CS5691: Pattern Recognition and Machine Learning

Assignment # 2

Pratham Sunkad, ME21B145

April 14, 2024

Contents

1 Mixture Models **1**

1.1 Overview of the functions defined 1

1.1.1 Functions for Bernoulli Mixture Model: 1

1.1.2 Functions for Multivariate Gaussian Mixture Model: 1

1.1.3 Functions for Kmeans and comparisions of models: 1

1.2 Q1 subpart (i) 2

1.2.1 Determining which mixture generated this dataset: 2

1.2.2 Deriving the EM Algorithm for Bernoulli Mixture Model: 2

1.3 Q1 subpart (ii) 4

1.3.1 EM Algorithm Equations for Multivariate Gaussian Mixture Model: . 4

1.3.2 Experiment Details: 5

1.3.3 Comparision of BMM & GMM and Insights: 5

1.4 Q1 subpart (iii) 6

1.5 Q1 subpart (iv) 6

2 Regression **6**

2.1 Overview of the functions defined 6

2.2 Q2 subpart (i) 7

2.2.1 Checking whether bias is needed in the dataset 7

2.3 Q2 subpart (ii) 7

2.3.1 Gradient Descent 7

2.4 Q2 subpart (iii) 8

2.4.1 Stochastic Gradient Descent 8

2.5 Q2 subpart (iv) 9

2.5.1 Cross Validating λ for Ridge Regression 9

List of Figures

1 Deriving the EM algorithm for Bernoulli Mixture Model 3

2 BMM Log Likelihood averaged over 100 random initializations vs Iterations . 4

3 Multivariate GMM Log Likelihood averaged over 100 random initializations vs Iterations 5

4 Kmeans - Objective Function vs Iterations 6

5 Gradient Descent 8

6 Stochastic Gradient Descent 8

7 K fold cross validation for λ; Training error vs Iterations 9

8 λ vs Validation error 10

1 Mixture Models

1.1 Overview of the functions defined

1.1.1 Functions for Bernoulli Mixture Model:

(1) `def read_data` : Creates the dataset containing by the reading the CSV file and converting it to a numpy array. The shape of the dataset returned is $d \times n$, where $d = 50$ and $n = 400$.

(2) `def initialize_mixture_params` : It randomly initializes the probability array p_k of shape $d \times k$ and π_k. Where k is number of mixtures.

(3) `def e_step` : Performs the expectation step for Bernoulli Mixture Model by finding λ_k^i

(4) `def m_step` : Performs the maximization step for Bernoulli Mixture Model by finding the vector p_k and values of π_k.

(5) `def log_likelihood` : Finds the Bernoulli log likelihood of the dataset using p_k and π_k.

(6) `def em_algorithm` : Performs the EM Algorithm by iteratively performing `e_step` and `m_step`. Calculates Log Likelihood averaged over 100 random initializations.

1.1.2 Functions for Multivariate Gaussian Mixture Model:

(1) `def initialize_mixture_params_gaussian` : It randomly initializes the mean vectors μ_k, covariance matrices Σ_k and π_k.

(2) `def e_step_gaussian` : Performs the expectation step for Multivariate Gaussian Mixture Model by finding λ_k^i

(3) `def m_step_gaussian` : Performs the maximization step for Multivariate Gaussian Mixture Model by mean vectors μ_k, covariance matrices Σ_k and π_k.

(4) `def log_likelihood_gaussian` : Finds the Multivariate Gaussian log likelihood of the dataset using the mean vectors μ_k, covariance matrices Σ_k and π_k.

(5) `def em_algorithm_gaussian` : Performs the EM Algorithm by iteratively performing `e_step_gaussian` and `m_step_gaussian`. Calculates Log Likelihood averaged over 100 random initializations.

1.1.3 Functions for Kmeans and comparisions of models:

(1) `def cal_mean` : Calculates the mean of the cluster points and stores it in an array called `mean_array`. Length of this array is k and each element in `mean_array` corresponds to the mean of a cluster

(2) `def reassign` : Given the new mean calculated from `def cal_mean`, it reassigns new clusters to data-points according to Lloyd's algorithm.

(3) `def cal_error` : Calculates the objective function which we are trying to minimize and returns it's value

(4) `def random_initialize` : Initializes each data-point to a random cluster at the beginning of the algorithm

(5) `def kmeans` : Performs the K - Means clustering Algorithm and returns `error_array` containing value of the objective function and `iter_array` containing iterations.

(6) `def compare_kmeans_em` : Finds error using the K-Means objective function for BMM and GMM by doing hard clustering at the end by using λ_k^i values.

