(w_t)] is obtained as the updated w i.e., w_{t+1}.

We perform gradient descent till convergence where squared error is minimized. The final w_t that we obtain here is greatly influenced by the step size η_t. Having a constant step size may lead to the problem of non convergence even after infinite iterations, therefore we choose step size in such a way that it decreases with the increase in iterations. So, we choose η_t such that $\sum_{t=1}^{\infty} \eta_t = \infty$ (diverging sum) and $\sum_{t=1}^{\infty} \eta_t^2 < \infty$ (converging sum).

Therefore, we choose η_t proportional 1/t.

The above discussed process can be concluded as:

- ➔ We initialize w_0, randomly choose a w.
- ➔ For t = 1,....,T, we perform gradient descent update step:

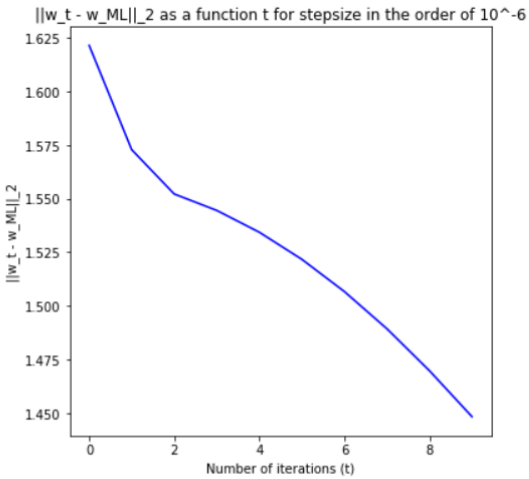
$$w_{t+1} = w_t - \eta_t \text{ [delta f}(w_t)]$$

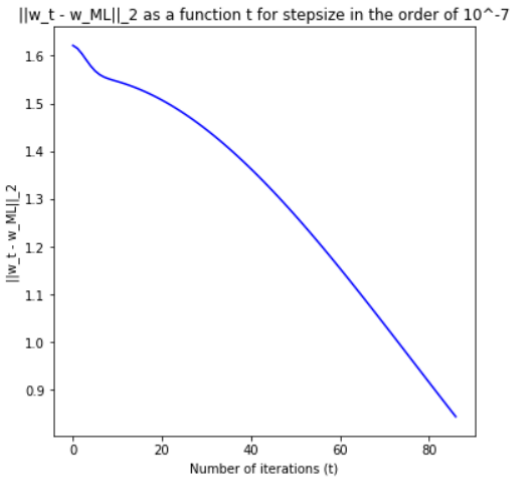
- ➔ End ($w_{gd} = w_T$)

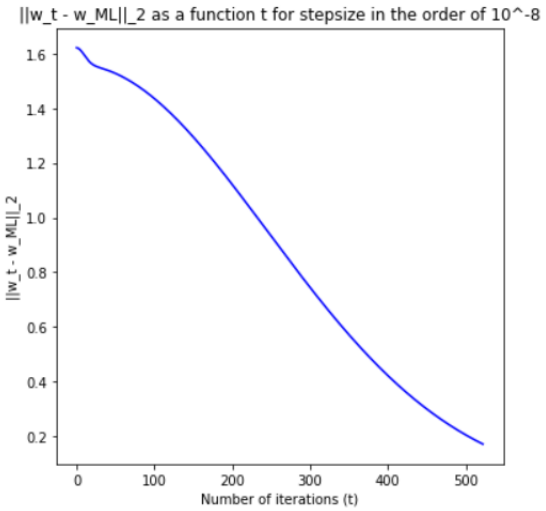
The above process results in minimization of $||X^T w_{gd} - Y||_2^2$.

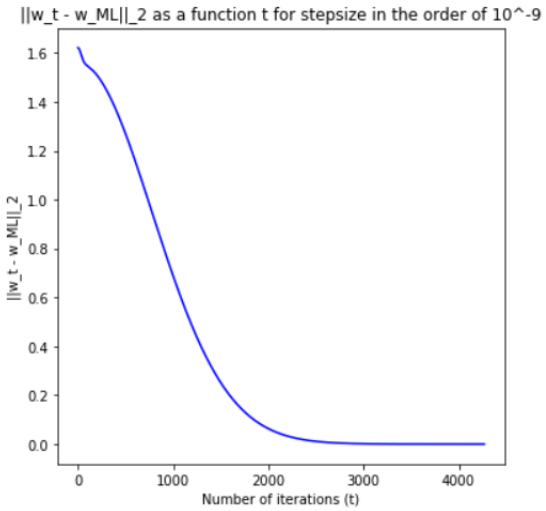
For the given data, the graph of '$||w_t - w_{ML}||_2$ vs t' depends on the step size η_t. By analyzing various step sizes and values, it has been observed that step size of the order 10^{-9} i.e., $\eta_t = \frac{10^{-9}}{t}$ gave the best converged w_T.

