CS5691: Pattern Recognition and Machine Learning

Assignment 2

Course Instructor : Arun Rajkumar

Muhammed Tharikh
cs22m058

Q1) *You are given a data-set with 400 data points in {0, 1} 50 generated from a mixture of some distribution in the file A2Q1.csv. (Hint: Each datapoint is a flattened version of a* $\{0, 1\}^{10x5}$ *matrix.)*

i. 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.

The probabilistic mixture model would be multivariate bernoulli. The story would be, there will be 4 mixtures (and probability of choosing each will be π_j j=1,2,3,4). Then for each mixture we will have 50 parameters (basically we have 50 coins for each mixture). So the dataset has m=400 data with dimension 50, k=4 mixtures with choosing a particular mixture j with probability π_j, then for each mixture we have a 50 dimensional μ vector, each dimension is like a coin (has probability some probability p). Now for a dataset x with dimension d, each is governed by parameter μ so that

$$p(x/\mu_j) = \prod_{t=1}^{d} \mu_{jt}^{x_t}(1 - \mu_{jt})^{(1-x_t)}$$

here j can be 1,2,3,4 (this formula is the same as bernoulli, but we will try to find the probability along the dimension), it also implies that probability along each dimension is independent (i.e, the 50 coins we have are independent of each other).
Now on taking log-likelihood we get that

$$log(L(X/\mu, \Pi) = \sum_{i=1}^{m} log(\sum_{j=1}^{k} \pi_j\, p(x^i/\mu_j))$$

Here there exist a summation inside log term, so we can use jensen's inequality to solve and also we try to incorporate λ value too. So we get:

$$mod - log(L) = \sum_{i=1}^{m}\sum_{j=1}^{k} \lambda_j^i\, (\, log(\pi_j) + \sum_{t=1}^{d} (x_t^m log(\mu_{jt}) + (1 - x_t^m)log(1 - \mu_{jt})))$$

This modified likelihood provides us with a lower bound. And on maximizing this function with respect to λ, we get the **Expectation step**:

$$\lambda_j^i = \frac{\Pi_j\, p(x^i/\mu_j)}{\sum_{l=1}^{k} \Pi_l\, p(x^i/\mu_l)}$$

basically same expectation as that of GMM, except that the probability distribution is different.
Now for **maximization step**, we need to maximize μ and π
Here Π and μ are same as that of GMM, given as :

$$\pi_j = \frac{\sum_{i=1}^{m} \lambda_j^i}{m} \quad \mu_j = \frac{\sum_{i=1}^{m} \lambda_j^i x^i}{\sum_{i=1}^{m} \lambda_j^i}$$

So the algorithm would be :

```
1)Initialize μ,π, using K-means
2)until convergence do:
3)          Expectation step
4)          Maximization step
5)End
```

Convergence is given as either the previous parameter is same as current parameter (same in the sense they only differ by small amount) or previous log likeli is same as current. Either way, it converges in about 40 steps, so we can take 40 as total iteration and plot for 100 random initializations.
The log-likelihood for 1 randomization until convergence is given as:

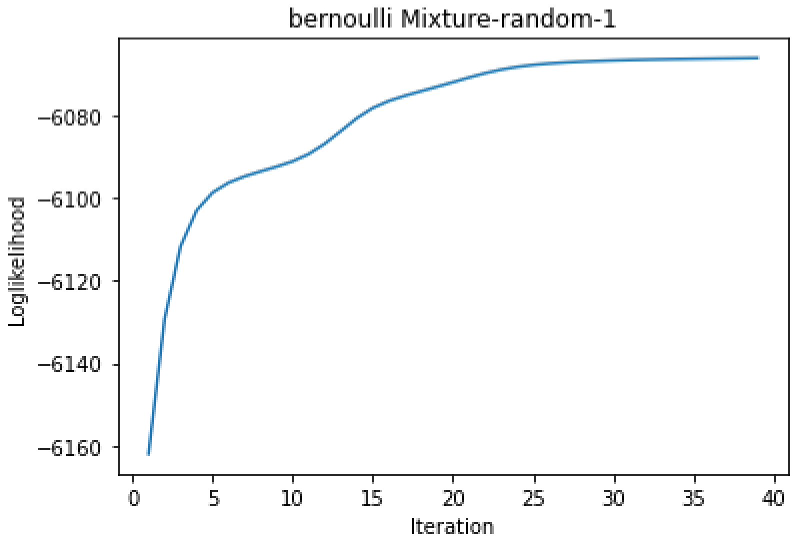

Fig 1. Iteration vs Log-likelihood of multivariate bernoulli

Now for 100 random initialization we get :

