CS5691: Pattern Recognition and Machine Learning
Assignment 2

Lalit Jayanti, ME21B096

Question 1

Part (i): Deriving EM algorithm

A probabilisitic mixture that could have generated this data, is a mixture of Bernoulli trials. This is considered as all data points are binary in nature and hence, Bernoulli trials are a natural fit for this kind of data.

Description of the Probabilistic mixture:
For each data point $x_i = [{x_i}^1, {x_i}^2 ... {x_i}^d] \in \mathbb{R}^d$, ${x_i}^j$ is drawn from a Bernoulli trial, of probability of success ${p_k}^j$ where $1 \leq j \leq d$. Here k is the mixture from which x_i is drawn, and probability of drawing from a particular mixture is π_k. Here $1 \leq k \leq K$, where K is the total number of mixtures.

The matrix of $K \times d$ probabilities ${p_k}^j$ is called $\theta \in \mathbb{R}^{K \times d}$ in this work. And θ_k refers to k^{th} row of θ.
For the given dataset `A2Q1.csv`

- Dimension $d = 50$
- Number of mixtures $K = 4$
- Number of points $n = 400$

Figure 1 shows a schematic of the probabilistic mixture model described above.

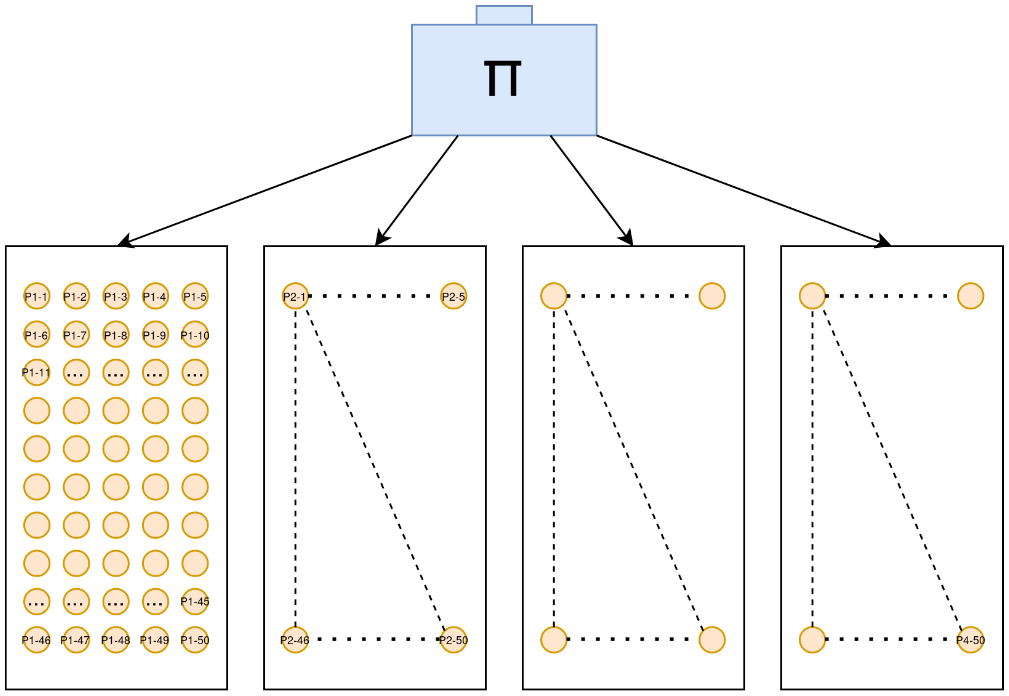


Figure 1: Bernoulli Mixture Model

EM Derivation

CS5691 ①

Derivation of EM algorithm

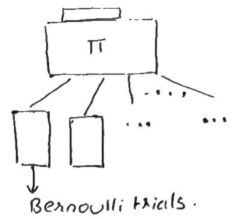


Fig. Schematic of the model

- Notation

* Data: $X = \{ x_1 \dots x_n \}$, $x_i \in \mathbb{R}^d \;\forall\; 1 \le i \le n$

$x_i = \{ x_i^1 \dots x_i^d \}$ $\quad x_i^j \in \mathbb{R} \quad \forall\; 1 \le j \le d$