1.2 Q1 subpart (i)

1.2.1 Determining which mixture generated this dataset:

- All the data points consist of only 0's and 1's
- Thus this dataset might be generated from a **Bernoulli Mixture Model**
- The parameters of this distribution are i) p_k each being a $d \times 1$ vector for different values of k and ii) π_k
- We use Jensen's Inequality to derive the EM algorithm for this Mixture Model

1.2.2 Deriving the EM Algorithm for Bernoulli Mixture Model:

Let π_k represent the mixture probability of the kth mixture
Let p_k represent a vector ($d \times 1$) of probabilities, where each value in p_k indicates the probability of that feature being equal to 1

$$\text{Log } L(\theta) = \sum_{i=1}^{N} \log \left(\sum_{k=1}^{K} \pi_k \prod_{j=1}^{d} \left(\mathbf{p}_k^j\right)^{x_i^j} \left(1 - \mathbf{p}_k^j\right)^{1-x_i^j} \right), \tag{1}$$

where x_i^j indicates the jth component of the $\mathbf{x}_i$ datapoint, and $j = 1$ to d .

$$\text{Modified Log } L(\theta) = \sum_{i=1}^{N} \sum_{k=1}^{K} \lambda_k^i \left(\log\left(\frac{\pi_k}{\lambda_k^i}\right) + \sum_{j=1}^{d} \left(x_i^j \log(\mathbf{p}_k^j) + (1 - x_i^j) \log(1 - \mathbf{p}_k^j) \right) \right), \tag{2}$$

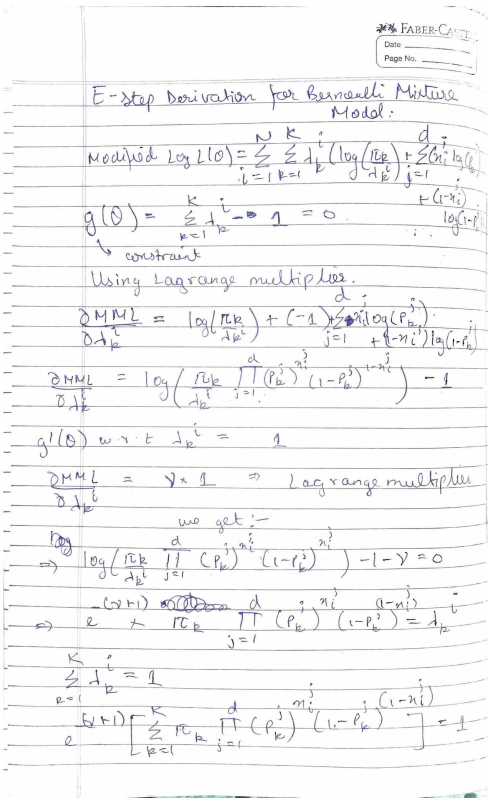
The Expectation step:

$$\lambda_k^i = \frac{\pi_k \prod_{j=1}^{d} (\mathbf{p}_k^j)^{x_i^j} (1 - \mathbf{p}_k^j)^{1-x_i^j}}{\sum_{k=1}^{K} \pi_k \prod_{j=1}^{d} (\mathbf{p}_k^j)^{x_i^j} (1 - \mathbf{p}_k^j)^{1-x_i^j}} \tag{3}$$

The Maximization step:

$$\mathbf{p}_k^j = \frac{\sum_{i=1}^{N} \lambda_k^i \cdot x_i^j}{\sum_{k=1}^{K} \lambda_k^i} \tag{4}$$

$$\pi_k = \frac{\sum_{i=1}^{N} \lambda_k^i}{N} \tag{5}$$



E-step Derivation for Bernoulli Mixture Model:

$$\text{Modified } \log L(\theta) = \sum_{i=1}^{N}\sum_{k=1}^{K} \lambda_k^i \left(\log\left(\frac{\pi_k}{\lambda_k^i}\right) + \sum_{j=1}^{d}\left(x_i^j \log(p_k^j) + (1-x_i^j)\log(1-p_k^j)\right)\right)$$

$$g(\theta) = \sum_{k=1}^{K} \lambda_k^i - 1 = 0$$

↳ constraint

Using Lagrange multiplier.

$$\frac{\partial MML}{\partial \lambda_k^i} = \log\left(\frac{\pi_k}{\lambda_k^i}\right) + (-1) + \sum_{j=1}^{d} x_i^j \log(p_k^j) + (1-x_i^j)\log(1-p_k^j)$$

$$\frac{\partial MML}{\partial \lambda_k^i} = \log\left(\frac{\pi_k}{\lambda_k^i}\prod_{j=1}^{d}(p_k^j)^{x_i^j}(1-p_k^j)^{1-x_i^j}\right) - 1$$