'$||w_t - w_{ML}||_2$ vs t' graphs are analyzed for $\eta_t = \frac{10^{-6}}{t}, \frac{10^{-7}}{t}, \frac{10^{-8}}{t}$ and $\frac{10^{-9}}{t}$ as shown below.









It can be observed that w_t tries to converge to w_{ML} with each iteration. It is also observed that as the step size decreased, number of iterations to closely converge increases with w_t moving more and more closely towards w_{ML}.

With different step size, we obtain different w_T which in turn influences the least square error. Least Square Error obtained by w_T can be calculated as,

$$||X^T w_T - Y||_2^2 = (X^T w_T - Y)^T (X^T w_T - Y)$$

For our data, least square error with the obtained w_T is calculated to be

$$2195.292263013331 \text{ for } \eta_t = \frac{10^{-6}}{t},$$

$$994.5344916146937 \text{ for } \eta_t = \frac{10^{-7}}{t},$$

$$420.8321967376389 \text{ for } \eta_t = \frac{10^{-8}}{t},$$

$$396.86441862958867 \text{ for } \eta_t = \frac{10^{-9}}{t}$$

Therefore, it is evident that w_T tries to converge to the optimal point as we can observe the decrease in least square errors when the iterations increased and the step size decreased to move more towards w_{ML}.

This algorithm is highly dependent on the step size that we choose (it may even diverge from the optimal value with some step sizes after certain iterations) and it takes so many iterations to converge to w_{ML} which was obtained easily in an analytical method. But by analytical method, it is not always easily possible to find the pseudo inverse of XX^T, it will get expensive when d is large. Hence we use a gradient descent algorithm, which would try to converge though it takes a large number of iterations.

III. Stochastic Gradient Descent:

In the above question, we saw how gradient descent is different from the analytical method. Gradient Descent process involves computation of XX^T which we need to avoid when n is too large. So, we adopt stochastic gradient descent to overcome this problem.

In the stochastic gradient descent algorithm, we perform gradient descent by choosing a different bunch of data in each step. The process goes as follows:

For t = 1,....T

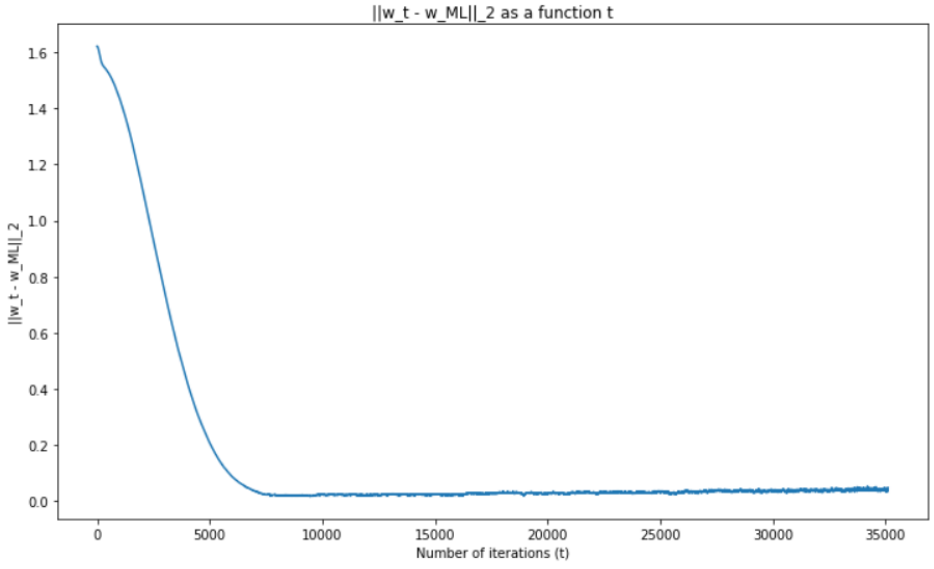
- ➔ At each step, we sample a bunch (in this question, we took it as 100 data observations i.e., k=100) of data points uniformly at random from the set of all points given in the data file.
- ➔ We pretend this sample as the entire data set and perform a gradient descent step $w_{t+1} = w_t - \eta_t [2(\overline{xx}^T)w_t - 2(\overline{xy})]$. This is possible to compute as $\overline{x} \in R^{dxk}$.
- ➔ end
- ➔ After T rounds, we find $w_{sgd,T} = \frac{1}{T}\sum_{t=1}^{T} w_t$

