CS5691 - Pattern Recognition in Machine Learning

Programming Assignemnt 2

lab Report

Name - Jay Jitendra Prajapati

Roll number - ME21B143

Contents

1 EM ALGORITHM

- The data has been read with the help of panda library and stored in data variable in such a way that each row corresponds to a datapoint with $d = 50$ features and there are total 400 rows.

1.1 EM Algorithm for Bernoulli Mixture

- The probabilistic mixture that could have generated this dataset is **Bernoulli mixture**. The reason is because every feature of data is binary i.e either 0 or 1 which indicates that bernoulli distribution could have generated the dataset. This model is also called **Latent Class Analysis**.
- Consider a datapoint x with d features.

$$P(x|\mu) = \prod_{i=1}^{d} \mu_i^{x_i}(1-\mu_i)^{(1-x_i)}$$

where $x = (x_1, \ldots, x_d)^T$ and $\mu = (\mu_1, \ldots, \mu_d)^T$

- Now consider a finite mixture of these distributions given by

$$P(x|\mu, \pi) = \sum_{k=1}^{K} \pi_k P(x|\mu_k)$$

where $\mu = (\mu_1, \ldots, \mu_K)$ and $\pi = (\pi_1, \ldots, \pi_K)$, and

$$P(x|\mu_k) = \prod_{i=1}^{d} \mu_{ki}^{x_i}(1-\mu_{ki})^{(1-x_i)}$$

- If we are given a data set $X = \{x_1, \ldots, x_n\}$ then the log likelihood function for this model is given by

$$P(X|\mu, \pi) = \prod_{i=1}^{n} \{\sum_{k=1}^{K} \pi_k P(x_i, \mu_k)\}$$

- Taking natural log of above function as it is increasing monotonically,

$$\ln P(X|\mu,\pi) = \sum_{i=1}^{n} \ln\{\sum_{k=1}^{K} \pi_k P(x_i,\mu_k)\}$$

- Again we see the appearance of the summation inside the logarithm, so that the maximum likelihood solution no longer has closed form.

- We now derive the EM algorithm for maximizing the likelihood function for the mixture of Bernoulli distributions. To do this, we first introduce an explicit latent variable z associated with each instance of x. As in the case of the Gaussian mixture, $z = (z_1,\ldots,z_K)^T$ is a binary K-dimensional variable having a single component equal to 1, with all other components equal to 0.

$$P(x|z,\mu) = \prod_{k=1}^{K} P(x|\mu_k)^{z_k}$$

- while the prior distribution for the latent variables is the same as for the mixture of Gaussians model, so that

$$P(\mu|\pi) = \prod_{k=1}^{K} \pi_k^{z_k}$$

- In order to derive the EM algorithm, we first write down the complete-data log likelihood function, which is given by

$$\ln P(X,Z|\mu,\pi) = \sum_{i=1}^{n}\sum_{k=1}^{K} z_{ik}\{\ln \pi_k +$$

$$\sum_{j=1}^{d}[x_{ij}\ln \mu_{kj} + (1-x_{ij})\ln(1-\mu_{kj})]\}$$

- z_{ik} is the posterior probability, or responsibility, of component k given data point x_i. In the Expectation step, these responsibilities are evaluated using Bayes' theorem and keeping bernoulli parameters constant, which takes the form

$$z_{ik} = \frac{\pi_k P(x_i|\mu_k)}{\sum_{j=1}^{K} \pi_j P(x_i|\mu_j)}$$

- Bernoulli parameters are evaluated by keeping z constant and are given by

$$\mu_k = \frac{\sum_{i=1}^{n} z_{ik} x_i}{\sum_{i=1}^{n} z_{ik}}$$

$$\pi_k = \frac{\sum_{i=1}^{n} z_{ik}}{n}$$

- Convergence criteria is chosen as

$$\|(\theta^{t+1} - \theta^t)/\theta^{t+1}\| < \epsilon$$

  where θ is the log-likelihood value and ϵ is the tolerance and taken as 10^{-2}

- Plot the log-likelihood (averaged over 100 random initializations) as a function of iterations.

