CS5691: Pattern Recognition and Machine Learning Assignment 2

Anirud N CE21B014

1 Question 1

1.1 Determine which probabilistic 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.

Assumption of data generation: each and every entry in the datasheet, i.e., each entry of a data point is generated by a Bernoulli Distribution, i.e., 0 or 1. Also the assumption that each column of the dataset is independent of the other. i.e., the correlation of each column with the other columns is assumed to be 0 in Bernoulli Model
The dataset is basically: $X \leftarrow (x_i)$ belongs to R^d
where $i = 1, 2, ..n$ and n is the number of data points
Each entry of x_i is either 0 / 1, ie:
$x_i[j] = 0/1$ for $j = 1, 2..d$,
where d is the dimension of each data point
Let i denote the index for ith datapoint ie x_i
K is the total number of clusters ($K = 4$)
Let k denote the index for cluster ie $k = 1, 2..K$
Let j denote the entry of the vector x_i ie $j = 1, 2..d$ the log-likelihood is given by :

$$\sum_{i=1}^{n} \log \sum_{k=1}^{K} \pi_k * \Pi_{j=1}^{d} (p_j^k)^{x_{ij}} (1 - p_j^k)^{1-x_{ij}}$$

The modified Log Likelihood is given by:

$$\sum_{i=1}^{n} \sum_{k=1}^{K} \lambda_i^k \log \frac{\pi_k * \Pi_{j=1}^{d} (p_j^k)^{x_{ij}} (1 - p_j^k)^{1-x_{ij}}}{\lambda_i^k}$$

This can be further simplified as :

$$\sum_{i=1}^{n} \sum_{k=1}^{K} \lambda_i^k (\log \pi_k - log \lambda_i^k + \sum_{j=1}^{d} (x_{ij} \log p_j^k) + (1 - x_{ij}) \log (1 - p_j^k))$$

1) Fix λ and maximise over θ (maximisation step)
differentiate the modified log-likelihood function over p_j^k

$$\sum_{i=1}^{n} \lambda_i^k \left(\frac{x_{ij}}{p_{jk}} - \frac{1 - x_{ij}}{1 - p_{jk}} \right) = 0 \tag{1}$$

Solving this gives:

$$p_j^k = \frac{\sum_{i=1}^{n} \lambda_i^k * x_{ij}}{\sum_{i=1}^{n} \lambda_{ik}} \tag{2}$$

Similarly we get :

$$\pi_k = \frac{\sum_{i=1}^{n} \lambda_k^i}{n} \tag{3}$$

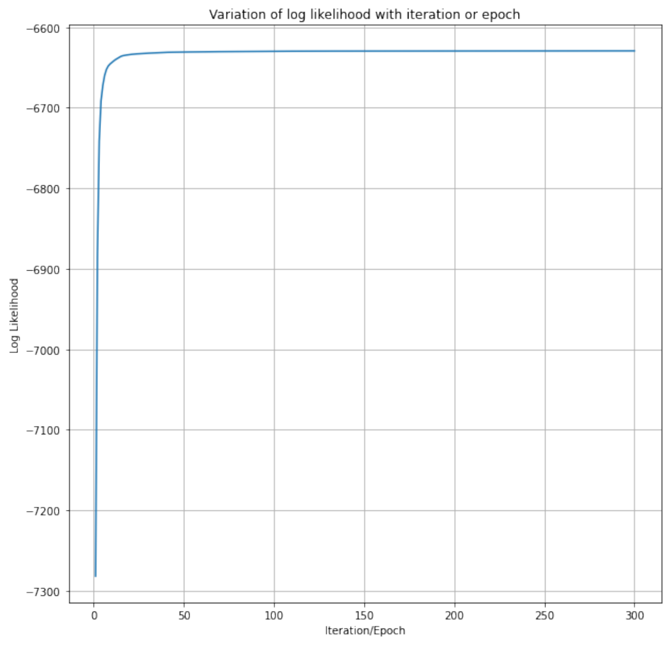


Figure 1: Bernoulli Mixture model - Variation of log likelihood with iteration - averaged over runs

2) Fix θ and maximise over λ (expectation step)
Differentiate with respect to λ_k^i

$$\lambda_i^k = \frac{\pi_k * \Pi_{j=1}^{d}(p_j^k)^{x_{ij}}(1-p_j^k)^{1-x_{ij}}}{\sum_{k=1}^{K}\pi_k * \Pi_{j=1}^{d}(p_j^k)^{x_{ij}}(1-p_j^k)^{1-x_{ij}}} \tag{4}$$

The EM Algorithm:

- Random Initialise p_k- R^d for k = 1,2..K and p $\in$ 0,1
- Random Initialise $\pi_k = \frac{1}{K}$ for all k=1,2..K
- Expectation step: Using equation (4), calculate all the λ_k^i values
- Maximisation step: Using equation (2,3), calculate the values for p_j^k and π_k
- Repeat the Expectation and maximization steps until convergence, i.e., when the difference in parameters θ becomes less than the allowable tolerance.