The above process results in minimization of $||X^T w_{sgd} - Y||_2^2$

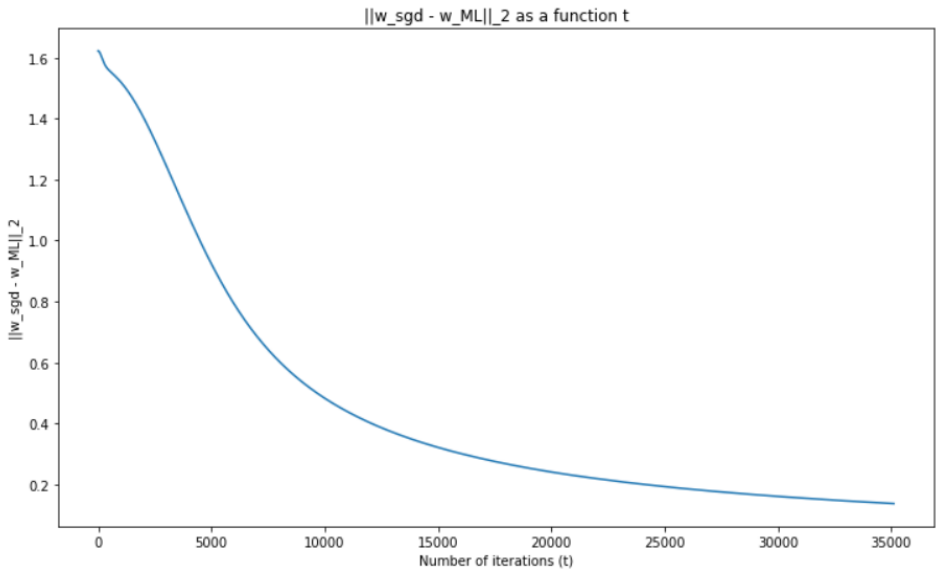
In this question, we randomly took a bunch of 100 data points from the sample of 10000 data points in each step and performed gradient descent.

We have also calculated $w_{sgd,t} = \frac{1}{t}\sum_{t'=1}^{t} w_{t'}$ to see how it converges after each iteration.

For the given data, the graph of $||w_t - w_{ML}||_2$ as a function of t is as shown below.



The graph of $||w_{sgd,t} - w_{ML}||_2$ as a function of t is as shown below.



From the above two graphs, we can observe that it takes so many iterations for the w_{sgd} to closely converge to the optimal points. Here we can see that w_{sgd} is trying to converge to w_{ML} after each step. w_t tries to converge to w_{ML} continuously at the start and after a certain number of iterations, it slightly diverges and converges randomly at the end; but $w_{sgd,t}$ is smoothly and continuously converging to w_{ML}. Though there are fluctuations in w_t, w_{sgd} always converged to w_{ML}. This can prove that stochastic gradient descent has the high probability of converging to the optimal solution.

Least Square Error obtained by w_{sgd} can be calculated as,

$$||X^T w_{sgd} - Y||_2^2 = (X^T w_{sgd} - Y)^T (X^T w_{sgd} - Y)$$

For our data, least square error with the obtained w_{sgd} for step size, $\eta_t = \frac{10^{-8}}{t}$ is calculated to be 413.59002733793045 and shows very high chances of going lower than this with the increase in iterations and modifying step size.

IV. Ridge Regression and gradient descent:

Ridge Regression aims at minimizing the loss function with regularization.

$$\text{Objective function: } \arg min_{w \in R^d} \sum_{i=1}^{n} (w^T x_i - y_i)^2 + \lambda ||w||_2^2$$

Here, the regularization term added is the l_2- norm of vector w.