$g'(\theta)$ w.r.t $\lambda_k^i = 1$

$$\frac{\partial MML}{\partial \lambda_k^i} = \nu \times 1 \Rightarrow \text{Lagrange multiplier}$$

we get :-

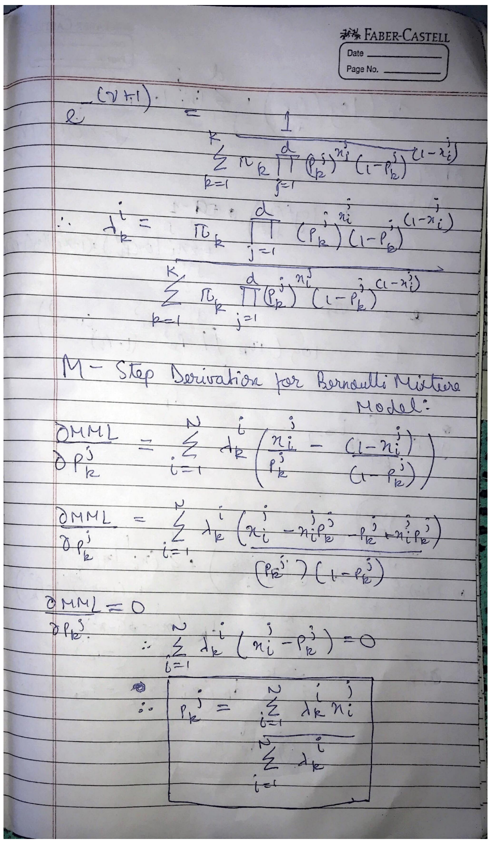
$$\Rightarrow \log\left(\frac{\pi_k}{\lambda_k^i}\prod_{j=1}^{d}(p_k^j)^{x_i^j}(1-p_k^j)^{x_i^j}\right) - 1 - \nu = 0$$

$$\Rightarrow e^{-(\nu+1)} \times \pi_k \prod_{j=1}^{d}(p_k^j)^{x_i^j}(1-p_k^j)^{(1-x_i^j)} = \lambda_k^i$$

$$\sum_{k=1}^{K}\lambda_k^i = 1$$

$$e^{-(\nu+1)}\left[\sum_{k=1}^{K}\pi_k\prod_{j=1}^{d}(p_k^j)^{x_i^j}(1-p_k^j)^{(1-x_i^j)}\right] = 1$$

(a) Page 1 of Derivation



$$e^{-(\nu+1)} = \frac{1}{\sum_{k=1}^{K}\pi_k\prod_{j=1}^{d}(p_k^j)^{x_i^j}(1-p_k^j)^{(1-x_i^j)}}$$

$$\therefore \lambda_k^i = \frac{\pi_k\prod_{j=1}^{d}(p_k^j)^{x_i^j}(1-p_k^j)^{(1-x_i^j)}}{\sum_{k=1}^{K}\pi_k\prod_{j=1}^{d}(p_k^j)^{x_i^j}(1-p_k^j)^{(1-x_i^j)}}$$

M - Step Derivation for Bernoulli Mixture Model:

$$\frac{\partial MML}{\partial p_k^j} = \sum_{i=1}^{N}\lambda_k^i\left(\frac{x_i^j}{p_k^j} - \frac{(1-x_i^j)}{(1-p_k^j)}\right)$$

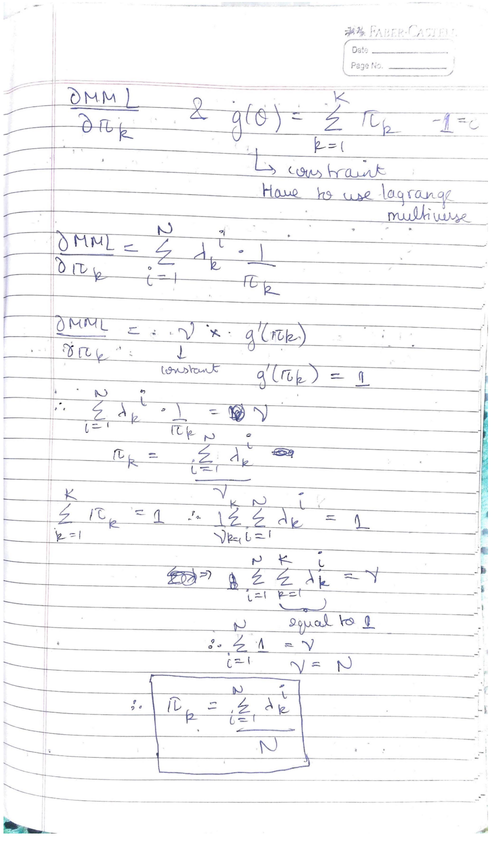
$$\frac{\partial MML}{\partial p_k^j} = \sum_{i=1}^{N}\lambda_k^i\frac{\left(x_i^j - x_i^j p_k^j - p_k^j + x_i^j p_k^j\right)}{(p_k^j)(1-p_k^j)}$$

$$\frac{\partial MML}{\partial p_k^j} = 0$$

$$\therefore \sum_{i=1}^{N}\lambda_k^i(x_i^j - p_k^j) = 0$$

$$\therefore \boxed{p_k^j = \frac{\sum_{i=1}^{N}\lambda_k^i x_i^j}{\sum_{i=1}^{N}\lambda_k^i}}$$

