Note to the Professor and TAs: Python codes written for this assignment include the following libraries:Numpy, Matplotlib, Seaborn (similar to Matplotlib but helps in better visualization).

1. Question 1(i)

Solution: We will use Bernoulli mixture model and derive the EM algorithm for the same. Our likelihood function for parameters $\theta = \{p_1 \text{ to } p_4, \pi_1 \text{ to } \pi_4\}$ is as follows:

$$\begin{aligned} L(\theta; x_1, \ldots, x_n) &= \prod_{i=1}^{n} P(x_i; \theta) \\ &= \prod_{i=1}^{n} \sum_{k=1}^{4} \pi_k \cdot P(x_i; p_k) \\ &= \prod_{i=1}^{n} \sum_{k=1}^{4} P(x_i, z_i = k; \theta) \end{aligned}$$

Taking log on both sides,

$$\begin{aligned} &= \sum_{i=1}^{n} log \sum_{k=1}^{4} P(x_i, z_i = k; \theta) \\ &= \sum_{i=1}^{n} log \sum_{k=1}^{4} \lambda_k^i \frac{P(x_i, z_i = k; \theta)}{\lambda_k^i} \end{aligned}$$

Using Jensen's inequality,

$$\begin{aligned} &\geq \sum_{i=1}^{n} \sum_{k=1}^{4} \lambda_k^i log \left(\frac{P(x_i, z_i = k; \theta)}{\lambda_k^i} \right) \\ &\geq \sum_{i=1}^{n} \sum_{k=1}^{4} \lambda_k^i log P(x_i, z_i = k; \theta) - \lambda_k^i log \lambda_k^i \\ &\geq \sum_{i=1}^{n} \sum_{k=1}^{4} \lambda_k^i log(\pi_k \cdot P(x_i; p_k)) - \lambda_k^i log \lambda_k^i \\ &\geq \sum_{i=1}^{n} \sum_{k=1}^{4} \lambda_k^i log(\pi_k \cdot (p_k)^{x_i} \cdot (1 - p_k)^{1 - x_i}) - \lambda_k^i log \lambda_k^i \end{aligned}$$

Taking partial derivative w.r.t p_k and equating to 0.

$$\frac{\partial L}{\partial p_k} \implies \sum_{i=1}^{n} \left(\frac{\lambda_k^i x_i}{p_k} + \frac{\lambda_k^i \cdot (1 - x_i) \cdot -1}{1 - p_k} \right) = 0$$

$$\implies (1 - p_k) \sum_{i=1}^{n} \lambda_k^i x_i + p_k \sum_{i=1}^{n} \lambda_k^i (x_i - 1) = 0$$

$$\hat{p}_k = \frac{\sum_{i=1}^{n} x_i \lambda_k^i}{\sum_{i=1}^{n} \lambda_k^i}$$

From GMM, we know that

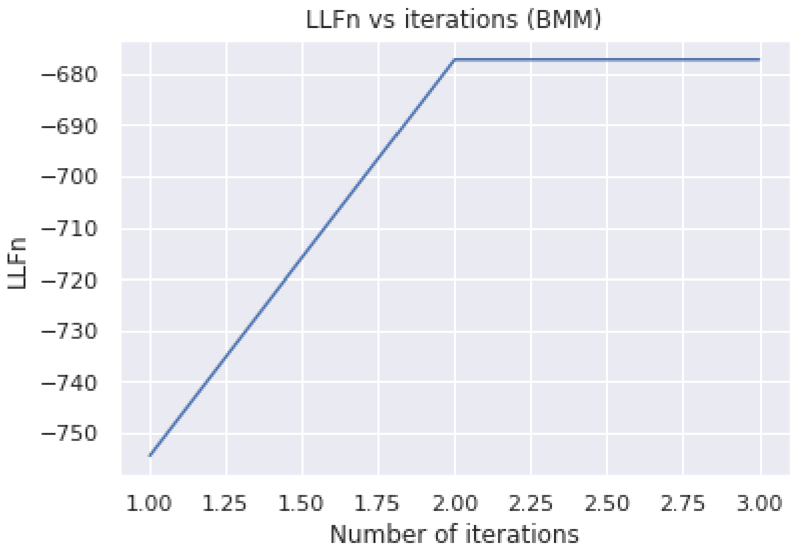
$$\lambda_k^i = \frac{P(x_i | z_i = k; \theta) \cdot P(z_i = k; \theta)}{\sum_{l=1}^{K} P(x_i | z_i = l; \theta) \cdot P(z_i = l; \theta)}$$

$$\lambda_k^i = \frac{(p_k)^{x_i} \cdot (1 - p_k)^{1 - x_i} \cdot \pi_k}{\sum_{l=1}^{4} (p_l)^{x_i} \cdot (1 - p_l)^{1 - x_i} \cdot \pi_l}$$

Our π_k's will be the same as GMMs.