$$\text{We have f(W)} = (X^T W - Y)^T (X^T W - Y) + \lambda (W^T W)$$

$$f(W) = W^T(XX^T)W - W^TXY - Y^TX^TW + Y^TY + \lambda(W^TW)$$

$$\text{delta } f(W) = 2(XX^T)W - 2(XY) + 2\lambda W^T$$

Gradient Descent update step for ridge regression algorithm is given as,

$$w_{t+1} = w_t - \eta_t [\text{delta } f(w_t)]$$

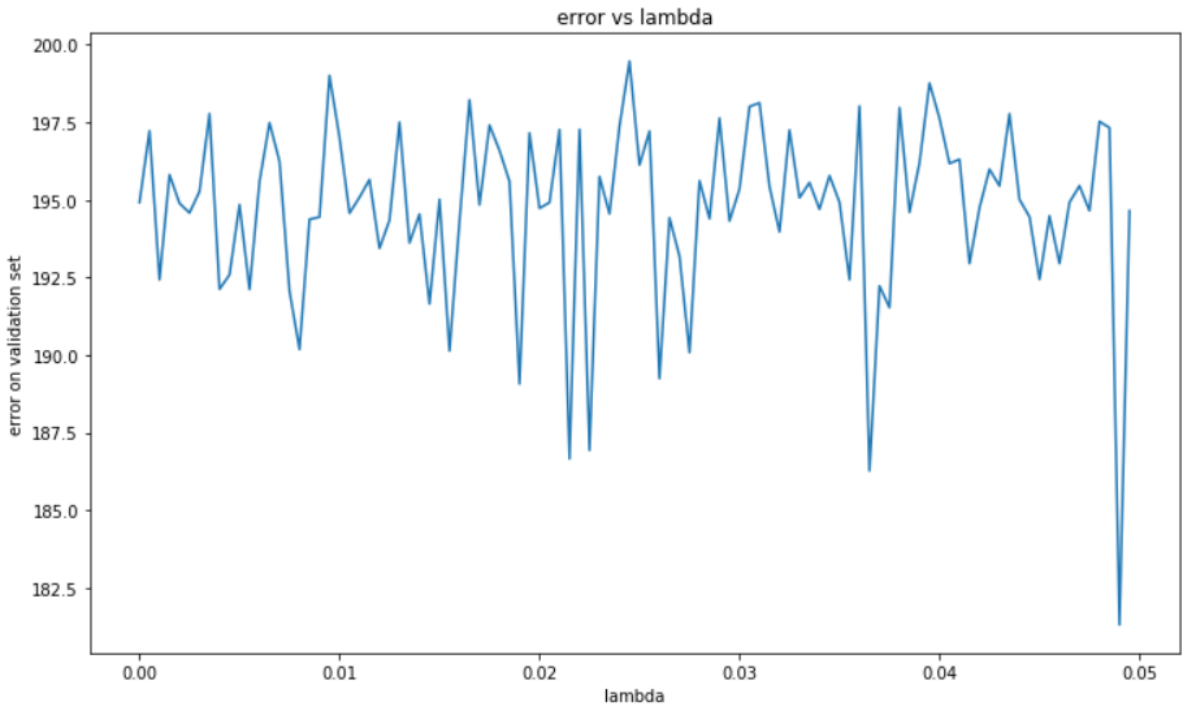
$$w_{t+1} = w_t - \eta_t [2(XX^T)W - 2(XY) + 2\lambda W^T]$$

We now perform a gradient descent process with the updated step as obtained above for a given value of λ. And the final w_T that we obtain will be the w_R that minimizes the objective function f(W).

We now want to choose the best λ by analyzing the effect of various choices of λ on the error of cross validation set. Cross validation is performed to get 80% as a training set and 20% as validation set. Then, we train on the training set and check the accuracy on the validation set.

Error on validation set is calculated as: $(X_{val}{}^T w_R - Y_{val})^T (X_{val}{}^T w_R - Y_{val}) + \lambda(w_R{}^T w_R)$ where w_R is the solution obtained from the training set that we cross validated.

Error in the validation set as a function of λ is shown in the graph below.



It has been observed that λ = 0.049 produced less error on the validation set. Hence, we choose λ as 0.049 to find our w_R to predict on the test data set (A2Q2Data_test.csv).

We can obtain w_R either through gradient descent or from analytical model as we know $w_R = (XX^T + \lambda I)^{\dagger} XY$ for λ = 0.049.

Dataset 'A2Q2Data_test.csv' is taken and the first 100 columns are assigned to features x1, x2,... x100, and the last column is assigned to y such that $X_{test} \in R^{dxn}$ and $Y_{test} \in R^{nx1}$.

We now compare the test error (squared error) on the test data A2Q2Data_test.csv of w_R $with$ w_{ML} which can be calculated as error(w) = $(X_{test}{}^T w - Y_{test})^T (X_{test}{}^T w - Y_{test})$.

It is observed that, error(w_{ML}) = 185.36365558489712 and $error(w_R)$ = 185.3532650116639. We can see that error(w_R) < error(w_{ML}). Errors (squared error) on train data set are also observed, error_train(w_{ML}) = 396.8644186272516 and error_train(w_R) = 396.8644257858158. Here error_train(w_{ML}) < error_train(w_R).

Mean squared errors: $MSE_R = 0.37070653002332776$ and $MSE_{ML} = 0.37072731116979424$

Mean squared errors on train data:
$MSE_{R,train} = 0.03968644257858158$ and $MSE_{ML,train} = 0.03968644186272516$

Hence the ridge regression algorithm is better as it produces less test error. In ridge regression, coefficients are shrunk so that minor contributed variables have their coefficients close to zero which minimizes the possibility of overfitting or underfitting the data by adding the regularization term because of which it is considered as the better approach as the least squares method fails to recognize overfitting and underfitting. We also saw that though the train error is more for w_R, test error is less for w_R than w_{ML} i.e., ridge regression performed better on the test data set when compared to the least squares method by accounting for overfitting and underfitting.