1.2 Assume that the same data was in fact 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.

$$p(\mathbf{x}|\mu,\mathbf{\Sigma}) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}}\exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\mu)\right) \tag{5}$$

This is the equation for multi-variate gaussian pdf. Since the data has more than 1 dimension, we have to use a covariance matrix that accounts even for the correlation between each column of the dataset.
EM Algorithm:

- Random initialize the mean vectors for each cluster and the covariance matrix(I just initialized the latter to the identity matrix in my code for convenience)
- Random Initialise $\pi_k = \frac{1}{K}$ for all k=1,2..K

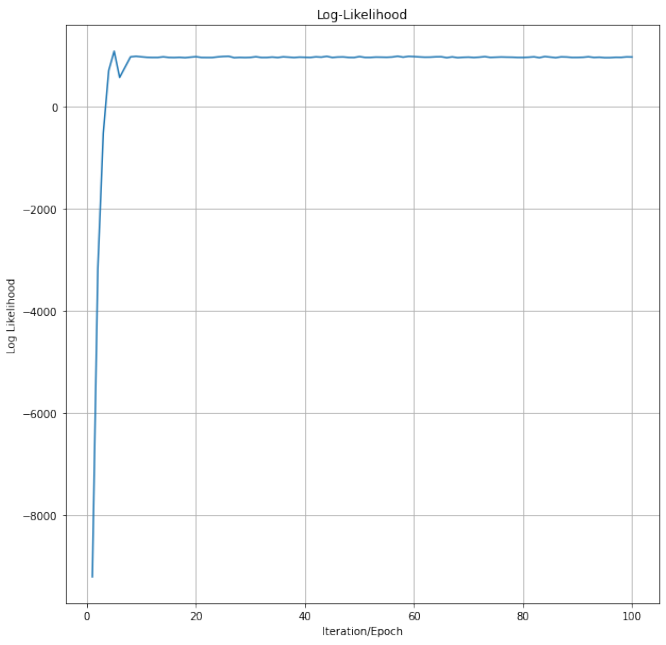


Figure 2: Variation of log likelihood vs iteration for Multivariate GMM - averaged over many iterations

- Expectation step:

$$\lambda_{nk} = \frac{\pi_k \mathcal{N}(\mathbf{x}_n|\mu_k, \mathbf{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}_n|\mu_j, \mathbf{\Sigma}_j)} \tag{6}$$

- Maximisation

$$N_k = \sum_{n=1}^{N} \gamma(z_{nk}) \tag{7}$$

$$\pi_k^* = \frac{N_k}{N} \tag{8}$$

$$\mu_k^* = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) \mathbf{x}_n \tag{9}$$

$$\Sigma_k^* = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk})(\mathbf{x}_n - \mu_k)(\mathbf{x}_n - \mu_k)^T \tag{10}$$

Comparing the plots for Bernoulli Model and Gaussian Mixture Model:

- The steepness in an increase of Log-likelihood for GMM is higher than Bernoulli
- GMM attains higher Log likelihood value upon convergence
- Both the plots start to converge faster, ie starting from iteration of around 10.
- The smoothness of the Bernoulli curve is higher than the GMM curve
- It was observed that the implemented GMM is more sensitive to initializations than Bernoulli.
- as a result, the variance among the random 100 initialisation was higher in GMM than in the bernoulli model
- This is because GMM has more parameters to optimize for when compared to the Bernoulli model. The fewer the parameters, the more stable the algorithm concerning the initializations but the lesser the flexibility