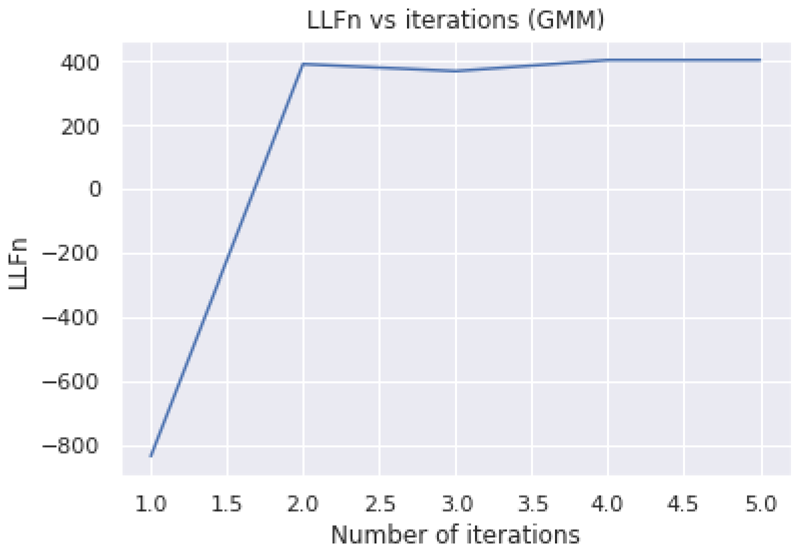
$$\hat{\pi}_k = \frac{\sum_{i=1}^{n} \lambda_k^i}{n}$$

After averaging over 100 random initialization, the log-likelihood as function of iterations for our Bernoulli mixture model is as follows:



2. Question 1(ii)

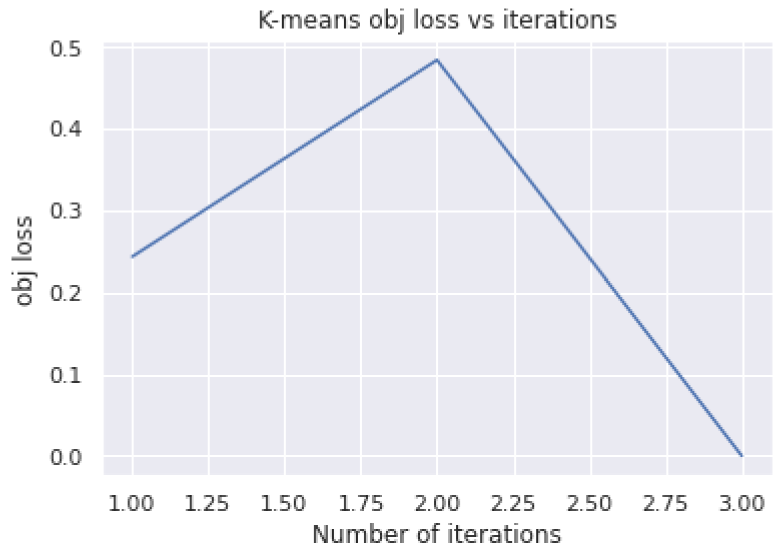
Solution: After averaging over 100 random initialization, the log-likelihood as function of iterations for our GMM is as follows:



We see that both BMM and GMM maximize likelihoods. However, BMM tries to cluster datapoints around clusters with different probabilities whereas GMM tries to center all clusters either around zeroes or around ones.

3. Question 1(iii)

Solution: Cluster centers of k-means are as follows: [0, 0, 0, 1]. Objective of K-means as a function of iterations.