* $\pi = \{ \pi_1 \cdots \pi_k \}$ $\quad \pi_k \in \mathbb{R},\; 0 \le \pi_k \le 1 \;\forall\; 1 \le k \le K$

such that,

$$\sum_{k=1}^{K} \pi_k = 1$$

* $$\Theta = \begin{bmatrix} p_1^1 & \cdots & p_1^d \\ \vdots & & \vdots \\ p_K^1 & \cdots & p_K^d \end{bmatrix} \quad \Theta \in \mathbb{R}^{K \times d}$$

$0 \le p_k^j \le 1 \quad \forall\; 1 \le j \le d,\; 1 \le k \le K$

Assume $\{ x_1 \cdots x_n \}$ drawn from i.i.d Bernoulli trials (each x_i^j is a bernoulli trial)

- Likelihood

$$\mathcal{L}(\Theta; X) = \prod_{i=1}^{n} p(x_i; \Theta) \quad \text{(By i.i.d)}$$

$$= \prod_{i=1}^{n} \Big[\sum_{k=1}^{K} \pi_k\, p(x_i; \Theta_k) \Big] \quad \left(\begin{array}{l} \text{By thm of total Probability} \\ \text{and } \Theta_k \text{ is } k^{th} \text{ row of } \Theta \end{array} \right)$$

$$= \prod_{i=1}^{n} \Big[\sum_{k=1}^{K} \pi_k \Big(\prod_{j=1}^{d} (p_k^j)^{x_i^j} (1 - p_k^j)^{1 - x_i^j} \Big) \Big]$$

[Note: $p(x_i; \Theta_k) = \prod_{j=1}^{d} (p_k^j)^{x_i^j} (1 - p_k^j)^{1-x_i^j}$ [Bernoulli trials]]

* Introduce $\lambda_k^i \quad 1 \le i \le n,\; 1 \le k \le K$

$$\log \mathcal{L}(\Theta; X) = \sum_{i=1}^{n} \log \Big[\sum_{k=1}^{K} \lambda_k^i \Big(\frac{\pi_k \prod_{j=1}^{d} (p_k^j)^{x_i^j} (1 - p_k^j)^{1-x_i^j}}{\lambda_k^i} \Big) \Big]$$

* log is concave, applying Jensen's inequality.

$$\text{mod}\cdot\log \mathcal{L}(\Theta; X) = \sum_{i=1}^{n} \sum_{k=1}^{K} \lambda_k^i \log \Big(\frac{\pi_k\, p(x_i; \Theta_k)}{\lambda_k^i} \Big)$$

$$= \sum_{i=1}^{n} \sum_{k=1}^{K} \lambda_k^i \Big[\log \Big(\frac{\pi_k}{\lambda_k^i} \Big) + \sum_{j=1}^{d} \big(x_i^j \log p_k^j + (1 - x_i^j) \log (1 - p_k^j) \big) \Big]$$

Here

$\log \mathcal{L}(\Theta) \ge \text{mod} \log \mathcal{L}(\Theta, \lambda)$ (Since log is concave)

Note: constraint on λ is $\displaystyle\sum_{k=1}^{K} \lambda_k^i = 1 \;\forall\; i \quad 0 \le \lambda_k^i \le 1 \;\forall\; k$

2

• Obtaining Maximum Likelyhood estimators (θ, Π)

$$\frac{\partial \text{ mod} \cdot \log L(\theta)}{\partial P_k^j} = \sum_{i=1}^{n} \lambda_k^i \left(\frac{x_i^j}{P_k^j} - \frac{(1-x_i^j)}{1-P_k^j} \right) = 0$$

$$\therefore \sum_{i=1}^{n} \left(\lambda_k^i \left(x_i^j - x_i^j P_k^j - P_k^j + P_k^j x_i^j \right) \right) = 0$$

$$\Rightarrow \boxed{P_k^j = \frac{\sum_{i=1}^{n} \lambda_k^i x_i^j}{\sum_{i=1}^{n} \lambda_k^i}} \quad \forall 1 \le j \le d, \ 1 \le k \le K \text{ ---- } (1)$$