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

Objective : sum of Norm of distance of each datapoint from its cluster center

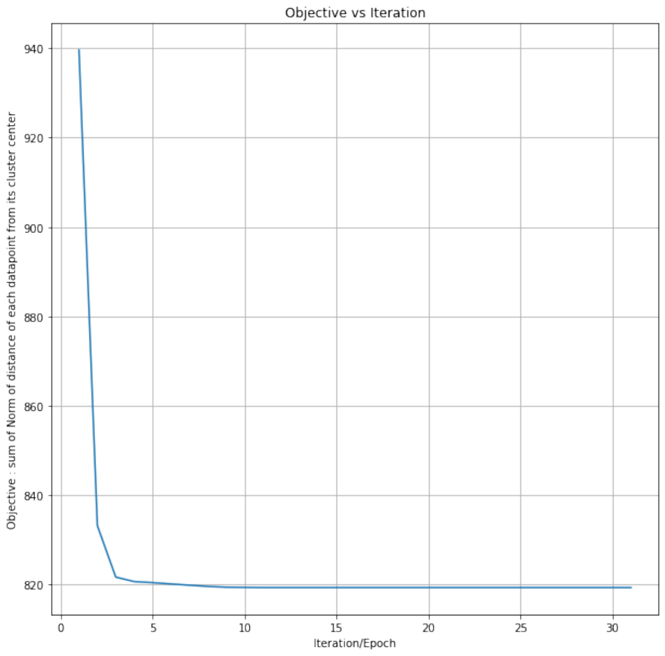


Figure 3: K-Means Clustering objective vs iteration plot

The value of objective upon convergence is found to be **819.2763**

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

We found out soft clustering, ie the λ_{ik} values. Using these values, let us find the assignment matrix ie the clusters each datapoint belongs to. Cluster that the datapoint i belongs to is defined as

$$z_i = \arg\max_k \lambda_{ik}$$

After finding the clusters and the datapoints belonging to each cluster, we can find the mean or centerpoints of each clusters or centroids by simple averaging.
Our Objective function : O is defined as :

$$O = \sum_{i=1}^{N} ||X_i - \mu_{zi}||$$

where μ_{zi} denotes the center point of the cluster z_i that the datapoint i belongs to
The values for the objective functions for different algorithms:
1) Bernoulli Model: **822.2071521700802**
2) Gaussian Mixture Model: **926.1802812329421**
3) K-Means Clustering: **819.2762894622942**
Purely in terms of the distance of each datapoint from the centroid of the cluster, K means is best followed by the Bernoulli Model and the least performing model is the Gaussian Mixture model.
But then K means is a hard clustering algorithm and it can divide only voronoi space data.
I would choose the Bernoulli Model approach for this dataset because it is a probabilistic model with a better assumption about how the data is generated(0 and 1) over the Gaussian Mixture Model.
However, it is important to note one key drawback of the Bernoulli Model: It is the assumption that each column of the dataset is independent of the other. i.e., the correlation of each column with the other columns is assumed to be 0 in Bernoulli Model, which may not always be correct. Gaussian Mixture Model, on the other hand, defines a correlation matrix and optimises over this matrix. This is one reason why we could find a steep increase in the plots for log-likelihood vs. iterations.

2 Question 2

2.1 Obtain the least squares solution wML to the regression problem using the analytical solution.

The analytical solution for w which fits the data is calculated using the direct formula $W = (XX^T)^{-1}XY$ This could be found in the code

2.2 Code the gradient descent algorithm with suitable step size to solve the least squares algorithms and plot wt wML 2 as a function of t. What do you observe?

The gradient descent was performed using the equation:

$$w_{t+1} = w_t - \frac{\alpha}{t} * \Delta f(w_t) \tag{11}$$

$$w_{t+1} = w_t - \beta * \Delta f(w_t) \tag{12}$$

where

$$\Delta f(w_t) = 2XX^T w - 2XY$$

here eq(11) specifies that the Learning rate is dependent and change with time, with LR proportional to 1/t. whereas eq(12) specifies a constant Learning rate. I have implemented both cases to show that the eq(11), i.e., time-dependent learning rate is better.
Utilizing a learning rate that diminishes proportionally to 1/t ensures a gradual decrease, allowing for smaller steps as the iteration number increases. This approach is selected eto achieve convergence and maintain the efficiency of each step, ensuring they remain impactful throughout the learning process.
I had iterated over different values of alpha and beta to find an optimal value of these for better convergence. (the process is in code notebook).

Learning rate log refers to the learning rate proportional to 1/t (as intergal 1/t is log). From the Figure 4, it is clear that even for 30,0000 iterations, the Time dependent learning rate converges to a smaller value faster than the constant learning rate.
As you can see in these plots, initially, the error, i.e., the $|w_ml - w_analytic|$, is higher, but as time increases, it learns to move closer to the optimal w_ml value, and so this error decreases over time. Divergence was observed for a very high learning rate. So, adjusting an optimal learning rate is essential to ensure convergence.

2.3 Code the stochastic gradient descent algorithm using batch size of 100 and plot wt wML 2 as a function of t. What are your observations?