4. Question 1(iv)

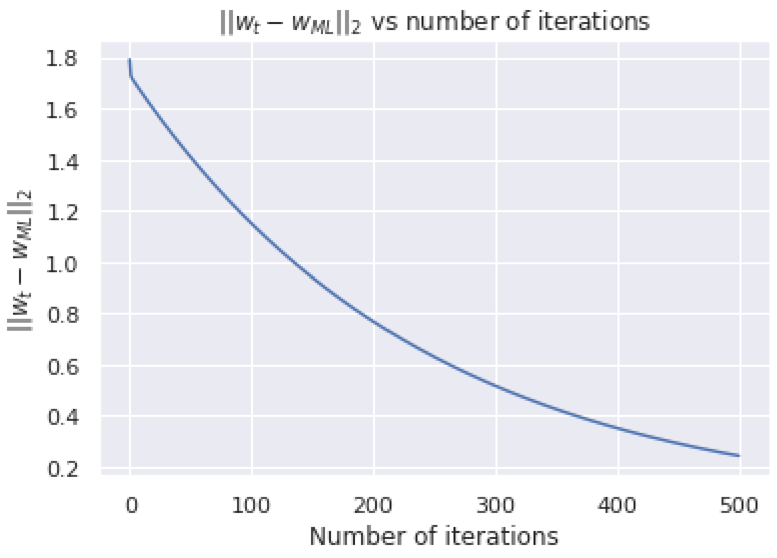
Solution: Clearly, we can't use k-means to cluster binary data. This is because the centers are still binary data. After initial cluster centers are chosen, in every iteration, every datapoint will always be at some integer distance from each of the cluster centers. This will lead to arbitrary cluster assignment to datapoints. Using Euclidean distance, it is infeasible to break this pattern in any meaningful way.

Similarly, we can see that both BMM and GMM maximize the likelihood. However, BMM performs much better than GMM. This is because in GMM, all the four cluster means and variances are close to each other. Since we have 411 ones and 589 zeroes in our dataset, the cluster means for GMM will be either closer to 0.58 or 0.411. Essentially, this tells us that GMM can't separate datapoints into different clusters. On the other hand, in BMM, the probabilities for the clusters, are more varied and distributed, thereby, clustering the datapoints across the four clusters and not centered around a single probability value. Hence, BMM should be used to separate binary data into suitable clusters.

5. Question 2(ii)

Solution: Our training dataset $X \in \mathbb{R}^{10000\times100}$. Since our data isn't centered, we can either center it and then run our linear regression or add the bias column in our training and validation dataset. This bias refers to nothing but the intercept c in our standard $y = mx + c$ equation.

In this sub-part of the question, we are using gradient descent where our batch size is the whole training dataset. Training is done by setting learning rate to 0.05 and number of iterations to run to 500. From the graph below, we see that there is a smooth decrease in $||w_t - w_{ML}||_2$ as our number of iterations increases. This smoothness is expected since while gradient calculation and update is taken into account w.r.t all the datapoints. This leads to a more stable decrease in 2-norm loss of weights. Let's say that there is a true 'weight vector' w_{true} for our data distribution, i.e., all training + validation dataset, and all those datapoints which we haven't encountered yet. By using batch size equal to the training set, $||w_{full_batch} - w_{true}||_2$ will be much smaller than training the regression algorithm with batch size less than training set.



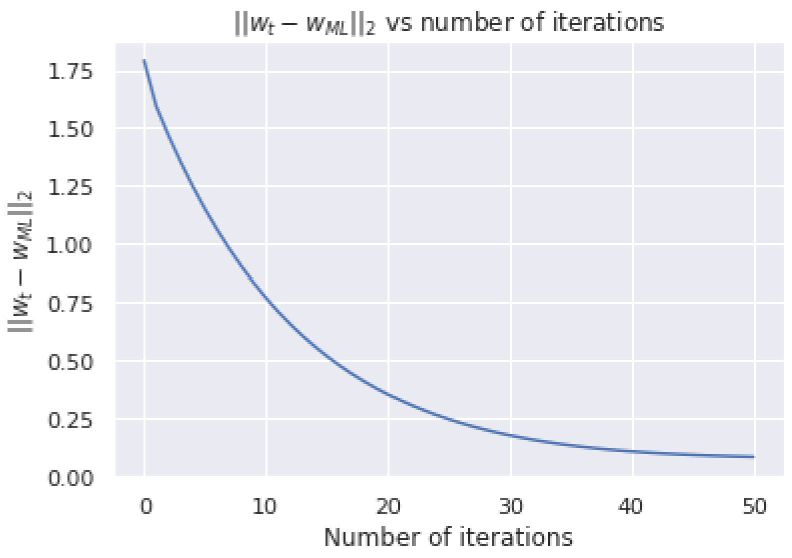
6. Question 2(iii)