Now,

$$\max_{\substack{\Pi_1 \cdots \Pi_K \\ \text{s.t} \sum_{k=1}^{K} \Pi_k = 1}} \text{mod} \log L(\theta) \equiv \min_{\substack{\Pi \in \mathbb{R}^K \\ \text{s.t} \sum_{k=1}^{K} \Pi_k = 1}} - \sum_{i=1}^{n} \sum_{k=1}^{K} \lambda_k^i \left(\log\left(\frac{\Pi_k}{\lambda_k^i}\right) + \log P(x_i, \theta_k) \right)$$

Reparametrize $\Pi_k = e^{\gamma_k}$, $\sum e^{\gamma_k} = 1$

$$\nabla(-\text{mod} \log(\theta, \lambda))_k = -\mathcal{L} e^{\gamma_k} \quad (\mathcal{L} \to \text{Lagrange multiplier})$$

$$\therefore \sum_{i=1}^{n} \lambda_k^i = +\mathcal{L} e^{\gamma_k} \Rightarrow \Pi_k = \frac{\sum_{i=1}^{n} \lambda_k^i}{\mathcal{L}}$$

$$\text{now } \sum_{k=1}^{K} \Pi_k = 1 = \frac{\sum_{i=1}^{n} \sum_{k=1}^{K} (\lambda_k^i)}{\mathcal{L}} = \frac{n}{\mathcal{L}} \Rightarrow n = \mathcal{L}$$

$$\therefore \boxed{\Pi_K = \frac{\sum_{i=1}^{n} \lambda_k^i}{n}} \quad \forall 1 \le k \le K \text{ ---- } (2)$$

Now Maximize mod $\log L(\theta)$ over λ with (θ, Π) as constants.

$$\max_{\substack{\lambda_k^i \cdots \\ \text{s.t} \sum_{i=1}^{K} \lambda_k^i = 1}} \sum_{k=1}^{K} \lambda_k^i \left(\log\left(\frac{\Pi_k}{\lambda_k^i}\right) + \log\left(P(x_i, \theta_k)\right) \right) \equiv \min_{\substack{\lambda_1^i \cdots \lambda_K^i \\ \text{s.t} \sum_{1=1}^{K} \lambda_k^i = 1}} - \sum_{k=1}^{K} \lambda_k^i \left(\log\left(\frac{\Pi_k}{\lambda_k^i}\right) + \log\left(P(x_i, \theta_k)\right) \right)$$

Reparametrize $\lambda_k^i = e^{\gamma_k}$

$$\nabla\left(-\sum_{k=1}^{K} \lambda_k^i \left(\log\left(\frac{\Pi_k}{\lambda_k^i}\right) + \log\left(P(x_i; \theta_k) \right) \right) \right)_k = -\mathcal{L} e^{\gamma_k} \quad (\mathcal{L} \text{ is lagrange multiplier})$$

$$\therefore e^{\gamma_k} \left(\log \Pi_k + \log(P(x_i; \theta_k)) - \gamma_k - 1 \right) = \mathcal{L} e^{\gamma_k}$$

$$\Rightarrow \gamma_k = \log \Pi_k P(x_i; \theta_k) - \mathcal{L} - 1 = \log \lambda_k^i$$

$$\therefore (e^{-\mathcal{L}-1}) \Pi_k P(x_i; \theta_k) = \lambda_k^i \quad (\text{take exp on both sides})$$

$$\text{Now } \sum_{k=1}^{K} \lambda_k^i = 1 \ \therefore \ e^{-\mathcal{L}-1} = \frac{1}{\sum_{k=1}^{K} \Pi_k P(x_i; \theta_k)} \Rightarrow \boxed{\lambda_k^i = \frac{\Pi_k P(x_i, \theta_k)}{\sum_{l=1}^{K} \Pi_l P(x_i; \theta_k)}} \text{ - } (3)$$

Therefore, the formulas for the algorithm are as follows,
Probability of occurrence of x_i by k^{th} mixture

$$p(x_i;\theta_k) = \prod_{j=1}^{d} (p_k^j)^{x_i^j}(1-p_k^j)^{1-x_i^j} \qquad \forall 1 \le k \le K, 1 \le i \le n$$

Expectation step:

$$\lambda_k{}^i = \frac{\pi_k p(x_i;\theta_k)}{\sum_{l=1}^{K} \pi_l p(x_i;\theta_l)} \qquad \forall 1 \le k \le K, 1 \le i \le n$$

Maximisation step:

$$p_k^j = \frac{\sum_{i=1}^{n} \lambda_k^i x_i^j}{\sum_{i=1}^{n} \lambda_k^i} \qquad \forall 1 \le j \le d, 1 \le k \le K$$

$$\pi_k = \frac{\sum_{i=1}^{n} \lambda_k^i}{n} \qquad \forall 1 \le k \le K$$

Note that p_k^j is the probability of obtaining 1 at the j^{th} location in ($1 \le j \le d$) by the k^{th} mixture.