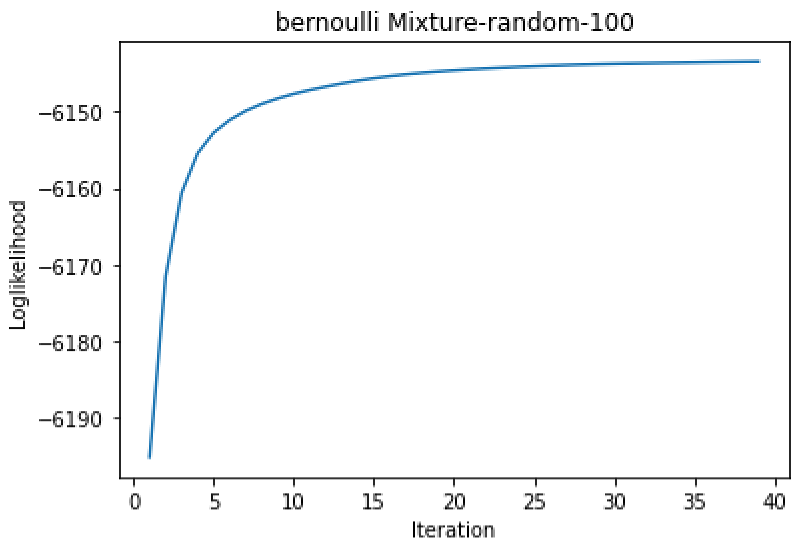

Fig 2. Iteration vs Log-likelihood of multivariate bernoulli for 100 random iterations

So the log likelihood keeps on increasing for each iteration until it reaches a point. Which has maximum likelihood. On doing 100 random initialization, we get a smoother curve, so this can remove any unwanted noise happening during taking random initialization.

ii. *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.*

It would not be a good model to incorporate into the dataset given, as the data is either 0 or 1, but if we use GMM, then even if variance is less, so that the data can be spread between [0,1], it would not tell a good generative story. As the data can be values which are not equal to 0 or 1, so say a pixel with value 0.5 can be generated by GMM, but the dataset will not have that data.
I initialized the parameters by k means, by taking means,covariances among the clusters, and fraction of datapoint in each cluster to get π.
In the implementation, as i am implementing gaussian function, there is a possibility that inverse doesn’t exit or determinant=0, so what i did is, just added some noise onto diagonals of the matrix to make it non-singualar.
Convergence is given as either the previous parameter is same as current parameter (same in the sense they only differ by small amount) or previous log likeli is same as current. Either way, it converges in about 40 steps, so we can take 40 as total iteration and plot for 100 random initializations.

GMM for 1 random initialization is :

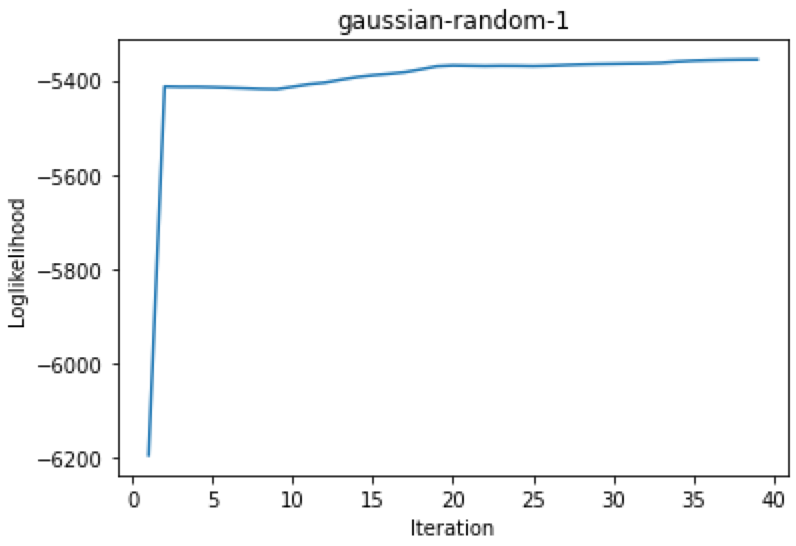

Fig 3. Iteration vs Log-likelihood of GMM 1 random iteration

Now for 100 random initialization, we get that :