Solution: In this sub-part of the question, our batch size is equal to 100 with learning rate set to 0.01 and number of iterations to 50. From the graph below, we see that the convergence rate is much faster than gradient descent with batch size equal to the training dataset. Some reasons for why this could happen:

1. When batch size is equal to 100, our gradient updates happens w.r.t these 100 datapoints. In every iteration, we have a new set of datapoints, thereby, new gradient directions. This variability might help in faster convergence. On the other hand, in normal gradient descent of batch size equal to the training

set, redundant computations might occur where gradients are recomputed for similar examples before each update.

2. While normal gradient descent worked well in our example, this might not be the case for large datasets. Storing all the training examples on memory to do vectorized computations might be infeasible.



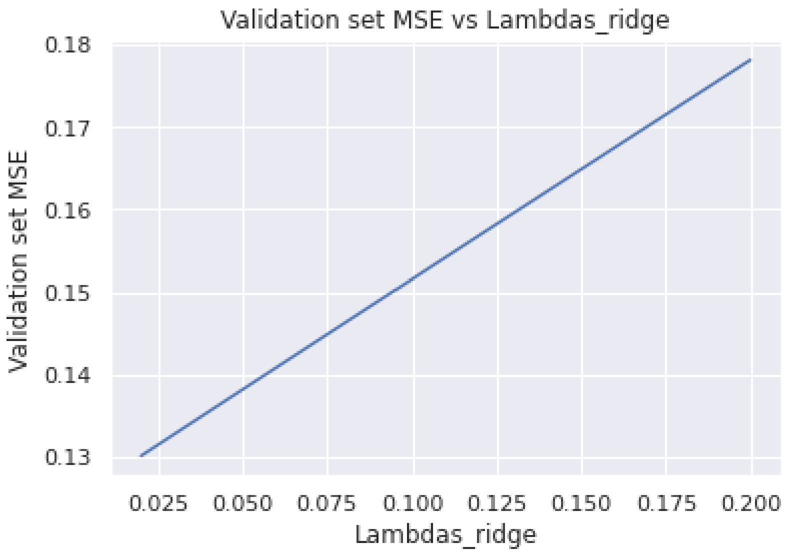
7. Question 2(iv)

Solution: In this experiment, we set our λ from 0.02 to 0.2 in increments of 0.02. After running the experiment, we find that $\lambda = 0.02$ performs best on validation set. Using this value of λ, we compare w_r and w_{ML}, and find out that $\text{MSE}_{w_r} < \text{MSE}_{w_{ML}}$. Clearly, on unseen validation dataset, ridge regression performs better than normal regression. Some reasons for why this could happen:

1. Since ridge regression is the Bayesian modeling of normal linear regression, it put constraints on the weight vector using priors. While training, this helps in ensuring that the gradients don't get affected by outliers or noise.

2. By priors on w in the form of λ, it leads to reduction in the estimator's variance, thereby resulting in the estimator's MSE on the validation set.

3. If there are irrelevant features in the training dataset, ridge regression will assign small, non-zero weights to remove the influence on the model's prediction capability.