Log Likelihood Plots: EM algorithm is implemented and tested on the given dataset. Following Figure 2 is the average log likelihood is computed over 100 random initializations.

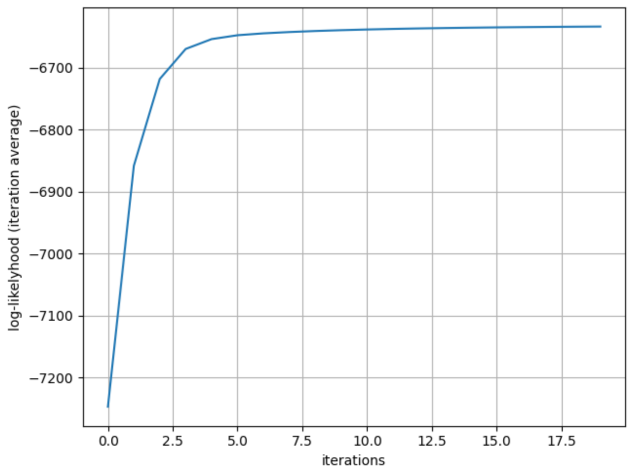


Figure 2: Average Log likelihood

Further, the obtained probabilities (θ, see derivation section 2) are reshaped to $K = 4$, 10×5 matrices (according to the given hint) matrices and are visualized to analyze the patterns obtained (Figure 3). It is observed that two mixtures consist of data points with one square or a "flipped P" in the center, while other 2 mixtures consist of data points with 2 squares or a "flipped R" like pattern.

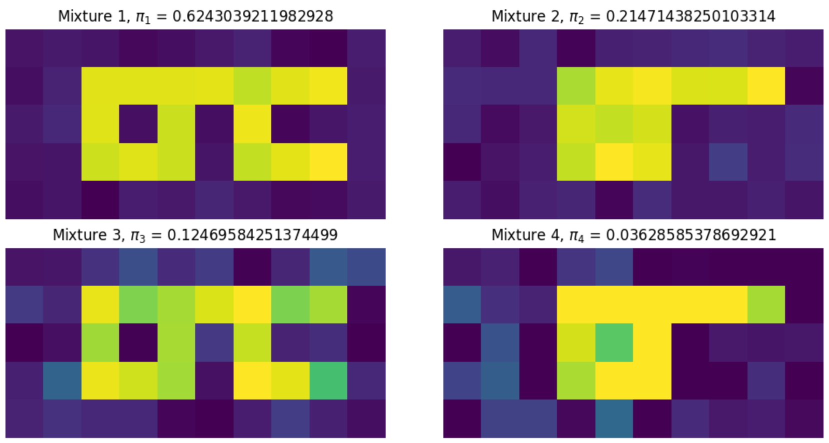


Figure 3: Pictorial representation of θ_k for $1 \leq k \leq K = 4$

Implementation:

- Program for Mixture Model: `scripts/bernoulli_em.py - bernoulliEM()`
- Plotting: `scripts/question_i.ipynb - Sub-question i.`
- Data: `data/A2Q1.csv`

Part (i): Gaussian Mixture Model

EM algorithm is implemented for Gaussian Mixture model. Figure 4 shows the average log likelihood for this model. The average is computed over 100 random initializations of parameters.