$$\ln P(X|\mu, \pi) = \sum_{i=1}^{n} \ln\{\sum_{k=1}^{K} \pi_k P(x_i, \mu_k)\}$$

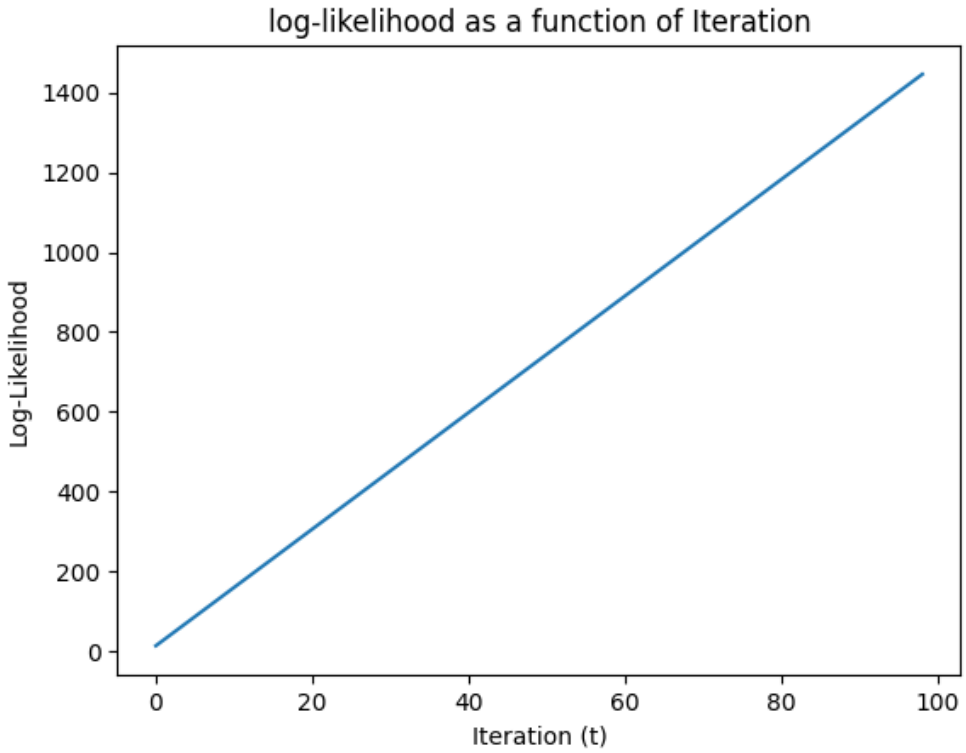

Figure 1:

1.2 EM Algorithm for Gaussian Mixture

- The data has now been generated from a mixture of Gaussians with 4 mixtures. The Gaussian mixture distribution can be written as a linear superposition of Gaussians in the form

$$P(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k)$$

where

$$\mathcal{N}(x|\mu_k, \Sigma_k) = \frac{1}{(2\pi)^{d/2}} \frac{1}{|\Sigma_k|^{1/2}} exp\{-\frac{1}{2}(x-\mu_k)^T \Sigma_k^{-1}(x-\mu_k)\}$$

- Let $\{\pi_k\}$ be mixture coefficient with given constraints and

write the distribution as follows:

$$0 \leq \pi_k \leq 1 \quad , \sum_{k=1}^{K} \pi_k = 1$$

$$P(z) = \prod_{k=1}^{K} \pi_k^{z_k}$$

- We have a data set of observations $\{x_1, \ldots, x_n\}$, and we wish to model this data using a mixture of Gaussians. We can represent this data as an $d \times n$ matrix X. The parameters for this distribution are μ, Σ, π, mean, covariance and mixture coefficient respectively.

- The likelihood for this distribution can be stated as follows:

$$P(X|\mu, \Sigma, \pi) = \prod_{i=1}^{n} \{\sum_{k=1}^{K} \pi_k \mathcal{N}(x_i|\mu_k, \Sigma_k)\}$$

- Since the function is montonically increasing will take natural log on both sides giving log-likelihood.

$$\ln P(X|\mu, \Sigma, \pi) = \sum_{i=1}^{n} \ln\{\sum_{k=1}^{K} \pi_k \mathcal{N}(x_i|\mu_k, \Sigma_k)\}$$

- **Expectation step:** Fix the parameters and evaluate λ.

$$\lambda_{ik} = \frac{\pi_k P(x_i|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j P(x_i|\mu_j, \Sigma_j)}$$