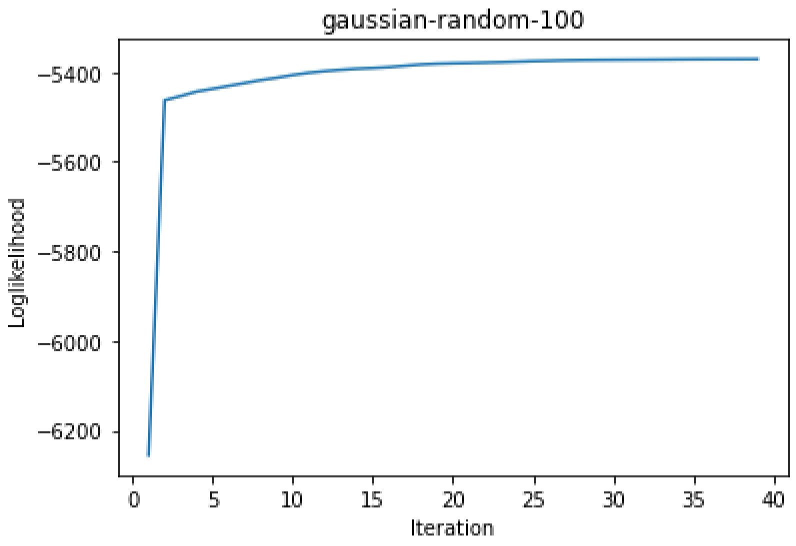

Fig 4. Iteration vs Log-likelihood of GMM for 100 random iterations

So the log likelihood keeps on increasing for each iteration until it reaches a point. Which has maximum likelihood. On doing 100 random initialization, we get a smoother curve, so this can remove any unwanted noise happening during taking random initialization.
When compared to previous, I am getting log-likely almost the same as the Bernoulli condition (but even higher). But I had several approximations like making it non-singular by adding some error terms and all and also GMM should not be the generative story, as it can generate several other values lying between 0 and 1. Also it took more time to run GMM compared to bernoulli.

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

After doing k-means on the dataset we got that after convergence, the objective/error, which is given as $\sum_{i=1}^{n} \left| x_i - \mu_{z_i} \right|_2^2$ is **1731.8145027244395.** Below shows the plot of iteration vs objective.

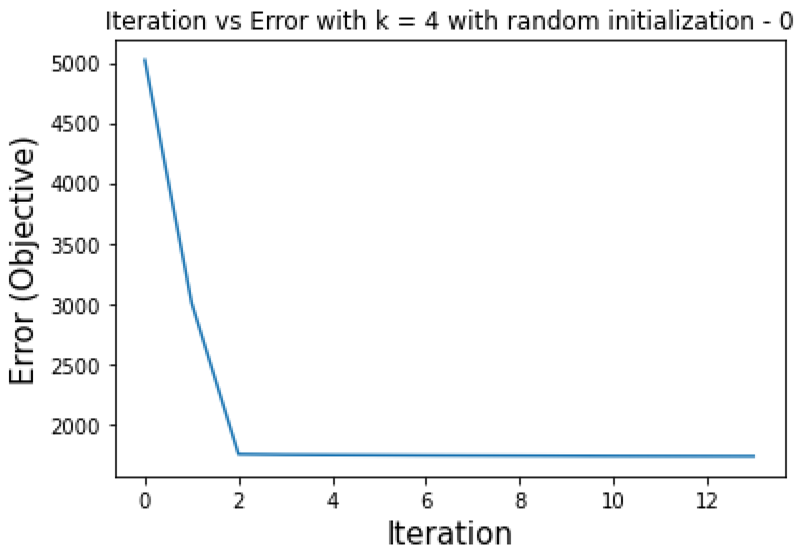

Fig 5. Iteration vs Objective of K-means

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

I would prefer multivariate Bernoulli, as it inherently has the capacity for generating the data which is either 0 or 1. GMM has the possibility of generating any values let alone 0 and 1. But from the log-likelihood graph obtained, he GMM seems a good fit for data, bu if use, then we can generate values which is not 0 nor 1.
Now for k-means, i converted GMM and multivariate bernoulli to hard clusters and tried to find the error of k means objective function and the values i got is :

GMM : **1759.2547141243558**
Multivariate Bernoulli: **1754.8773117234716**
K-means : **1755.3347781447715**

Here multivariate bernoulli has lower value compared to all others, but the value is not that much smaller than k means and GMM. So if we go by objective function, then multivariate bernoulli is better.

