Assignment - 02
CS5691 Pattern Recognition and Machine Learning

Ramasamy Kandasamy
CS22M068

October 26, 2022

Question - 1

Code: `q1.py`

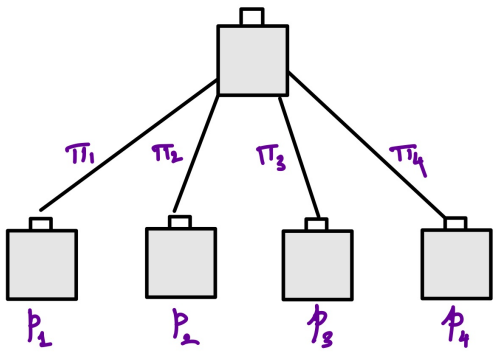
i. Determine which probabilisitic mixture could have generated this data (It is not a Gaussian mixture). Derive the EM algorithm for your choice of mixture and show your calculations. Write a piece of code to implement the algorithm you derived by setting the number of mixtures K = 4. Plot the log-likelihood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

Code: `em_bernoulli.py`

Notations:

N	Size of input dataset.
D	Dimension of the dataset.
K	Number of clusters in the mixture model.
i	Index on the datapoints.
j	Index for dimension.
k	Index for clusters.

My choice of mixture model for this problem is a mixture of four multivariate Bernoulli distribution. This is depicted in the figure below.



In the first step one of the model is chosen with a probability π_k, and $\sum_{k=0}^{3} \pi_k = 1$. After one of the four models is chosen we proceed to the second step where the data is generated by the model. Each model has a parameter p_i which is a vector/array of dimension 50. The input data is a set of array x_i of dimension 50. For simplicit, I make an assumption that each position within the array is generated independently of each other by a Bernoulli trial with success probability p_k. Therefore the probability that a data point x_i is generated by the model k would be:

$$P(x_i \mid p_k) = \prod_{j=1}^{D} P(x_{i,j} \mid p_{k,j}) = \prod_{j=1}^{D} p_{k,j}^{x_{i,j}} \times (1 - p_{k,j})^{1-x_{i,j}}$$

The probability of observing the datapoint x_i is given by:

$$P(x_i \mid p_0 \cdots p_K) = \sum_{k=1}^{K} \pi_k \times P(x_i \mid p_k) = \sum_{i=1}^{K} \pi_k \times \prod_{j=1}^{D} \left(p_{k,j}^{x_{i,j}} \times (1 - p_{k,j})^{1 - x_{i,j}} \right)$$

Further, I also assume that each datapoint is generated independently. With this assumption the likelihood of observing the data given paraments is given by:

$$L(X \mid \theta) = \prod_{i=1}^{N} \left(\sum_{k=1}^{K} \pi_k \times P(x_i \mid p_k) \right)$$

Log likelihood is given by:

$$logL(X \mid \theta) = \log \left(\prod_{i=1}^{N} \left(\sum_{k=1}^{K} \pi_k \times P(x_i \mid p_k) \right) \right) = \sum_{i=1}^{N} \left(\log \left(\sum_{k=1}^{K} \pi_k \times P(x_i \mid p_k) \right) \right)$$

Now, we use Jensen's inequality and use modified log-likelihood instead of the original log-likelihood. Now our goal to maximize this modified log-likelihood which is given by:

$$mod\_logL(X \mid \theta) = \sum_{i=1}^{N} \sum_{k=1}^{K} \left(\lambda_i^k \left(\sum_{j=1}^{D} x_{i,j} \log p_{k,j} + (1 - x_{i,j}) \log(1 - p_{k,j}) \right) + \log \pi_k - \log \lambda_i^k \right)$$

First, differentiating this with respect to $p_{k,j}$ gives:

$$\sum_{i=1}^{N} \lambda_i^k \left(x_{i,j} \frac{1}{p_{k,j}} - (1 - x_{i,j}) \frac{1}{1 - p_{k,j}} \right) = 0.$$

$$\implies \sum_{i=1}^{N} \lambda_i^k \left(x_{i,j}(1 - p_{k,j}) - (1 - x_{i,j}) p_{k,j} \right) = 0.$$