- **Maximization step:** Fix λ and evaluate the parameters from given expression:

$$\mu_k = \frac{\sum_{i=1}^{n} \lambda_{ik} x_i}{\sum_{i=1}^{n} \lambda_{ik}}$$

$$\Sigma_k = \frac{\sum_{i=1}^{n} \lambda_{ik}(x_i - \mu_k)(x_i - \mu_k)^T}{\sum_{i=1}^{n} \lambda_{ik}}$$

$$\pi_k = \frac{\sum_{i=1}^{n} \lambda_{ik}}{n}$$

- Convergence criteria is chosen as

$$\|(\theta^{t+1} - \theta^t)/\theta^{t+1}\| < \epsilon$$

  where θ is the log-likelihood value and ϵ is the tolerance and taken as 10^{-1}

- Plot the log-likelihood function (averaged over 100 random initializations) as a function of iterations.

$$\ln P(X|\mu, \Sigma, \pi) = \sum_{i=1}^{n} \ln\{\sum_{k=1}^{K} \pi_k \mathcal{N}(x_i|\mu_k, \Sigma_k)\}$$

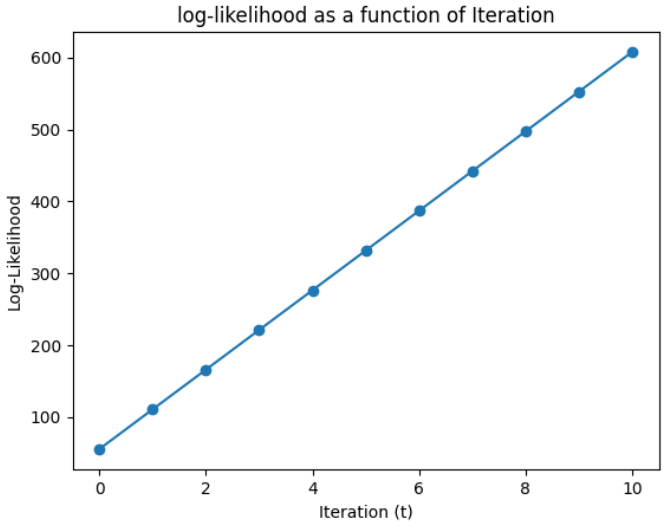

Figure 2:

- On plotting both log-likelihood values on same graph we get:

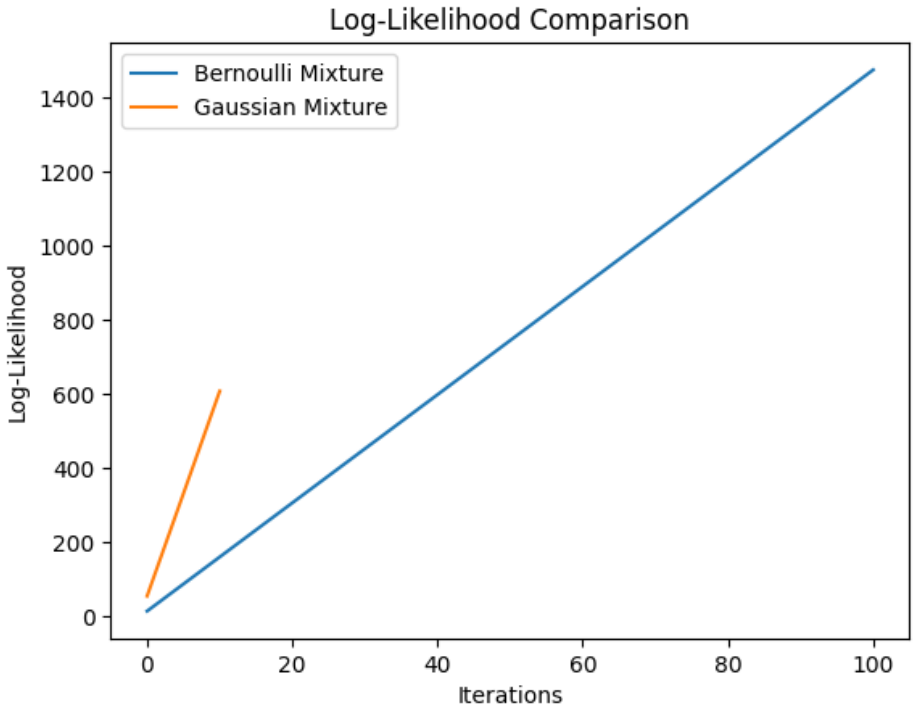

Figure 3:

- Gaussian mixture model converges much faster (in terms of iterations) compared to the Bernoulli mixture model but for lesser convergence rate i.e for gaussian 0.1 and for bernoulli 0.01.
- This suggests that the Gaussian mixture model may be a better fit for your data compared to the Bernoulli mixture model. However the computational time for Gaussian is much larger than bernoulli (nearly 4 times bernoulli).
- If computational efficiency is a priority and the performance difference between the two models is not critical for your

application, you might opt for the Bernoulli mixture model despite its lower log-likelihood.