Now if we look at the dataset clearly, it is actually an image, which contains values 0 and 1 only, so 1 is white and 0 is black, so if we go by gaussian, it is not a good generative model as, it should only contain white or black pixel nothing more. Also k-means creates a voronoi type region and whereas the mixture models can have different boundaries and also is probabilistic in nature, whereas the k-means is deterministic. And also mixture models will be more robust.
So multivariate bernoulli is preferred over the other two.

Q2) *You are given a data-set in the file A2Q2Datatrain.csv with 10000 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).*
i. *Obtain the least squares solution w_{ML} to the regression problem using the analytical solution.*

After doing analytical solution, i got W_{ML}. Then tested with both the training and test data and got the error as : (here error is the objective function : $||X^T w - y||^2$)

Train Error: **396.86441863**

Test Error: **185.36365558**

ii) *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?*

After doing gradient descent on the dataset,there are several things observed:

- On every iteration the squared error decreases, i.e, the objective error function is decreasing on each iteration of gradient descent.
- $||w^t - w_{ML}||_2$ is also decreasing on each iteration, i.e, w in each iteration tends to become the true w that we got analytically.

I tried with many constant step sizes and when step sizes are large like 0.1,0.01, etc,.. The gradient explodes and goes to infinity. After some trial and error I found that a step size of 0.0000039 on a constant rate could solve the exploding gradient problem. Also i tried with varying step sizes, where on each iteration the step will step/t where t is the iteration. But the problem is that, if the initial step is low, then it doesn't converge easily (as I looked into the error value, the value is decreasing only at a smaller rate and step is still reducing). When I use a large initial step size and with each step it will be step/t, then, it explodes to infinity faster than reducing it to converge. So I used constant step size as mentioned above.
For convergence condition i have used 3 methods:

- Convergence by some iteration, say about 100,1000 etc. On doing so with T=1000 the train error was around **396.89624765** and the test error was around **185.0671856**. And below fig shows the iteration vs error and iteration vs $||w^t - w_{ML}||_2$