(b) Page 2 of Derivation



$$\frac{\partial MML}{\partial \pi_k} \quad \& \quad g(\theta) = \sum_{k=1}^{K}\pi_k - 1 = 0$$

↳ constraint

Have to use lagrange multiuse

$$\frac{\partial MML}{\partial \pi_k} = \sum_{i=1}^{N}\lambda_k^i \cdot \frac{1}{\pi_k}$$

$$\frac{\partial MML}{\partial \pi_k} = \underset{\text{constant}}{\nu} \times g'(\pi_k) \qquad g'(\pi_k) = 1$$

$$\therefore \sum_{i=1}^{N}\lambda_k^i \cdot \frac{1}{\pi_k} = \nu$$

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

$$\sum_{k=1}^{K}\pi_k = 1 \quad \therefore \frac{1}{\nu}\sum_{k=1}^{K}\sum_{i=1}^{N}\lambda_k^i = 1$$

$$\Rightarrow \sum_{i=1}^{N}\underbrace{\sum_{k=1}^{K}\lambda_k^i}_{\text{equal to } 1} = \nu$$

$$\therefore \sum_{i=1}^{N} 1 = \nu \qquad \nu = N$$

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

(c) Page 3 of Derivation

Figure 1: Deriving the EM algorithm for Bernoulli Mixture Model

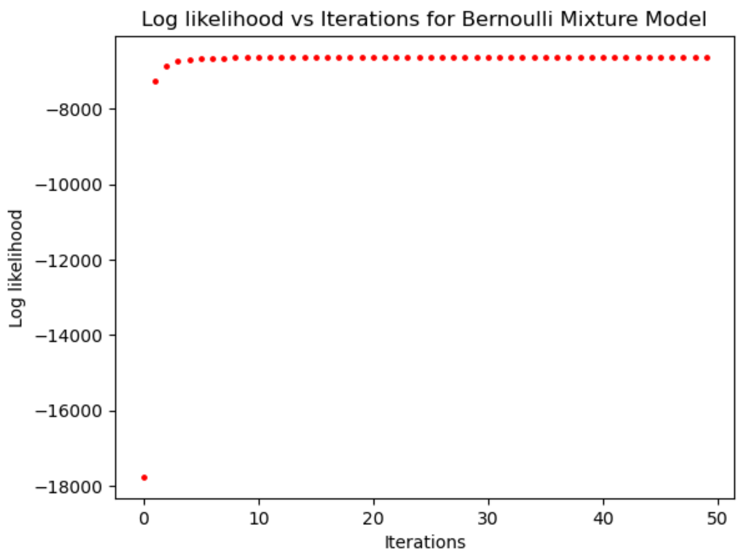


Figure 2: BMM Log Likelihood averaged over 100 random initializations vs Iterations

The EM algorithm is run for 50 iterations for 100 random initializations. The Log Likelihood values converges to -6,633.

1.3 Q1 subpart (ii)

1.3.1 EM Algorithm Equations for Multivariate Gaussian Mixture Model:

$$\text{Log } L(\theta) = \sum_{i=1}^{N} \log \left(\sum_{k=1}^{K} \pi_k \frac{1}{(2\pi)^{d/2} |\mathbf{\Sigma}_k|^{0.5}} e^{-\frac{1}{2}(\mathbf{x}_i - \mathbf{u}_k)^T \mathbf{\Sigma}_k^{-1} (\mathbf{x}_i - \mathbf{u}_k)} \right) \tag{6}$$

where $\mathbf{x}_i$ indicates the ith datapoint, $\mathbf{u}_k$ is the mean, and $\mathbf{\Sigma}_k$ is the covariance of the kth mixture.

$$\text{Modified Log}L(\theta) = \sum_{i=1}^{N} \sum_{k=1}^{K} \lambda_k^i \left(\log \left(\frac{\pi_k}{\lambda_k^i} \right) + \log \left(\frac{1}{(2\pi)^{d/2} |\mathbf{\Sigma}_k|^{0.5}} \right) - \frac{1}{2} (\mathbf{x}_i - \mathbf{u}_k)^T \mathbf{\Sigma}_k^{-1} (\mathbf{x}_i - \mathbf{u}_k) \right) \tag{7}$$