- However, if model performance is paramount and the computational cost is acceptable, then the Gaussian mixture model may be the preferred choice due to its higher likelihood and potentially better fit to the data.

1.3 K-Means Algorithm

- The following is the method to apply K-Means algorithm:-

Step 1:- Initialization:

- Initialize z (cluster indicator array) with some random value {0,1,2,3}.

$$z_1^0, z_2^0, \ldots, z_n^0 \quad \epsilon \ \{0, \ldots, 3\}$$

Step 2:- Mean Calculation:

- Calculate the mean of all clusters $k = 4$ at iteration t given by

$$\mu_k^t = \frac{\sum_{i=1}^{n} x_i 1(z_i = k)}{\sum_{i=1}^{n} 1(z_i = k)} \quad \forall \ k$$

Step 3:- Reassignment step:

- Reassign all data points according to the following equation.

$$z_i^{t+1} = arg\ min\ k\ \|(x_i - \mu_k^t)\|^2$$

Step 4:- Convergence step:

- Check if there is no change in any cluster indicator then stop the algorithm and it means every point is in its desired cluster.

- Plot the following objective function of K-Means as function of iterations.

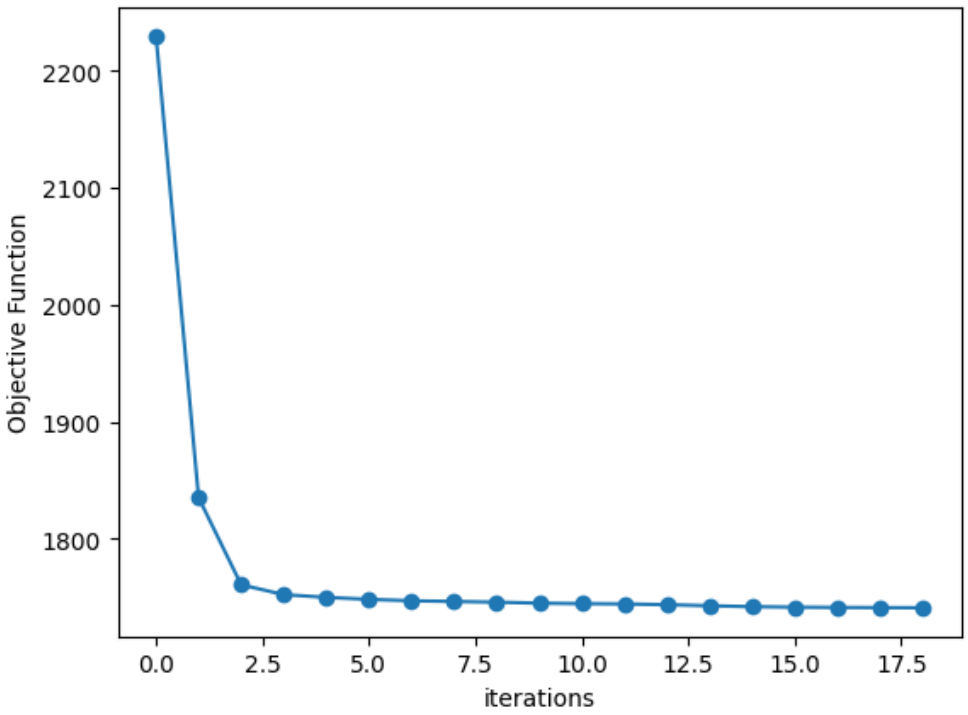

Figure 4:

$$Objective\ function = \sum_{i=1}^{n} \|(x_i - \mu_{z_i})\|^2$$

1.4 Best Algorithm for this dataset

- So to compare which algorithm is better we will evaluate the obejctive function for all.
- For GMM and BMM, from lamda will get for each data point which cluster they belong then calculate mean of clusters and then objective function given as below:

$$Objective\ function = \sum_{i=1}^{n} \|(x_i - \mu_{z_i})\|^2$$

Objective function value for BMM = 1742.4589023792912
Objective function value for GMM = 2295.7199999999966
Objective function value for K-Means = 1743.3803323987788

- So, EM algorithm with bernoulli mixture is best for the data as data is also binary and objective function is also minimum and log-likelihood values are also large then K-Means can be a good algorithm because of computationally less expensive. EM algorithm for Gaussian mixture is not so good because objective function is large and computational takes long time.