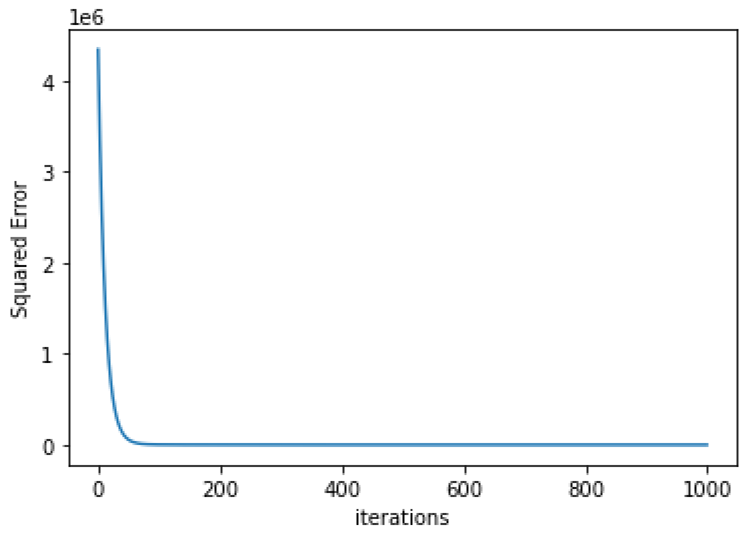

Fig 6.Iteration vs Error(Linear regression Objective) for 1000 iteration

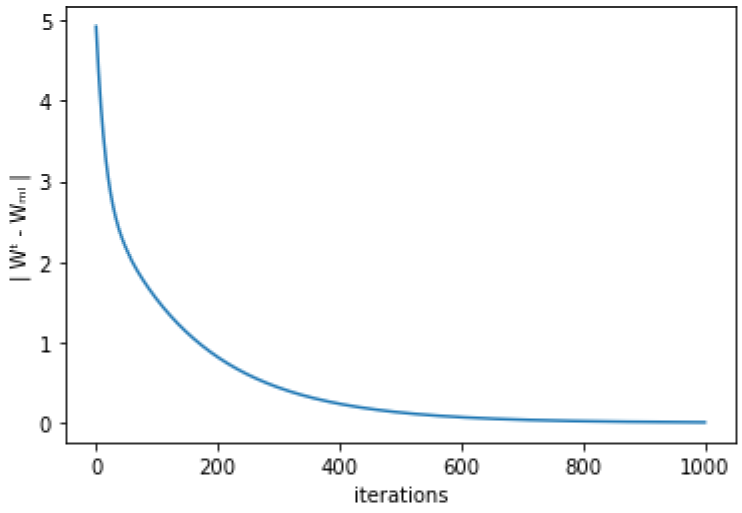

Fig 7. Iteration vs $\|w^t - w_{ML}\|_2$ for 1000 iteration

- Convergence by norm of $|w^t - w^{t-1}| < const,$ where the const is 0.000001, i.e, previous w is in less distance than some constant of current w. On doing so the train error was around **398.96358425** and the test error was around **183.06941274** close to analytical. And below fig shows the iteration vs error and iteration vs $\|w^t - w_{ML}\|_2$

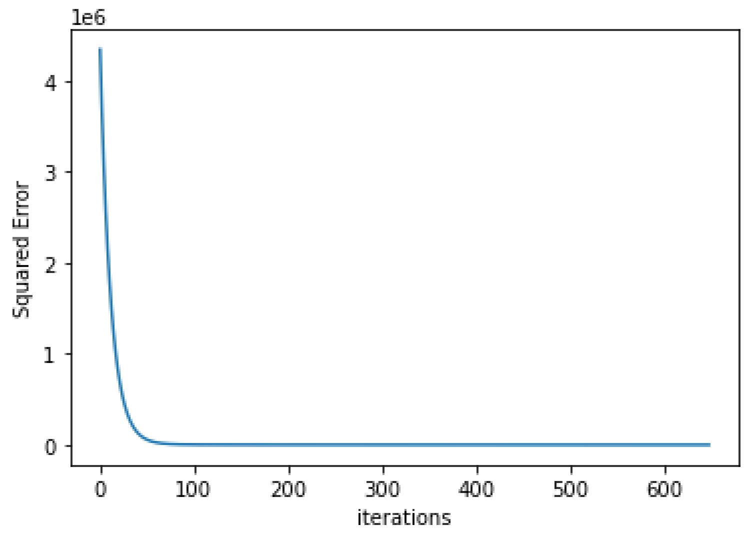

Fig 8. Iteration vs Error(Linear regression Objective) for $|w^t - w^{t-1}| < const$

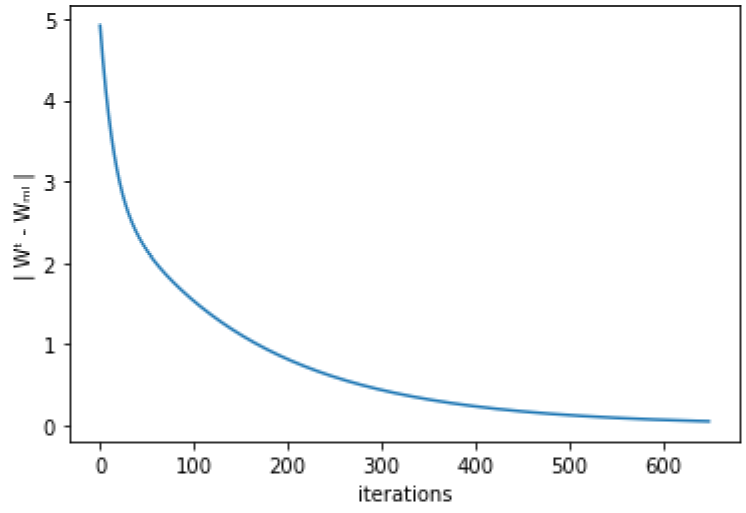

Fig 9. Iteration vs $||w^t - w_{ML}||_2$ for $|w^t - w^{t-1}| < const$

- Convergence by $|e^t - e^{t-1}| < const$ where the const is 0.00001 and e^t is error at tth iteration, so if previous and current error is same, then concluded that it is converged. On doing so the train error was around **396.865273** and the test error was around **185.31602672**. And below fig shows the iteration vs error and iteration vs $||w^t - w_{ML}||_2$.