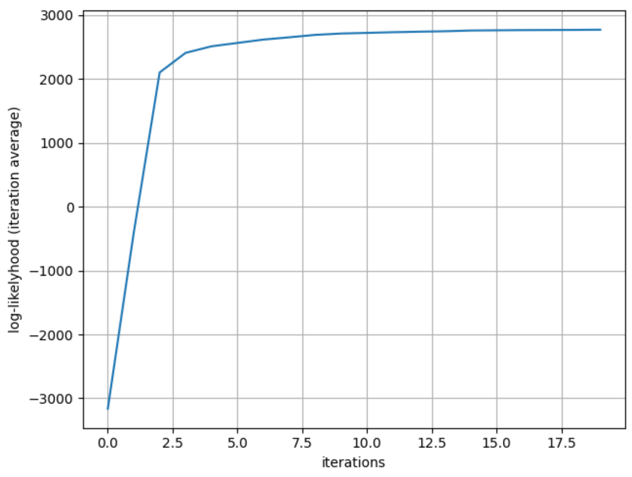


Figure 4: Average Log likelihood

The log likelihood for the Gaussian mixture model (Figure 4) is seen to be much larger than Bernoulli mixture model (Figure 2) for all iterations as well as averaged over multiple initializations. However, it must be noted that the Gaussian mixture model is over a continuous distribution and makes use of the probability density function which may even be greater than 1 depending on the covariance, while the Bernoulli mixture model is over a discrete distribution and makes use of the probability mass function which is always less than or equal to 1. This explains the difference in the magnitude.

Another insight is that, the covariance matrices for the Gaussian mixture model often become singular causing the log likelihood to become very large. Upon investigation, the root cause for this is found to be that, one (or more) of the mixtures "fits" to all the copies of a single data point in the dataset. The covariance of such a mixture is naturally singular. Here the term "fit" is used to define the condition where λ_k^i is the largest among all k for a given i.

Applying a Gaussian mixture model to a dataset which does not follow such a model can therefore lead to such cases of overfitting. Hence, this model may not be suitable to practical applications. In this particular implementation, a small regularization term ($10^{-6}I$) is added to the covariance when it becomes singular, this ensures that the log likelihood does not blow up.

Implementation:

- Program for Mixture Model: `scripts/gaussian_em.py - gaussianEM()`
- Plotting: `scripts/question_i.ipynb - Sub-question ii.`
- Data: `data/A2Q1.csv`

Part (ii): K-Means on the dataset

The K-Means algorithm is run on the given dataset. The objective is plotted as a function of iterations in Figure 5

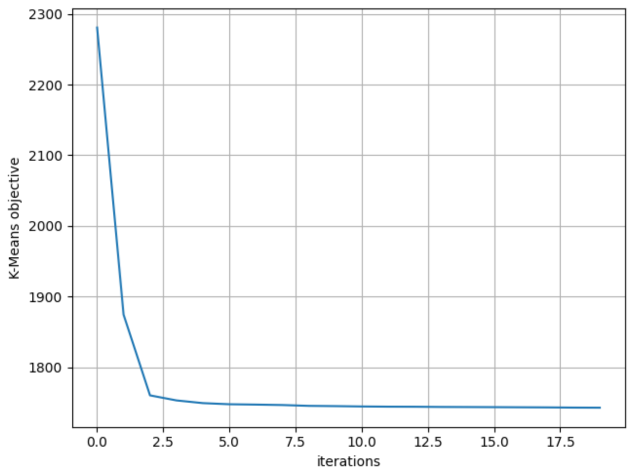


Figure 5: K-Means Objective

Implementation:

- Program for K-Means: `scripts/k_means.py - Kmeans()`
- Plotting: `scripts/question_i.ipynb - Sub-question iii.`
- Data: `data/A2Q1.csv`

Part (iii): Results / Conclusions

The Bernoulli mixture model is most appropriate for this dataset, for the following reasons.

- Data is binary in nature, hence it is appropriate to assume that the underlying distribution is discrete (such as Bernoulli distribution) as compared to a continuous distribution such as Gaussian distribution.
- It is observed that the Gaussian mixture model often results in pathological solutions (singular covariance) and over fitting (detailed in subsection (i)). Where as no such issue is observed for the Bernoulli mixture model.

- While, K-means measures similarity in data points by Euclidean distance, it may not be an appropriate model for binary data. (Eucledian distance may not be a good metric for this data)
- Further, K-Means provides a deterministic result while the mixture models provide a probabilistic assignment resulting in better performance at boundaries of clusters.
- Moreover, Bernoulli mixture model provides a more insightful result in form of the probabilities (θ), which reveal the internal structure of the data better (see Figure 3).
- Finally, the Bernoulli mixture model, provides a model to generate more data points similar to the original data points (as compared to the Gaussian mixture model) and can therefore be considered a better "compressed representation".