The Expectation Step:

$$\lambda_k^i = \frac{\pi_k \frac{1}{(2\pi)^{d/2} |\mathbf{\Sigma}_k|^{0.5}} e^{-\frac{1}{2}(\mathbf{x}_i - \mathbf{u}_k)^T \mathbf{\Sigma}_k^{-1} (\mathbf{x}_i - \mathbf{u}_k)}}{\sum_{k=1}^{K} \pi_k \frac{1}{(2\pi)^{d/2} |\mathbf{\Sigma}_k|^{0.5}} e^{-\frac{1}{2}(\mathbf{x}_i - \mathbf{u}_k)^T \mathbf{\Sigma}_k^{-1} (\mathbf{x}_i - \mathbf{u}_k)}} \tag{8}$$

The Maximization Step:

$$\boldsymbol{u}_k = \frac{\sum_{i=1}^{n} \lambda_k^i \cdot x_i}{\sum_{i=1}^{n} \lambda_k^i} \tag{9}$$

$$\mathbf{\Sigma}_k = \frac{\sum_{i=1}^{n} \lambda_k^i (\mathbf{x}_i - \mathbf{u}_k)(\mathbf{x}_i - \mathbf{u}_k)^T}{\sum_{i=1}^{n} \lambda_k^i} \tag{10}$$

$$\pi_k = \frac{\sum_{i=1}^{N} \lambda_k^i}{N} \tag{11}$$

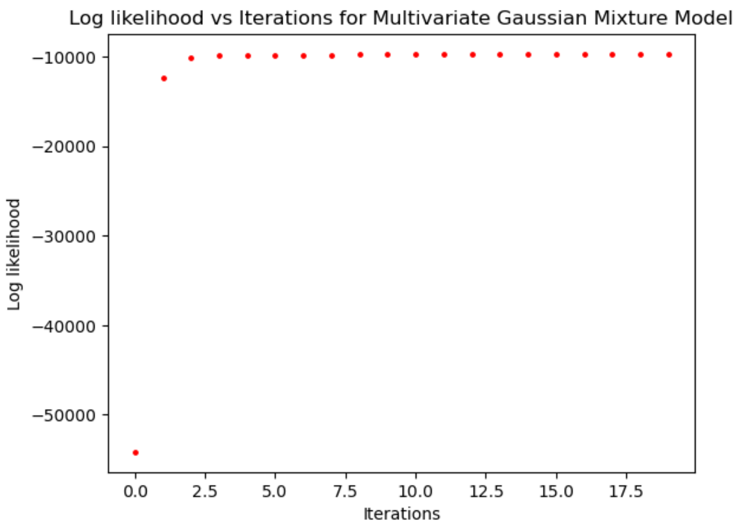


Figure 3: Multivariate GMM Log Likelihood averaged over 100 random initializations vs Iterations

The EM algorithm is run for 20 iterations for 100 random initializations. The Log Likelihood values converges to -9722.

1.3.2 Experiment Details:

- During execution of the code, $|\mathbf{\Sigma}_k|$ was going close to 0, thus giving divide by zero errors. To counter this I updated Σ_k as:
  $\Sigma_k = \Sigma_k + \alpha I$
  The value of α is chosen to be a small number, in my case $\alpha = 0.04$. Such that $|\mathbf{\Sigma}_k|$ became non zero
- Choosing a large value of α is not encouraged as that will significantly change the covariance matrix.

1.3.3 Comparision of BMM & GMM and Insights:

- The Bernoulli Mixture Model converges to a much larger value of Log Likelihood : -6633 whereas The Multivariate Gaussian Mixture Model converges to a much smaller value Log Likelihood : - 9927
- BMM converges faster to the local maxima where as the GMM takes more iterations to be converge to the local maxima
- Both plots have a sudden jump after the first iteration indicating movement to their local maxima
- The GMM has a covariance matrix Σ_k which takes into account the relation between all the features
- The Gaussian Mixture decently approximates this dataset even though the dataset is binary in nature. It maybe even better than Bernoulli Mixture.

1.4 Q1 subpart (iii)

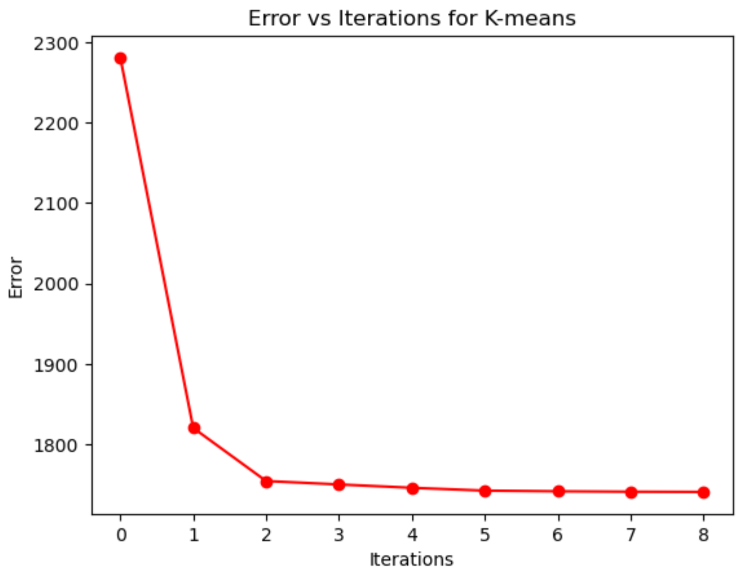


Figure 4: Kmeans - Objective Function vs Iterations

The Kmeans Objective function converges to a value of 1740.