2 REGRESSION

- The training data has been read with the help of panda library and stored in X in such a way that each column corresponds to a datapoint with 100 features and Y has labels corresponding to every datapoint.

2.1 Least Square Solution

- The least square solution to regression problem is given by the following equation :-

$$w_{ML} = (XX^T)^{-1}Xy$$

where

$$X = [x_1, x_2, \ldots, x_n]$$
$$y = [y_1, y_2, \ldots, y_n]^T$$

Step 1:- Transpose the Feature Matrix X:

- Transpose the feature matrix X to obtain X^T, where each row represents a data point and each column represents a feature.

Step 2:- Compute XX^T:

- Multiply the feature matrix X with its transpose X^T to obtain the matrix XX^T. This results in a square matrix od size $d \times d$, where d is the number of features.

Step 3:- Compute the pseudo inverse of XX^T:

- Calculate the pseudo inverse of the matrix XX^T. This can be done using various numerical methods, such as the LU decomposition method or using matrix libraries available in programming languages like NumPy.

Step 4:- Compute Xy:

- Perform the dot product of the feature matrix X with the target vector y to obtain the vector Xy. This results in a vector of size d, where each element represents the dot product of the corresponding feature with the labels.

Step 5:- Multiply $(XX^T)^{-1}$ with Xy:

- Mutliply the inverse of XX^T obtained in step 3 with vector Xy. this operation yields vector $(XX^T)^{-1}Xy$, which represents the optimal weights w_{ML}.

Step 6:- Final Result:

- The resulting vector w_{ML} contains the optimal weights for the linear regression model obtained using the maximum likelihood estimation method. Each element of w_{ML} corresponds to the weight assigned to the respective feature in predicting the target variable y.

$$w_{ML} = \begin{bmatrix} -7.84961008 \times 10^{-3} \\ -1.36715320 \times 10^{-2} \\ -3.61656438 \times 10^{-3} \\ 2.64909162 \times 10^{-3} \\ 1.88551446 \times 10^{-1} \\ 2.65314657 \times 10^{-3} \\ \vdots \\ -8.58616116 \times 10^{-3} \end{bmatrix}$$

for all values, check the code.

2.2 Gradient Descent Algorithm

- Gradient Descent is an optimization algorithm used to minimize a function by iteratively moving in the direction of steepest descent as defined by the negative of the gradient. In the context of finding the optimal weights w for a linear regression model, the algorithm iteratively adjusts the weights to minimize the mean squared error (MSE) between the predicted values and the actual values.

$$w^* = (XX^T)^{-1}Xy$$

Step 1:- Initialize w^0:

- Start with an initial guess denoted as w^0. This could be randomly chosen or set to zeros.

$$w^0 = [0, \ldots, 0]$$

Step 2:- Compute the Gradient of the Loss Function:

- Calculate the gradient of the loss function $f(w)$ with respect to the weights w. The loss function is typically the

mean squared error (MSE) between the predicted values and the actual values. The gradient is computed as:

$$\bigtriangledown f(w) = 2(XX^T)w - 2Xy$$

Step 3:- Update w:

- Update the weights using gradient descent with a learning rate η :
  for $t = (1, 2, \ldots, T)$

$$w^{t+1} = w^t - \eta \bigtriangledown f(w^t)$$

- The step size chosen for this algorithm is as follows :-

$$\eta = \frac{10^{-6}}{t}$$

Step 4:- Final Result:

- The final w obtained after T rounds represents the optimal weights for the linear regression model using gradient descent algorithm.

$$w_{Grad} = w^T$$

- Plot $||w^t - w_{ML}||^2$ as function of iteration t.