Question 4

Part (i): Linear Regression - Analytical Solution

A program is written to find the analytical solution to the least squares solution $\mathbf{W}_{\text{ML}}$.

$$\mathbf{W}_{\text{ML}} = (XX^T)^{-1}Xy \qquad \text{Analytical Solution}$$

Following is the solution:

```
[-7.84961009e-03 -1.36715320e-02 -3.61656438e-03  2.64909160e-03
  1.88551446e-01  2.65314657e-03  9.46531786e-03  1.79809481e-01
  3.73757317e-03  4.99608944e-01  8.35836265e-03  4.29108775e-03
  1.42141179e-02  3.94232414e-03  9.36795890e-03 -1.12038274e-03
  3.35727500e-03  1.16152212e-03 -9.40884707e-03 -2.45575476e-03
 -1.17409629e-02 -1.01960612e-02  7.95771321e-03 -1.00574854e-02
  6.04882939e-03 -4.67345192e-03 -3.09091547e-03  8.14909193e-03
  1.20264599e-02 -6.82458163e-03 -8.65405539e-03  9.86273479e-04
  4.92968011e-03  5.99772461e-03 -1.34667860e-02  1.07075729e-03
  1.32745992e-02 -1.14148742e-02 -2.01056697e-02  5.85096240e-01
  4.94483247e-04 -7.86666920e-04 -2.71926574e-03 -9.54021938e-03
 -5.44161058e-03  9.80679209e-03 -6.72540624e-03 -4.45414276e-04
  6.98516508e-03  3.16138907e-02  4.51763485e-01 -8.75221380e-03
  2.55167390e-03  4.24921150e-03  2.89847927e-01  7.03723255e-03
 -1.95796946e-03  1.41523883e-02 -1.06508170e-02  7.72743903e-01
 -5.67126044e-03 -6.30026188e-04  6.50943015e-03 -4.84019165e-03
  4.63832329e-03  4.54887177e-03 -2.99475114e-03  8.38781696e-03
 -2.47558716e-03  9.00947922e-04  1.14713514e-03 -1.87641345e-03
 -1.05175760e-02 -9.31304110e-03 -1.23550002e-03  5.97797559e-01
 -4.78625013e-03 -1.13727852e-02  2.88477060e-03  8.48999776e-01
 -1.08924235e-02  2.26346489e-03 -1.38099800e-03 -6.35934691e-03
  5.83784109e-03  5.69286755e-03  5.35566859e-03 -8.20616315e-03
  1.29884015e-02 -2.30575631e-03 -1.22263765e-04  8.66629171e-03
 -4.29446300e-03  5.69510898e-03  7.55483353e-03 -9.43540843e-03
  1.82905446e-02 -1.16998887e-03 -2.61599136e-03 -8.58616114e-03]
```

Implementation:

- Program for Linear Regression: `scripts/linear_regression.py - linear_regression()`
- Printing: `scripts/question_2.ipynb - Sub-question i.`
- Data: `data/A2Q2Data_train.csv`

Part (ii): Gradient Descent

The solution to the linear regression problem is found using gradient descent algorithm. The error $||\mathbf{w}^{\mathrm{t}} - \mathbf{w}_{\mathrm{ML}}||_2$ is plotted as a function of t (iterations).

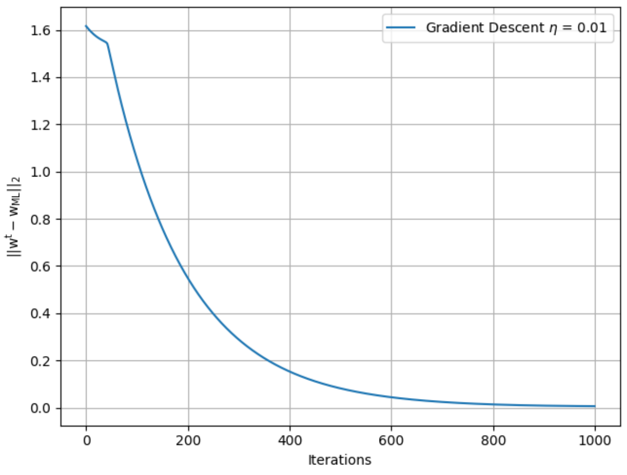


Figure 6: Gradient Descent

For this work,

$$\mathbf{W}^{t+1} = \mathbf{W}^{t} - \eta \frac{\nabla f(\mathbf{W}^t)}{||\nabla f(\mathbf{W}^t)||} \qquad \text{Gradient Descent}$$

$$\nabla f(\mathbf{W}^t) = 2(XX^T)\mathbf{W}^t - 2Xy \qquad \text{Gradient}$$

$$\text{Step size} = \frac{\eta}{||\nabla f(\mathbf{W}^t)||} \qquad \eta = 10^{-2}$$