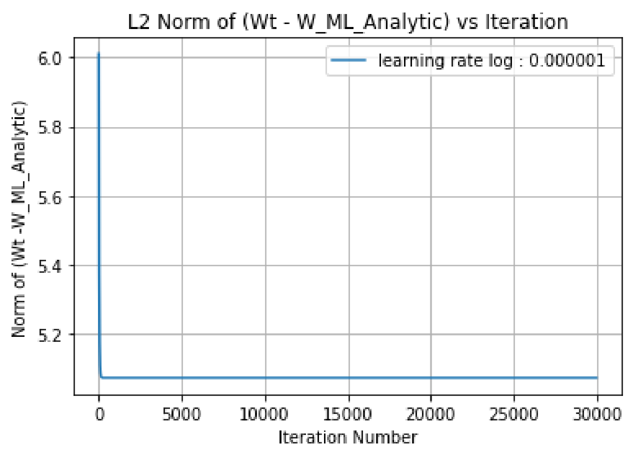
The batch size was set to 100. For each iteration, out of the entire dataset, randoml, a batch of 100 data points was chosen. Then, this was pretended to be the dataset, and gradient descent was performed with this batch alone. This was done for every iteration, and finally, the weights from stochastic gradient descent were obtained by averaging all the weights. The learning rate was set to be $0.00001/t$, i.e., a time-dependent learning rate for better convergence.
Refer to **Figure 5**
This SGD significantly reduces our computation demands, as we only need to compute matrices of dimensions 100 instead of 10000. Moreover, this alteration doesn't negatively impact our error, with the difference being minimal, thus rendering this algorithm highly efficient.

2.4 Code the gradient descent algorithm for ridge regression. Cross-validate for var ious choices of and plot the error in the validation set as a function of . For the best chosen, obtain wR. Compare the test error (for the test data in the le A2Q2Data test.csv) of wR with wML. Which is better and why

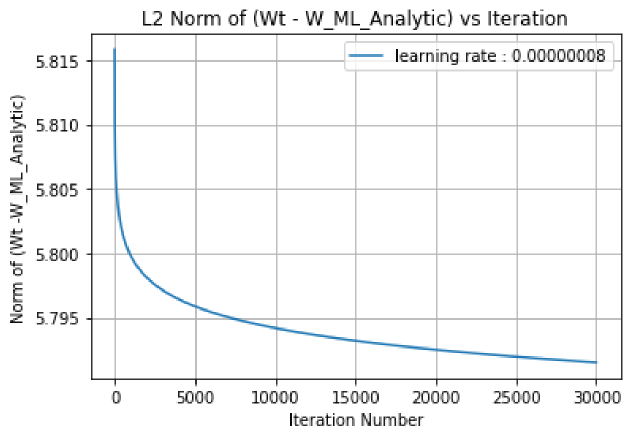
The update equation in ridge regression is given by:

$$w_{t+1} = w_t - \frac{\alpha}{t}(2XX^T w - 2XY) + 2\lambda w_t \tag{13}$$

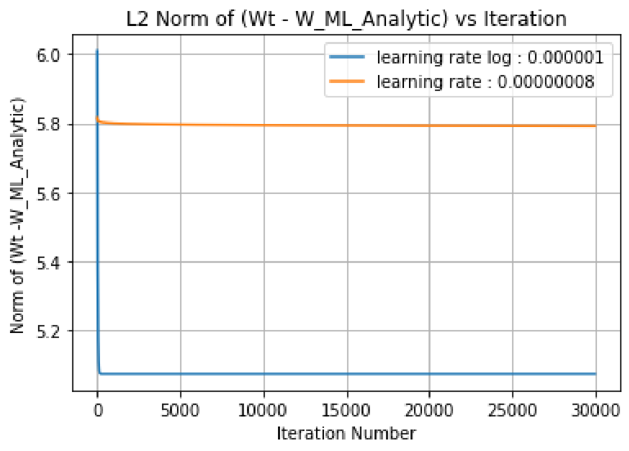
where α/t is the learning rate.
Finding λ is done through Cross Validation
There are two ways to do cross validations:



(a)



(b)



(c)

Figure 4: (a) Learning rate time dependent - alpha/t (b) Constant Learning rate beta (c) Comparison Plots between (a) and (b)