8. Question 3

Solution: We are given a single sample y from a uni-variate Gaussian distribution with mean μ and variance $\sigma^2 = 1$. We know that,

$$
\begin{aligned}
L(y, \mu) &= P(y; \mu, \sigma) \\
L &= f_y(y; \mu, \sigma) \\
L &= \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(y-\mu)^2}{2\sigma^2}} \\
L &= \left(\frac{1}{\sqrt{2\pi}}\right) e^{\frac{-(y-\mu)^2}{2}} && (\because \sigma^2 = 1) \\
logL &= log\left(\frac{1}{\sqrt{2\pi}}\right) + log(e^{\frac{-(y-\mu)^2}{2}}) \\
logL &= log\left(\frac{1}{\sqrt{2\pi}}\right) - \frac{(y-\mu)^2}{2} \\
\frac{\partial logL}{\partial \mu} &= 2\hat{\mu}_{ML} - 2y = 0 \\
2\hat{\mu}_{ML} - 2y &= 0 \\
\hat{\mu}_{ML} &= y
\end{aligned}
$$

Now, let's find our new estimator by assuming Gaussian prior $\mathcal{N}(\mu_0, 1)$ on mean μ.

$$\begin{aligned}
L(y,\mu) &= P(y;\mu,\sigma)\cdot P(\mu)\\
L &= f_y(y;\mu,\sigma)\cdot f_\mu(\mu)\\
L &= \frac{1}{\sigma\sqrt{2\pi}}e^{\frac{-(y-\mu)^2}{2\sigma^2}}\cdot\frac{1}{\sigma_0\sqrt{2\pi}}e^{\frac{-(\mu-\mu_0)^2}{2\sigma_0^2}}\\
L &= \left(\frac{1}{2\pi}\right)e^{\frac{-(y-\mu)^2}{2}}e^{\frac{-(\mu-\mu_0)^2}{2}} && (\because \sigma^2=\sigma_0^2=1)\\
logL &= log\left(\frac{1}{2\pi}\right)+log(e^{\frac{-(y-\mu)^2-(\mu-\mu_0)^2}{2}})\\
logL &= log\left(\frac{1}{2\pi}\right)-\frac{(y-\mu)^2}{2}-\frac{(\mu-\mu_0)^2}{2}\\
\frac{\partial logL}{\partial\mu} &= 4\hat{\mu}_{new}-2(\mu_0+y)=0\\
4\hat{\mu}_{new}-2(\mu_0+y) &= 0\\
\hat{\mu}_{new} &= \frac{\mu_0+y}{2}\\
\hat{\mu}_{new} &= \frac{0+y}{2} && (\text{Let } \mu_0=0)\\
\hat{\mu}_{new} &= \frac{\hat{\mu}_{ML}}{2} && (\because \hat{\mu}_{ML}=y)
\end{aligned}$$

Now, let's calculate MSE for $\hat{\mu}_{ML}$ and $\hat{\mu}_{new}$. For $\hat{\mu}_{ML}$,

$$\begin{aligned}
\text{MSE}_{ML} &= \text{Bias}+\text{Variance}\\
&= (\mathbf{E}[\hat{\mu}_{ML}]-\mu)^2+\mathbf{Var}[\hat{\mu}_{ML}]\\
&= (\mathbf{E}[y]-\mu)^2+\mathbf{Var}[y]\\
&= (\mu-\mu)^2+1 && (\because \mathbf{E}[y]=\mu)\\
&= 1
\end{aligned}$$

For $\hat{\mu}_{new}$,

$$\begin{aligned}
\text{MSE}_{new} &= \text{Bias}+\text{Variance}\\
&= (\mathbf{E}[\hat{\mu}_{new}]-\mu)^2+\mathbf{Var}[\hat{\mu}_{new}]\\
&= \left(\frac{\mathbf{E}[\hat{\mu}_{ML}]}{2}-\mu\right)^2+\mathbf{Var}\left(\frac{\hat{\mu}_{ML}}{2}\right) && (\because \hat{\mu}_{new}=\frac{\hat{\mu}_{ML}}{2})\\
&= \left(\frac{\mathbf{E}[y]}{2}-\mu\right)^2+\mathbf{Var}\left(\frac{y}{2}\right)\\
&= \left(\frac{\mu}{2}-\mu\right)^2+\frac{1}{4} && (\because \mathbf{E}[y]=\mu \text{ and } \mathbf{Var}[y]=1)\\
&= \frac{\mu^2+1}{4}
\end{aligned}$$

For $\text{MSE}_{new} < \text{MSE}_{ML}$.

$$\begin{aligned} \frac{\mu^2+1}{4} &< 1 \\ \mu^2 + 1 &< 4 \\ \mu^2 &< 3 \end{aligned}$$

Hence, μ can take values as follows: $\mu < \sqrt{3}$ and $\mu > -\sqrt{3}$.