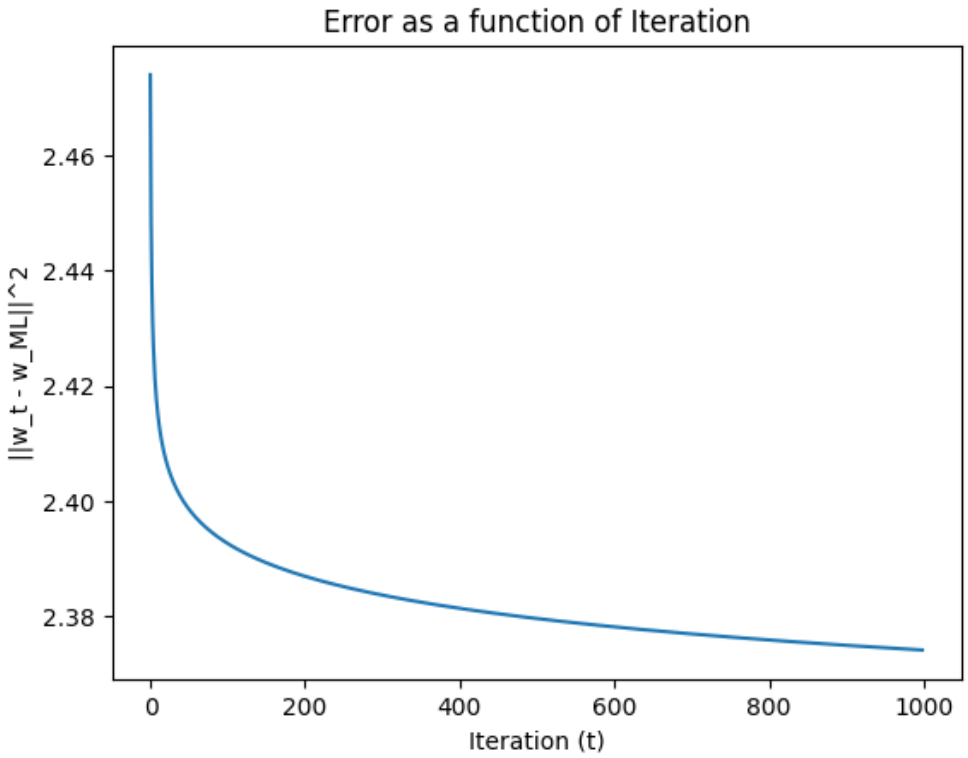

Figure 5:

- **Obseravtion** :-

  1. **Initial Rapid Decrease:** In the early iterations, the error typically decreases rapidly as the algorithm makes significant progress towards minimizing the loss function. This initial phase often reflects the algorithm's ability to quickly adjust the weights to reduce prediction errors.
  2. **Slower Convergence:** As the algorithm progresses, the rate of decrease in the error may slow down. This phenomenon occurs as the algorithm approaches the minimum of the loss function. The rate of convergence can be influenced by factors such as the learning rate and the curvature of the loss landscape.

3. **Convergence Criteria:** The point at which the algorithm terminates (convergence) can be identified from the error curve. Convergence criteria may be based on reaching a certain threshold of error, achieving a maximum number of iterations, or observing negligible changes in the error over successive iterations.
4. **Learning Rate Impact:** The choice of learning rate can significantly impact the convergence behavior. Too small a learning rate may lead to slow convergence, while too large a learning rate may cause oscillations or divergence. Experimentation with different learning rates can help identify an optimal value which in our case has been identified as mentioned above in step size.

2.3 Stochastic Gradient Descent Algorithm

- Stochastic Gradient Descent (SGD) is a popular optimization algorithm commonly used in machine learning and deep learning. It is an extension of the Gradient Descent algorithm and is particularly useful when dealing with large datasets.

  Algorithm :-

- Start with an initial guess denoted as w^0. This could be randomly chosen or set to zeros.

$$w^0 = [0, \ldots, 0]$$

  for $t = 1, \ldots, T$

- At each step, sample a bunch ($Q = 100$) of datapoints uniformly at random from the set of all points.

- Pretend this is your entire dataset and take a gradient step with respect to it.

$$[2(\tilde{X}\tilde{X^T}w^* - \tilde{X}\tilde{y})] \quad \tilde{X} \in \mathcal{R}^{d \times Q}$$

- Update the weights using gradient descent with a learning rate η :
  for $t = (1, 2, \ldots, T)$

$$w^{t+1} = w^t - \eta \bigtriangledown f(w^t)$$

- The step size chosen for this algorithm is as follows :-

$$\eta = \frac{10^{-6}}{t}$$

- The final w obtained after T rounds represents the optimal weights for the linear regression model using gradient descent algorithm. It is guaranteed to converge to optima with high probability.

$$w_{SGD}^T = \frac{\sum_{t=1}^{T} w^t}{T}$$

- Plot $||w^t - w_{ML}||^2$ as function of iteration t.