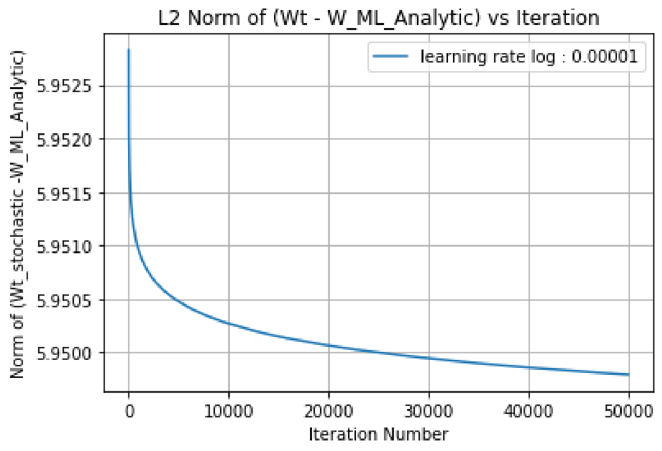
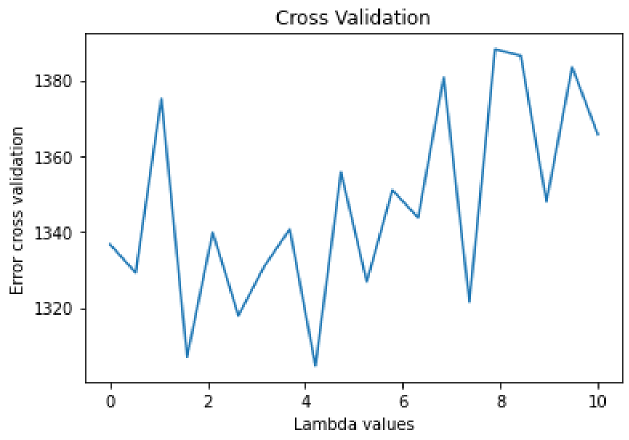


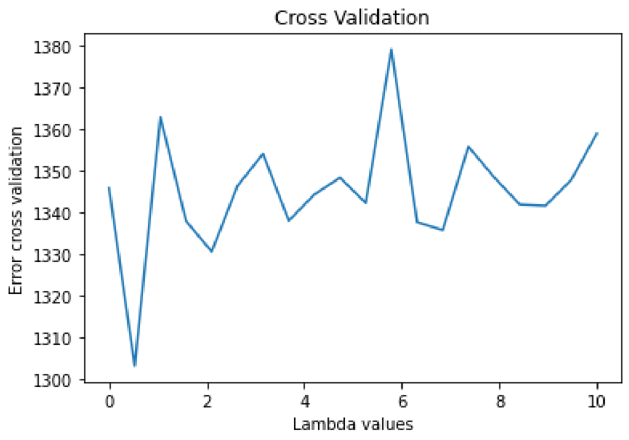
Figure 5: Stochastic gradient descent plot

1) **80-20 split:** In this the dataset was divided into 80 percent Train and 20 percent validate and then λ was found so as to minimise error in the validate data after fitting the weights over train data.
2) **K Fold Cross Validation:** In this, I set k=5. Splitting the dataset into K folds and then iteratively training the model on K-1 folds and testing it on the remaining fold. This process is repeated K times, with each fold being used as the test set exactly once. Then the average of all the errors is obtained, and the best λ is found by minimizing the average error.

The codes for both the above procedures have been implemented in the code notebook.
The search space for λ was from 0 to 10.
The optimal value for λ for 80-20 split case was found to be 4.2105263157894735
The optimal value for λ for K fold cross-validations was found to be 0.5263157894736842
From Figure 7, it is evident that ridge regression slightly outperforms the case of Wml. Ridge regression performs better

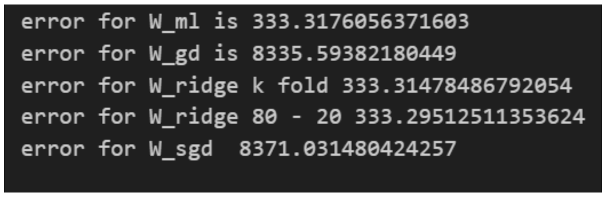


(a)



(b)

Figure 6: (a) 80-20 split (b) K fold cross validation k =5



```
error for W_ml is 333.3176056371603
error for W_gd is 8335.59382180449
error for W_ridge k fold 333.31478486792054
error for W_ridge 80 - 20 333.29512511353624
error for W_sgd  8371.031480424257
```

Figure 7: The values of Errors from the different gradient descent approaches

than the analytical solution as the 'noise' in the input is removed out, and only the important parameters that influence the output are remaining. Gradient Descent and Stochastic Gradient descent both have nearly similar values of error which proves the efficiency of stochastic gradient descent. Further this implies that these has to be trained for more iteration to reach closer to the Wml case.

Note: I had centered and standardised the data for better interpretations