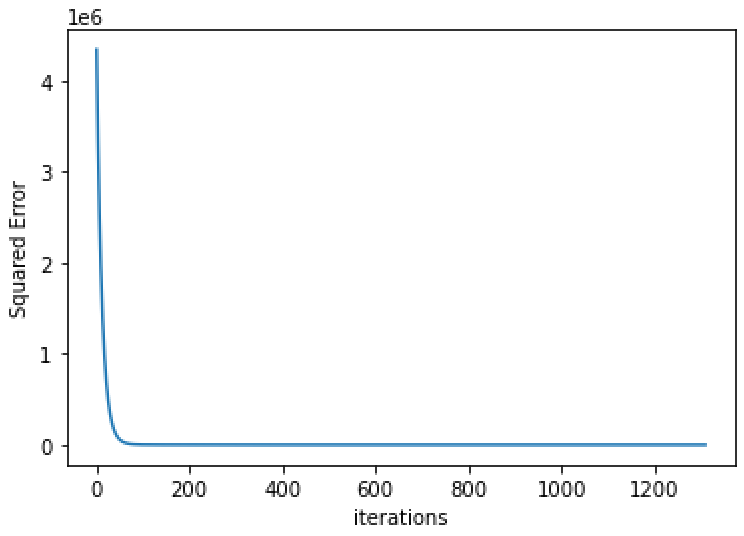

Fig 10. Iteration vs Error(Linear regression Objective) for $|e^t - e^{t-1}| < const$

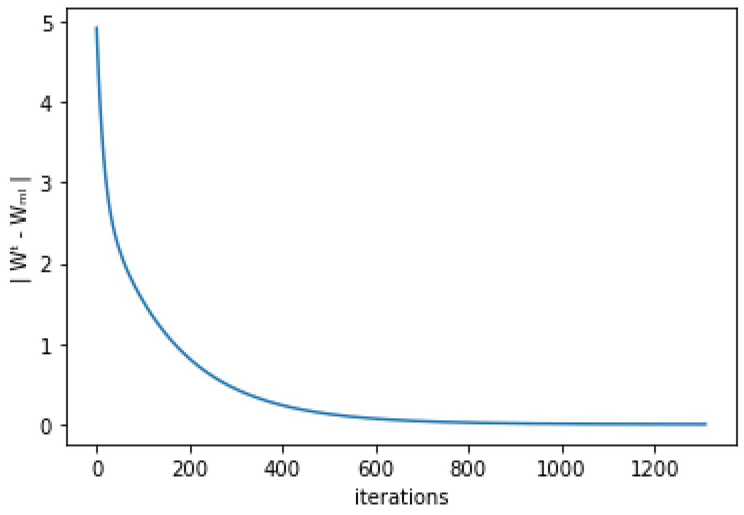

Fig 11. Iteration vs $||w^t - w_{ML}||_2$ for $|e^t - e^{t-1}| < const$

iii. *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?*

So after doing stochastic gradient descent with batch size=100, I observed the following plot - 2 plots are there , iteration vs error and iteration vs $||w^t - w_{ML}||_2$.