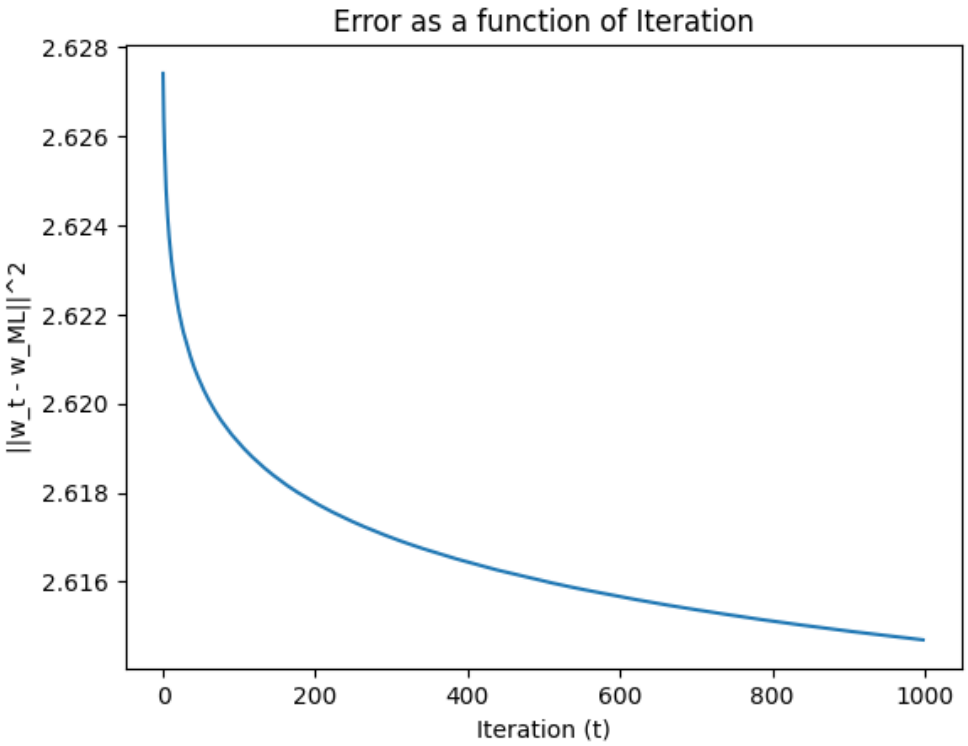

Figure 6:

- **Obseravtion** :-

  1. **Speed and Efficiency:** SGD is generally faster and more efficient compared to traditional gradient descent algorithms, especially when dealing with large datasets. By updating parameters based on only a subset of data at each iteration, SGD avoids the computational burden of computing gradients for the entire dataset.
  2. **Stochastic Nature:** Due to its stochastic nature, SGD introduces randomness into the optimization process. This stochasticity can help the algorithm escape local minima and saddle points, making it less likely to get stuck in suboptimal solutions. However, it can also lead to noisy updates, which may require careful

tuning of hyperparameters.

3. **Learning Rate Sensitivity:** The learning rate parameter in SGD plays a crucial role in determining the convergence behavior of the algorithm. Choosing an appropriate learning rate is essential for achieving convergence and stability. Too small a learning rate can result in slow convergence, while too large a learning rate can lead to oscillations or divergence.
4. **Mini-Batch Size:** The choice of mini-batch size in mini-batch SGD affects the trade-off between computational efficiency and optimization performance. Smaller mini-batch sizes introduce more stochasticity but may lead to slower convergence, while larger mini-batch sizes reduce variance but may increase computational overhead.
5. **Convergence and Robustness:** While SGD may converge to a suboptimal solution due to its noisy updates, it is generally robust and can handle non-convex and non-smooth objective functions. Techniques such as learning rate schedules, momentum, and adaptive learning rates are often employed to improve convergence and stability.

2.4 Ridge Regression

- Ridge Regression is a regression technique that is widely used for dealing with multicollinearity and overfitting in linear regression models.
- The solution to Ridge regression is as follows:

$$\hat{w}_R = (XX^T + \lambda I)^{-1} Xy$$

- To find λ, we have to do cross-validation.

Training Set:	**Validation Set:**
80%	20%

- For the values of $\lambda \in \{0.001, 0.01, 0.1, 1, 10, 100\}$, evaluate the w_R from training set and calculate error on validation set.

$$Error = \|(X_{val}^T w_R - y_{val})\|^2 + \lambda \|w_R\|^2$$

- Plot the error in validation set as a function of λ.