It is observed that the solution converges to $\mathbf{W}_{\mathrm{ML}}$, with an error of around 10^{-3} in about 1000 iterations. However, there is a computational overhead in computing the value XX^T since it is $((100 \times 10000) \times (10000 \times 100))$ matrix multiplication.

Suitable step size is chosen after experimentation, using smaller η results in slow convergence and using large η results in oscillations of the error. Here $\eta = 0.01$ works well for this dataset.

Implementation:

- Program for Gradient Descent: `scripts/linear_regression.py - gradient_descent_linear_regression()`
- Plotting: `scripts/question_2.ipynb - Sub-question ii.`
- Data: `data/A2Q2Data_train.csv`

Part (iii): Stochastic Gradient Descent

The solution to the linear regression problem is found using stochastic gradient descent algorithm (with a batch size of 100). The error $||\mathbf{w}^{\mathrm{t}} - \mathbf{w}_{\mathrm{ML}}||_2$ is plotted as a function of t (iterations).

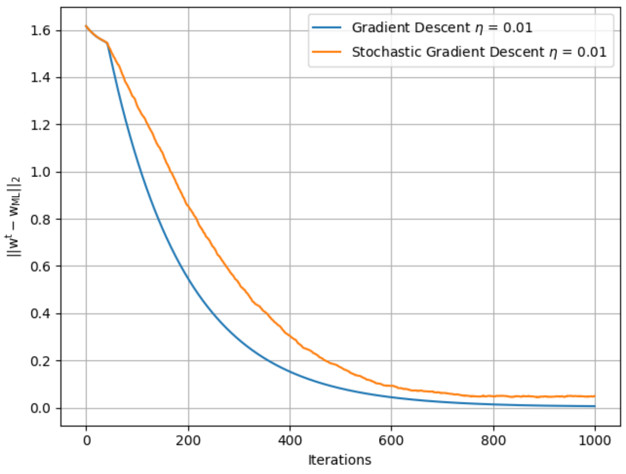


Figure 7: Stochastic Gradient Descent

For this work,

$$\mathbf{W}^{t+1} = \mathbf{W}^t - \eta \frac{\nabla f(\mathbf{W}^t)}{||\nabla f(\mathbf{W}^t)||} \qquad \text{Gradient Descent}$$

$$\nabla f(\mathbf{W}^t) = 2(\tilde{X}\tilde{X}^T)\mathbf{W}^t - 2\tilde{X}\tilde{y} \qquad \text{Gradient}$$

$$\text{Step size} = \frac{\eta}{||\nabla f(\mathbf{W}^t)||} \qquad \eta = 10^{-2}$$

Here, $\tilde{X}$ and $\tilde{y}$ are the sets of 100 (batch size) randomly sampled data points from X and y
It is observed that the solution converges to $\mathbf{W}_{\mathrm{ML}}$, with an error of around 10^{-2} in about 1000 iterations. Moreover, there is only a small computational overhead in computing the value $\tilde{X}\tilde{X}^T$ since it is $((100 \times 100) \times (100 \times 100))$ matrix multiplication (as compared to subsection (ii)). Subsequently, Stochastic gradient descent runs faster per iteration as compared to Gradient Descent.

However, Stochastic Gradient Descent requires more number of iterations to converge to $\mathbf{W}_{\mathrm{ML}}$ with the same error tolerance as compared to Gradient Descent(when same learning rates are used).This can be attributed to the fact that less amount of data is used to compute gradient which may result in an inaccurate estimate of gradient.

Suitable step size is chosen after experimentation, using smaller η results in slow convergence and using large η results in oscillations of the error. Here $\eta = 0.01$ works well for this dataset.