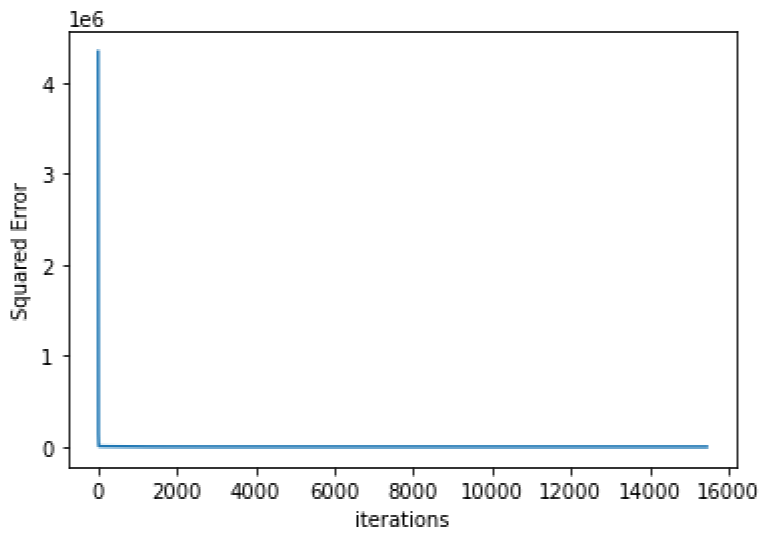

Fig 12. Iteration vs squared error until convergence

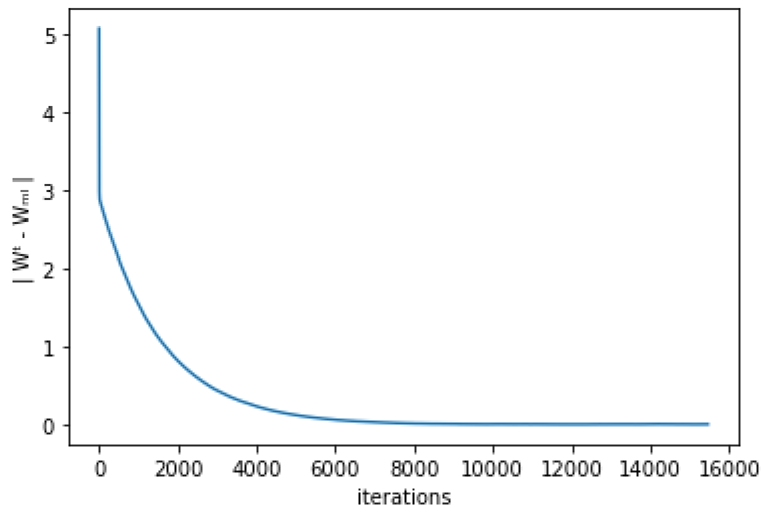

Fig 13. Iteration vs $||w^t - w_{ML}||_2$ until convergence

but this is not properly visible as the range of values are high for initial w's and after some iteration only it converges and it oscillates at last iterations, at start as the w is so far away from optimal, each step reduces errors, but on getting on some iteration, it starts to oscillate as the data are coming as batch and thus there is a possibility of degrading w. But atlast after some iteration, by using the convergence idea mentioned in part(i), it converges to a w. The below plot shows error and $||w^t - w_{ML}||_2$ at last iterations (i.,e from iteration number : 7000).

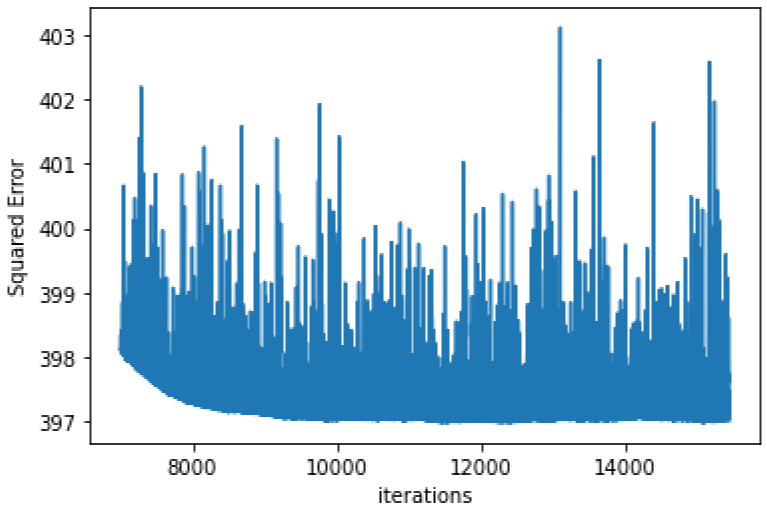

Fig 14. Iteration(from 7000 iteration) vs squared error until convergence

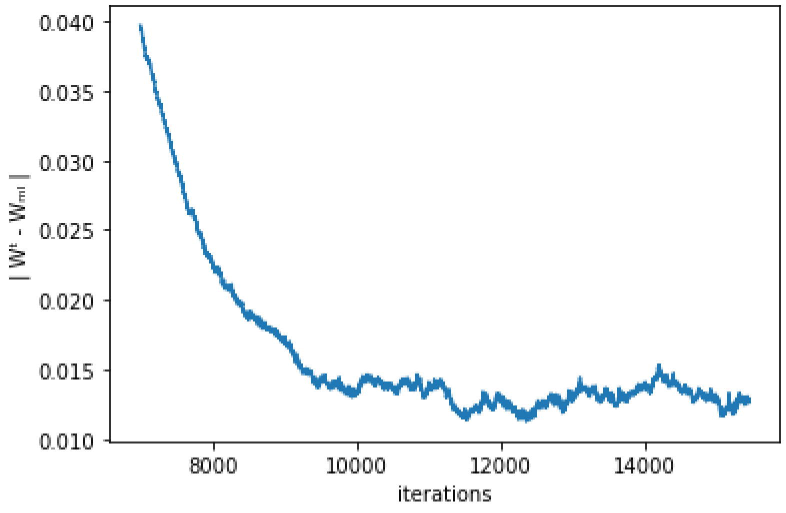

Fig 15. Iteration(from 7000 iteration) vs $\left\|w^{t} - w_{ML}\right\|_2$ until convergence