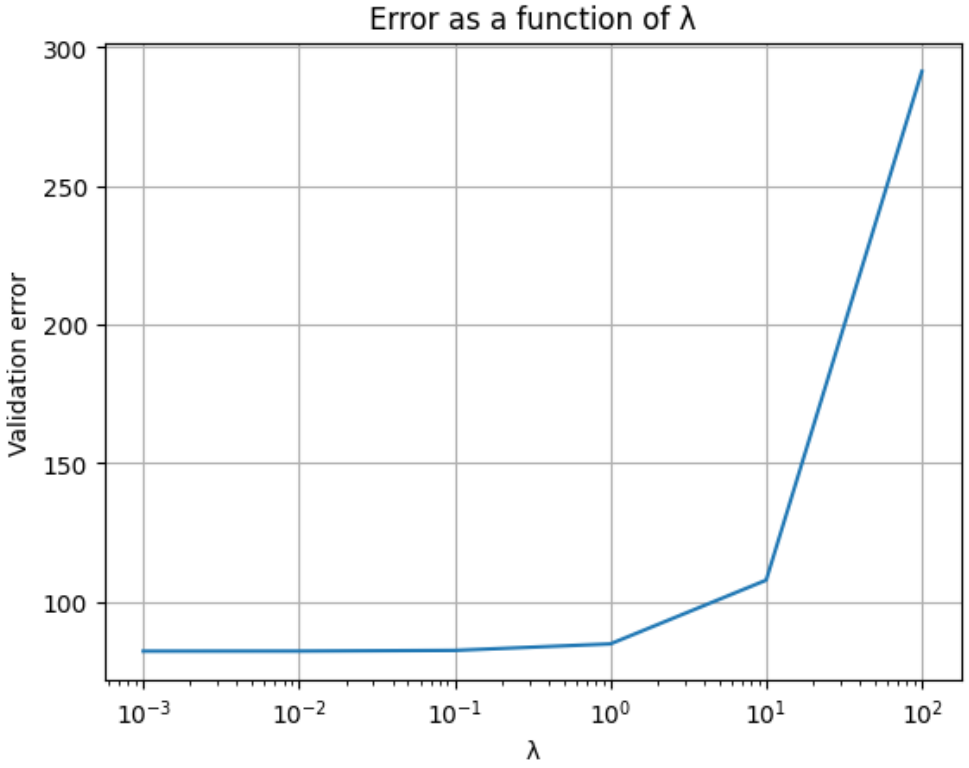

Figure 7:

- From the plot, we will choose the value of λ which minimizes the error in validation set and get w_R for the best λ.

$$\lambda = \mathbf{0.001}$$

$$w_R = \begin{bmatrix} -6.47536459 \times 10^{-3} \\ -1.59033021 \times 10^{-2} \\ -4.22580659 \times 10^{-3} \\ 9.43994738 \times 10^{-3} \\ 1.76614682 \times 10^{-1} \\ 1.15138252 \times 10^{-2} \\ \vdots \\ -8.36412009 \times 10^{-3} \end{bmatrix}$$

for all values, check the code.

- Compare the test error of w_R with w_{ML} on test data.

$$Error = \|(X_{test}^T w_R - y_{test})\|^2 + \lambda \|w_R\|^2$$

error in test dataset from $\mathbf{w_{ML}}$ is **2491.7628703781365**
error in test dataset from $\mathbf{w_R}$ is **2478.290466549822**

- This proves the existence theorem that
$\exists\, \lambda \in \mathcal{R}$ s.t.

$$\hat{w_R} = (XX^T + \lambda I)^{-1} Xy$$

has lesser MSE than $\hat{w}_{ML}$

- **Ridge Regression** is better than least sqaure solution (linear regression).The following are the reasons for the same:

  1. **Error:** Error in the validation set is smaller for ridge regression solution than linear regression.
  2. **Regularization:** Ridge Regression includes a penalty term that helps prevent overfitting by penalizing large coefficients. With 100 dimensions, Ridge Regression can help mitigate the effects of multicollinearity and stabilize the estimates of the coefficients.

3. **Stability:** Ridge Regression tends to be more stable than ordinary linear regression when features are highly correlated or when the number of predictors is large relative to the number of observations.
4. **Generalization:** Ridge Regression often leads to better generalization performance compared to Linear Regression, especially in high-dimensional feature spaces.