Simple algebraic simplification gives:

$$p_{k,j} = \frac{\sum_{i=1}^{N} \lambda_i^k x_{i,j}}{\sum_{i=1}^{N} \lambda_i^k}$$

where, P is matrix of $p_{k,j}$s, X is the input data and $\mathbb{1}_{N \times D}$ is a matrix of ones of dimension $N \times D$.
The value of paramenters π_k is obtained from λ_k^i which inidcates the probability that the i-th datapoint was generated by the k-th model. Since π_k is the probability that any datapoint is generated by the k-th model, it is given by:

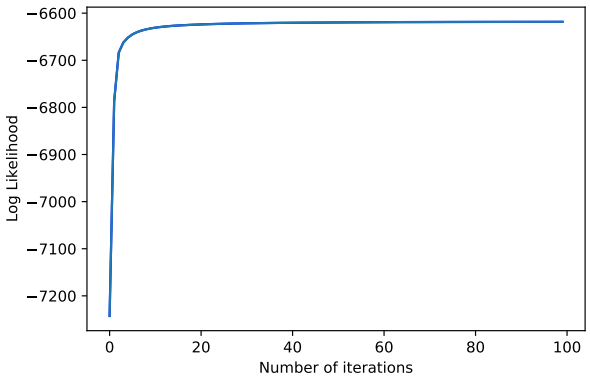
$$\pi_k = \sum_{i=1}^{N} \lambda_k^i$$

where Π is vector of π_ks and $\mathbb{1}_N$ is a vector of ones of length N.
Now, λ_k^i is the probability that the i-th datapoint is generated by k-th model. Therefore this is given by:

$$\lambda_k^i = \frac{P(x_i \mid p_k)\pi_k}{\sum_{k=1}^{K} P(x_i \mid p_k)\pi_k}$$

These equations are used in matrix form in the code for EM algorithm, which is in the file `em_bernoulli.py`

The plot of log-likelihood (averaged over 100 random initializations) as function of iterations is as follows

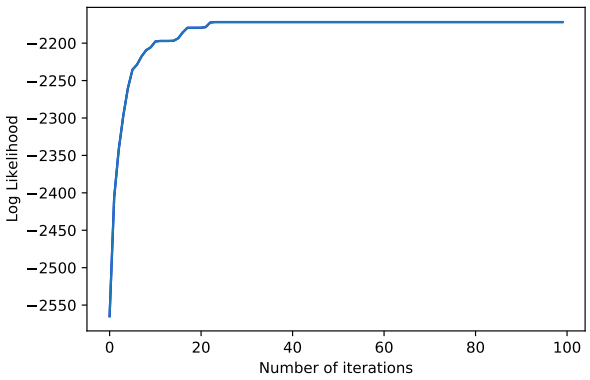


ii. Assume that the same data was infact generated from a mixture of Gaussians with 4 mixtures. Implement the EM algorithm and plot the log-likelihood (aver- aged over 100 random initializations of the parameters) as a function of iterations. How does the plot compare with the plot from part (i)? Provide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

Code: `em_gauss.py`

Here I assumed that the data was generated by a mixture of multivariate Gaussian distribution. The implementation is in the file `em_gauss.py`.

The plot of log-likelihood (averaged over 10 random initializations) as function of iterations is as follows.

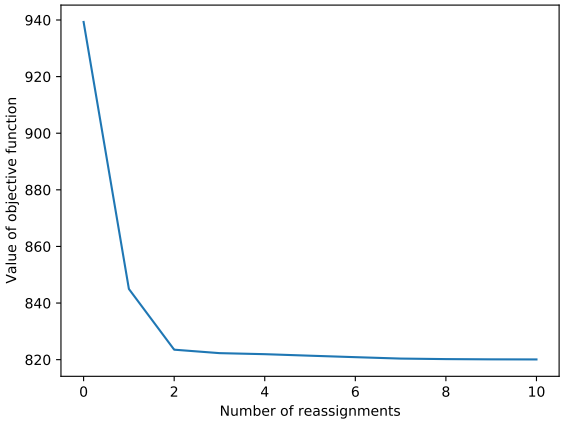


The log-likelihood plot reaches a plateau within 100 itertions. However the log-likelihood is around $-2,200$ which significantly higher than what is obtained with the previous model in Q1-i. I think this might be because of overfitting since there are lot more parameters in this model.