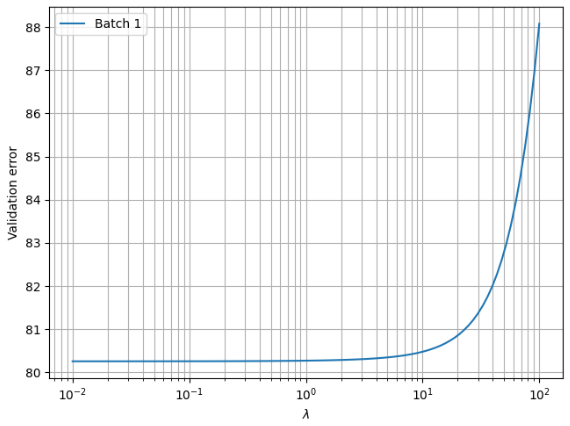
Implementation:

- Program for Gradient Descent: `scripts/linear_regression.py - stochastic_descent_linear_regression()`
- Plotting: `scripts/question_2.ipynb - Sub-question iii.`
- Data: `data/A2Q2Data_train.csv`

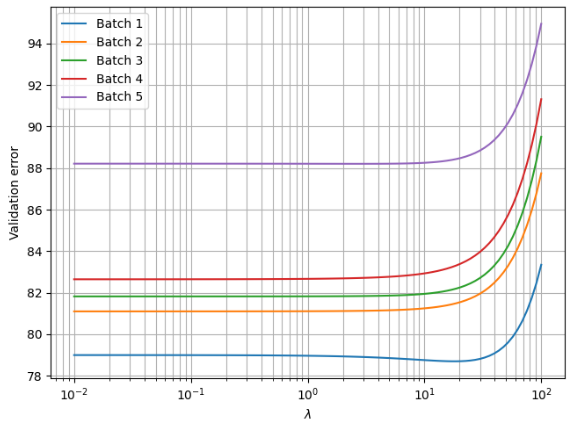
Part (iv): Ridge Regression

The solution to the ridge regression problem is found using gradient descent algorithm for a given value of λ.

This λ is then cross validated on a validation set using simple cross validation and K-Fold cross validation.



(a) Cross Validation with 20% data



(b) K-Fold Cross Validation with 5 folds (20% data)

Figure 8: Cross Validation for λ

Now the best λ is chosen from the cross validation by finding the λ at which validation error is minimum. This is then used to obtain $\mathbf{w}_R$. Test error is evaluated on the test set and presented in Table 1.

	λ	Test Error	MSE (Test Error / number of points)
Least Squares	-	185.3636556	0.3707273
Ridge Regression (Cross Validation)	0.01	184.7879664	0.3695759
Ridge Regression (K fold)	4.284852402310109	183.9077566	0.3678155

Table 1: Test errors for various λ

It is observed that the error on the test set is lower in case of Ridge Regression as compared to Linear regression (about 1 unit). And among the methods of cross validation, K-Fold cross validation yields a better λ in this case. This then verifies the expectation that biasing W in regression towards a smaller norm (motive for ridge regression) can potentially lead to better results on the test set. Hence, it is concluded that Ridge regression performs better for this dataset.

Along with this a few handpicked values of λ are tested to observe performance on the test set.

	λ	Test Error	MSE (Test Error / number of points)
Least Squares	-	185.3636556	0.3707273
Ridge Regression	1.0	184.5828777	0.3691658
Ridge Regression	10.0	182.7525098	0.3655050
Ridge Regression	1000.0	119.1062617	0.2382125
Ridge Regression	10000.0	108.0943723	0.2161887
Ridge Regression	100000.0	323.6804807	0.6473610

Table 2: Test errors for various handpicked λ

It observed that $\lambda = 10^4$ results in the smallest test error by a considerable margin (75 units, compared to ridge regression in Table 1). This shows that, the optimal λ obtained on validation sets may not be optimal for test set.

Implementation:

- Program for Ridge Regression: `scripts/ridge_regression.py - ridge_regression()`
- Program for Cross Validation: `scripts/ridge_regression.py - cross_validate_ridge_regression()`
- Plotting: `scripts/question_2.ipynb - Sub-question iv.`
- Data for training: `data/A2Q2Data_train.csv`
- Data for testing: `data/A2Q2Data_test.csv`