1.5 Q1 subpart (iv)

Which algorithm out of **BMM**, **GMM** or **K-Means** will you choose for this dataset and why?
Ans : **GMM better than K-Means better than BMM**

- The metric used to judge them is the K-Means objective function. For GMM and BMM hard clustering is done using the values of λ_k for all datapoints.
  Then the objective function is found out.
- Value of objective function for GMM = 1734.05, K-Means = 1745.56 ,BMM = 1763.43
- This shows that the Gaussian Mixture Model still approximates this data well if the K-means objective function is the criteria to judge.

2 Regression

2.1 Overview of the functions defined

(1) `def read_data` : Creates the dataset containing by the reading the CSV file and converting features and labels to a numpy array. The shape of the features returned is $d \times n$, where $d = 100$ and $n = 10000$. The shape of labels returned is $d \times 1$

(2) `def read_data_bias` : The dataset need not be centered. Thus function creates the dataset by accounting for an additional feature for the bias. The shape of the dataset returned is $d + 1 \times n$, where $d = 100$ and $n = 10000$. The shape of labels returned is $d \times 1$

(3) `def linear_reg` : Calculates w_{ML} for the dataset using the analytical solution

(4) `def cal_error` : Calculates the least squares error : $\sum_{i=1}^{N}(w^T x_i - y_i)^2$ for a w and returns it

(5) `def grad_desc` : Initializes w and performs the gradient descent algorithm for w_{ML} with a constant learning rate for a certain number of iterations.

(6) `def stoch_grad` : Initializes w and performs the stochastic gradient descent algorithm for w_{ML} with a constant learning rate and a given batch size for a certain number of iterations.

(7) `def ridge_reg` : Initializes w and performs the gradient descent algorithm for ridge regression with a constant learning rate and for a certain number of iterations.

(8) `def ridge_analytical` : Calculates w_R for the dataset using the analytical solution.

(9) `def kfold_val` : Splits training data, performs K - fold cross validation and finds the average Least Square error for given λ value in the case of ridge regression.

2.2 Q2 subpart (i)

2.2.1 Checking whether bias is needed in the dataset

- To check whether this data needs a bias term as an extra feature I performed Linear regression with and without the bias term. I compared the least squared error obtained.
- With the bias term, the least squared error obtained : 396.85210868791546.
  Without the bias term, the least squared error obtained : 396.86441862725104
- Error difference between model with excluding the bias and model with including the bias : 0.012309939335580111
- Since this value is negligible we can assume that the dataset does not require a bias term in Linear regression.

Least Squared Error of w_{ML} ; 396.86441862725104

.

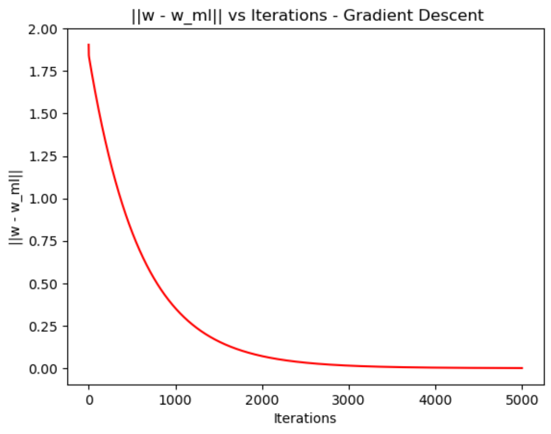
2.3 Q2 subpart (ii)

2.3.1 Gradient Descent

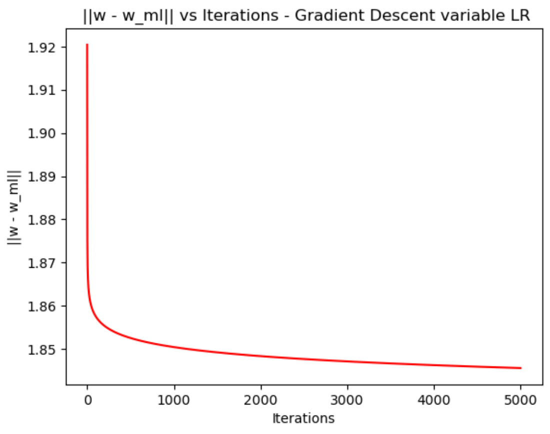
Observations:

- The learning rate being used is very important. If it is too high the model diverges and weights go to inf .
  If it is too low the model takes a long time to converge.
- I also tried with learning rate = $1/t$ which satisfies the property of diverging sum and converging sum of squares.
  But the model converged at a point with high least squared error.
- While using a constant learning rate, there would be a critical learning rate for which the model will learn the fastest but would diverge as soon as the learning rate is beyond that threshold.
- After experimenting a constant learning rate of 1e-6 worked the best. The model converged to w_{ML}. Least Squared Error = 396.8648650645291

- w was initialized using a Normal Distribution with mean $= 0$ and deviation $= 0.1$



(a) Gradient Descent to solve the least squares algorithm.



(b) Gradient Descent to solve the least squares algorithm with variable LR

Figure 5: Gradient Descent

2.4 Q2 subpart (iii)

2.4.1 Stochastic Gradient Descent

Observations:

- The Stochastic Gradient descent is more computationally feasible than standard gradient descent as a batch of derivatives is being used instead of the whole dataset.
- This is the reason why it takes lesser time for Stochastic Gradient Descent to run as compared to gradient descent for the same number of iterations.
- Gradient Descent converges faster than Stochastic Gradient descent. The final least squared error of GD is lesser than SGD. Error SGD : 397.19139844141415
- A larger learning rate of 1e-4 is used in this case. This learning rate is constant. Using the variable learning rate was giving the same issue as in the case of Gradient Descent.

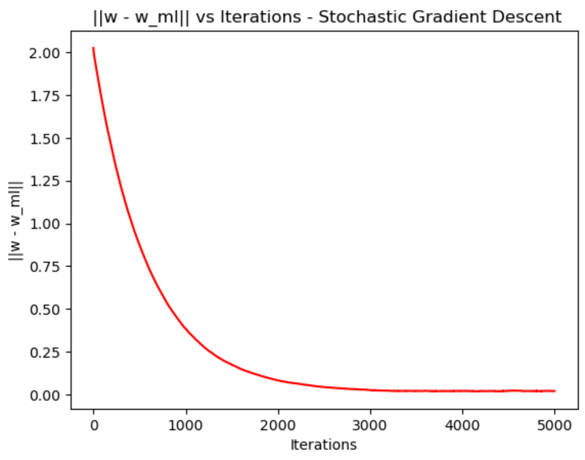


Figure 6: Stochastic Gradient Descent

2.5 Q2 subpart (iv)

2.5.1 Cross Validating λ for Ridge Regression

- Doing k - fold cross validation for λ with $k = 5$.To check which value of lambda is the best, we have to run till an iteration such that non of the models have converged and see which has the lowest l2 error.
- Running the Gradient Descent Algorithm for 1000 iterations for every combination of train and validation set. Finding the least square error and averaging it across all validation sets.
- λ values = [5000, 1000, 500, 250, 100, 50, 10, 5, 2, 1, 0.5 , 0.1, 0.05, 0.01, 0.005, 0.001]