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

Code: `k_means.py`

The K-means algorithms was run for $K = 4$ and the objective function was plotted.



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

I would choose the algorithm in Q1-i, which is based on multivariate Bernoulli distribution. This because first mixture models are better than K-means clustering particularly when the data are not clearly separable into clusters. I will choose Q1-i over Q1-ii because according GMM the data should have values other than 0 and 1, however this not the case, therefore I will choose the model in Q1-i.

Question-2

Code: `q2.py`

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

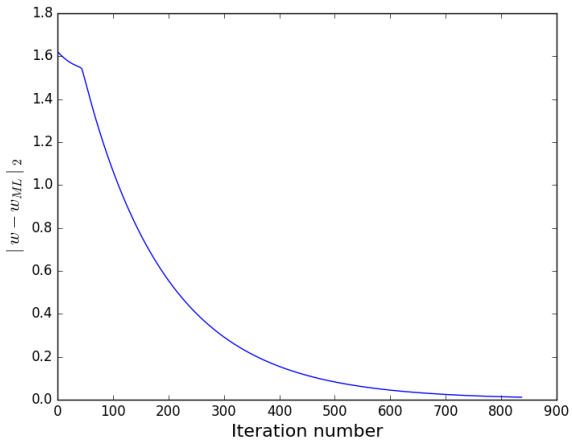
Code: `q2.py`

```
Q2-i: First five values of w_ml
[[-0.00784961]
 [-0.01367153]
 [-0.00361656]
 [ 0.00264909]
 [ 0.18855145]]
Q2-i: Last five values of w_ml
[[-0.00943541]
 [ 0.01829054]
 [-0.00116999]
 [-0.00261599]
 [-0.00858616]]
```

Code the gradient descent algorithm with suitable step size to solve the least squares algorithms and plot $| w^t - w_{ML} |^2$ as a function of t. What do you observe?

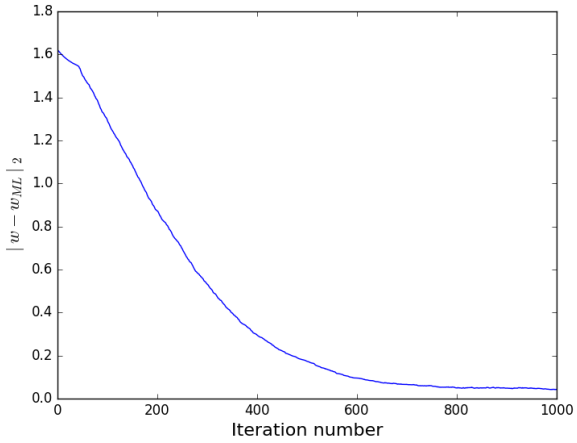
Code: `grad_descent.py`

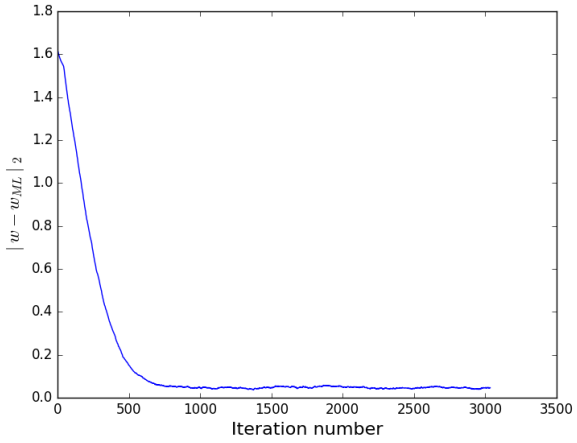
The plot of $| w^t - w_{ML} |^2$ is as follows:



w^t approached w_{ML} smoothly, i.e. monotonously.