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 w_R. Compare the test error (for the test data in the file A2Q2Datatest.csv) of w_R with w w_{ML}. Which is better and why?*

Here I used k-fold cross validation, where each fold consists of 1000 data points, so 10 folds. I have plotted the lambdas with respect to its error, i tried with 0-1000 values of lambda, then i got only a dip near lambda = 1 to 4 point, and for higher lambdas, the error is only increasing, so tried it with 0 to 10 to get approximate value between 1.6 to 2 and is given in the below figure:

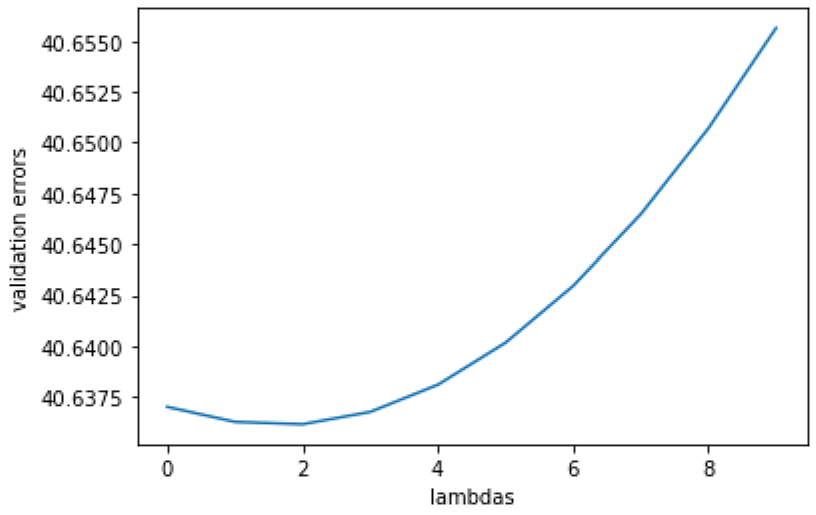

Fig 16. lambdas (0-10) vs validation errors

Then to refine lambda, i tried it with more minute steps, so i got the following figure:

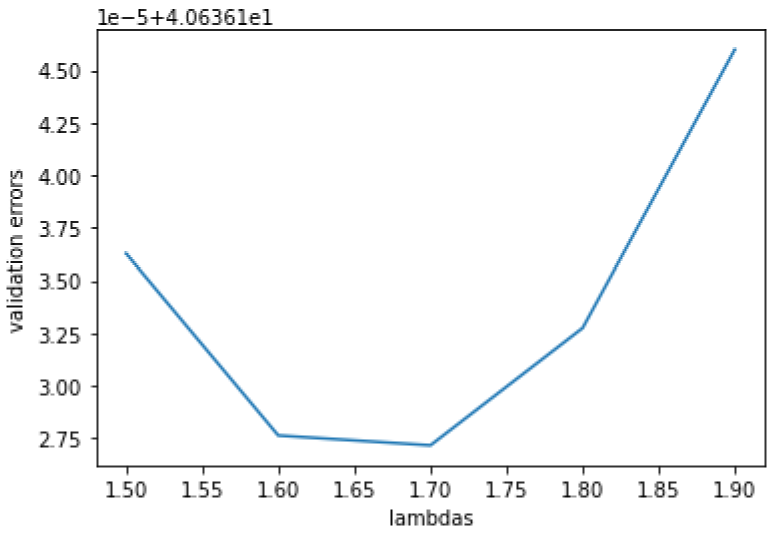

Fig 17. lambdas (1.5-1.9) vs validation errors

So for lambda=1.7, we have got less validation error as compared with other values of lambda including lambda=0. Even Though test samples error of both ridge and normal linear regression were almost same, validation error is very less for lambda=1.7. Also on looking at values of w, some of the values of w is less as compared to w_{ML} But train and test errors are approximately the same: test error : **396.87316721,** train error: **185.00107888.**

So either of them can be used for solving it, but from cross validation we can see that the perfect lambda would be 1.7, and also some value of w will be less. But based on test data, we can use either of them.