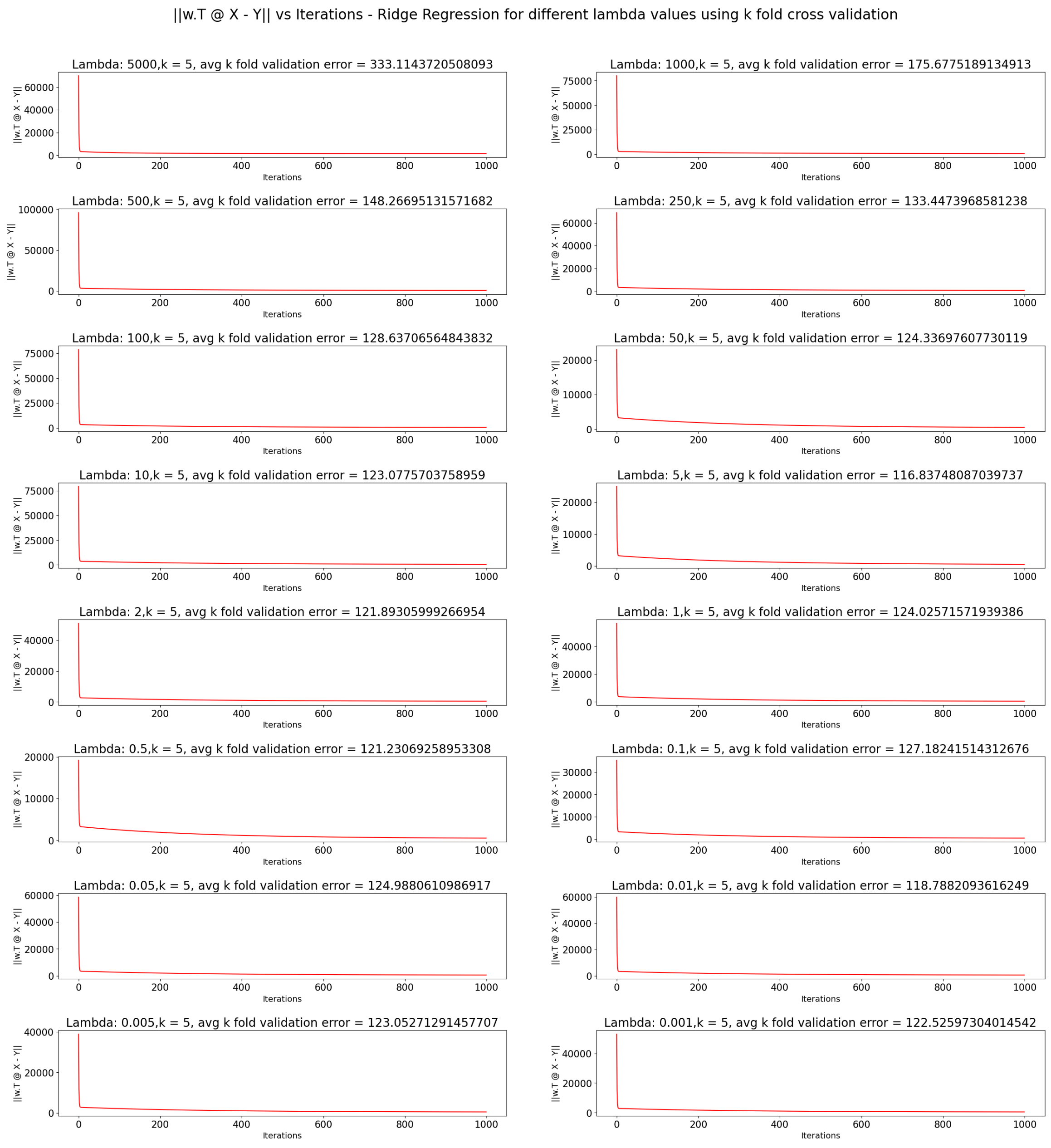


Figure 7: K fold cross validation for λ; Training error vs Iterations

Observations:

- Figure 7 Plots the training error $\sum_{i=1}^{N}(w^T x_i - y_i)^2$ against iterations. The corresponding title of the figure contains the average validation error obtained from k - fold cross validation.
- For $\lambda = 5$ the K-fold cross validation error is the least. This means that the best λ lies in the range of 5.
- Seeing the decreasing trend of error after $\lambda = 2$ and increasing trend at $\lambda = 10$, we can predict that $2 < \lambda < 10$
- Test Error for $w_{ML} = 185.3636555848964$
  Test erro for $w_R = 184.31339018614096$
- w_R has lesser test error and is better than w_{ML}.
  This is because ridge regression tries to bring features with lesser importance close to 0. This reduces the variance of the model and leads to better performance on unseen test data.

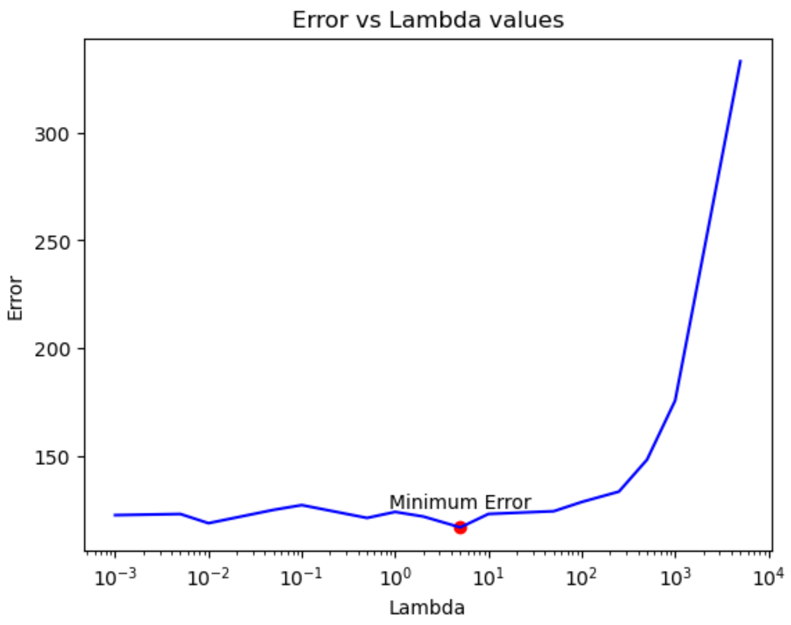


Figure 8: λ vs Validation error

******