Code the stochastic gradient descent algorithm using batch size of 100 and plot $|w^t - w_{ML}|^2$ as a function of t. What are your observations?
Code: `stochastic_grad_descent.py`
The plot of $|w^t - w_{ML}|^2$ is as follows:



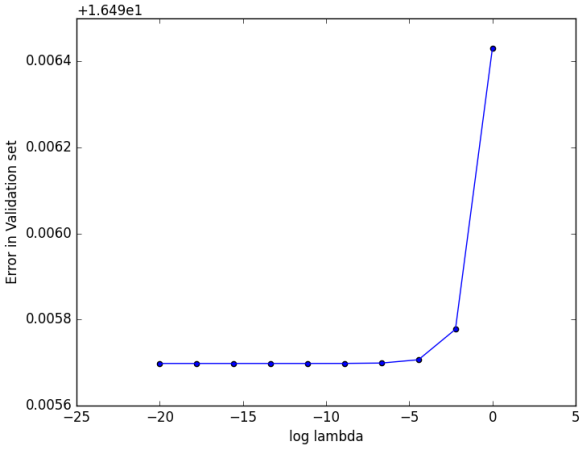


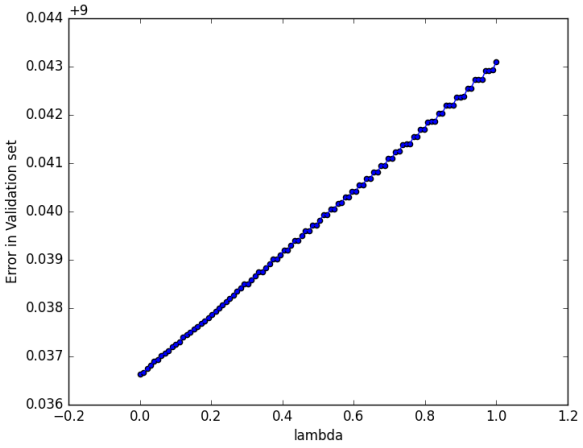
Unlike the normal gradient descent stochastic gradient descent approaches w_{ML} in a not so smooth manner, i.e. the plot is not monotonously decreasing.

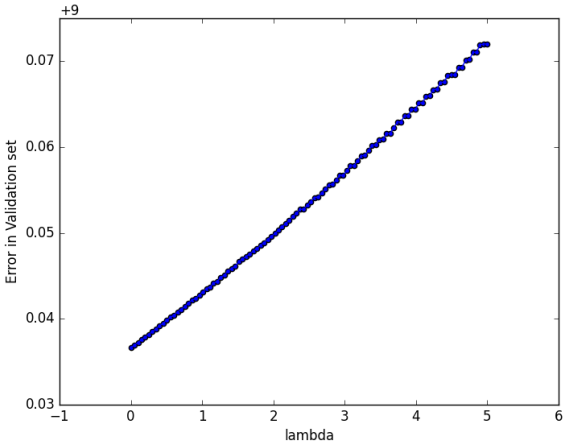
Code the gradient descent algorithm for ridge regression. Cross-validate for various choices of λ and plot the error in the validation set as a function of λ. For the best λ chosen, obtain w_R. Compare the test error (for the test data in the file A2Q2Data test.csv) of w_R with w_{ML}. Which is better and why?

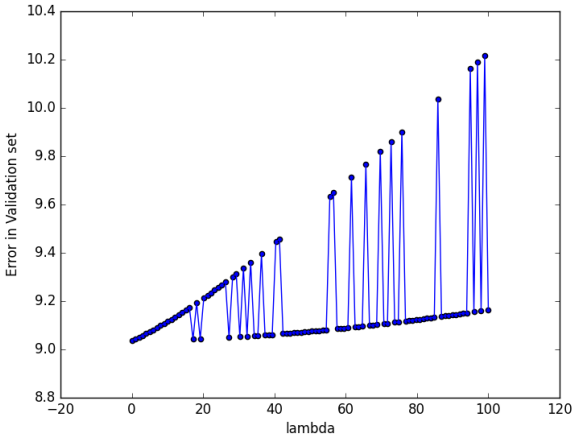
Code: `grad_descent_ridge.py`, `cross_validation.py`

The plot of error on validation set for various values of λ are as follows









The value of lambda for which the error was minimum was $\lambda = 0$. The test error of w_R and w_{ML} are as follows:

```
min lambda:
0.0
Error in test data for w_R is:
184.33438512662875
Error in test data for w_ML is:
185.36365558489564
```

Both are equivalent because they have very simillar test error and also in w_R uses $\lambda = 0$ which equivalent to having w_{ML}.