Introduction to ML (CS771), Autumn 2023
Indian Institute of Technology Kanpur
Homework Assignment Number 2

Student Name: Ruby Prajapati
Roll Number: 231110042
Date: November 16, 2023

QUESTION 1

SGD for K-means Objective

Loss function for standard K-means is given as:

$$L(X, Z, \boldsymbol{\mu}) = \sum_{n=1}^{N} \sum_{k=1}^{K} z_{nk} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2$$

To apply Stochastic Gradient Descent (SGD) to the standard K-means loss function, the update rule for the cluster centers $\boldsymbol{\mu}_k$ is:

$$\boldsymbol{\mu}_k = \boldsymbol{\mu}_k - \alpha \cdot \left(-2 \sum_{n=1}^{N} z_{nk} (\mathbf{x}_n - \boldsymbol{\mu}_k) \right)$$

where α is the learning rate. This update is performed iteratively, adjusting each $\boldsymbol{\mu}_k$ to minimize the K-means loss. The negative sign ensures that the update is in the direction of decreasing the objective.

Choosing an appropriate learning rate α involves trade-offs. Too large a step size may lead to overshooting the minimum, while too small a step size may slow down convergence. Commonly, a small positive constant or a decreasing step size ($\alpha = \frac{c}{t}$, where c is a constant and t is the iteration or the number of data points processed so far) is used.

This SGD-based approach allows for an online or incremental update of the cluster centers as data points are processed one at a time.

Step 1: Assigning x_n to the "Best" Cluster

Given the current cluster means $\{\mu_k\}_{k=1}^{K}$, we want to find the assignment z_n that minimizes the distance between x_n and the cluster mean.

$$z_n = \arg\min_k \sum_{k=1}^{K} z_{nk} \|x_n - \mu_k\|^2$$

This is essentially assigning x_n to the cluster with the closest mean.

Step 2: SGD-based Cluster Mean Update Equations

We want to update the cluster means $\{\mu_k\}_{k=1}^{K}$ using SGD on the objective function L. The update equation for μ_k can be derived by taking the gradient of L with respect to μ_k. So we can right following equation .

The optimized cluster centers $\hat{\boldsymbol{\mu}}$ are obtained by minimizing the K-means loss $L(X, Z, \hat{\boldsymbol{\mu}})$:

$$\hat{\boldsymbol{\mu}} = \arg\min_{\boldsymbol{\mu}} \sum_{n=1}^{N} \sum_{\hat{z}_n = k} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2$$

For a specific cluster k, the optimal $\boldsymbol{\mu}_k$ is found by minimizing the sum of squared distances:

$$\hat{\boldsymbol{\mu}}_k = \arg\min_{\boldsymbol{\mu}_k} \sum_{n:\hat{z}_n=k} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2$$

During the optimization process, we choose a data point $\mathbf{x}_n$ uniformly randomly, and approximate the gradient g as done in Stochastic Gradient Descent (SGD). At any iteration t, the gradient g_t with respect to $\boldsymbol{\mu}_k$ is given by:

$$g_t = \frac{\partial}{\partial \boldsymbol{\mu}_k} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2 \approx g_t^n = -2(\mathbf{x}_n - \boldsymbol{\mu}_k)$$

The update equation for the mean in SGD is then:

$$\boldsymbol{\mu}_k^{(t+1)} = \boldsymbol{\mu}_k^{(t)} - \eta g_t^n$$

where η is the learning rate, and $\boldsymbol{\mu}_k^{(t)}$ represents the cluster center at iteration t.

$$\mu_k^{(t+1)} = \mu_k^{(t)} + 2\eta \sum_{n:z_n=k} (x_n - \mu_k^{(t)})$$

Intuition for the Update Equation

1. **Gradient Descent Intuition:**
   - The update $\mu_k^{(t+1)} = \mu_k^{(t)} + 2\eta \sum_{n:z_n=k}(x_n - \mu_k^{(t)})$ is proportional to the negative gradient of the K-means objective function.
   - It adjusts the cluster mean in the direction that reduces the objective function, which is the sum of squared distances between data points and their assigned cluster means.
2. **Minimization of Loss Function:**
   - The goal of the K-means clustering algorithm is to minimize the sum of squared distances. The update equation is derived from this objective function, ensuring that the algorithm moves towards, where the loss is minimized.
3. **Online Learning:**
   - By updating the cluster means one data point at a time, the algorithm becomes "online," adapting to new data points gradually.
4. **Adaptation to Cluster Size:**
   - The learning rate η is a crucial parameter. A good choice is $\eta \propto \frac{1}{N_k}$, where N_k is the number of data points in cluster k.
   - This ensures that larger clusters contribute less to the update, preventing them from dominating the adjustment. It helps achieve balanced influence and prevents sensitivity to cluster size.

Suggested Step Size (Learning Rate)

A good choice for the step size (η) is inversely proportional to the number of data points in the cluster (N_k):

$$\eta \propto \frac{1}{N_k}$$

Here's why this is a reasonable choice:

- **Balanced Influence:** In groups with more data points, using a smaller step size ensures that larger groups don't have too much influence on the adjustment. This prevents big groups from dominating the decision-making process.
- **Avoiding Oscillations:** Using a smaller step size is like taking smaller steps when trying to find the best position for each group. It helps the algorithm make smooth adjustments without swinging back and forth too much, making the process more stable.
- **Adaptation to Cluster Density:** In areas where there are many data points close together, a smaller step size allows the algorithm to make precise adjustments to the group's center. It's akin to being more careful in crowded spaces to avoid collisions. This adaptability helps the algorithm work well in different situations, whether the groups are spread out or closely packed.

Overall, this choice of step size promotes stable convergence, balanced influence, and adaptability to different cluster sizes and densities.

Introduction to ML (CS771), Autumn 2023
Indian Institute of Technology Kanpur
Homework Assignment Number 2

Student Name: Ruby Prajapati
Roll Number: 231110042
Date: November 16, 2023

QUESTION 2

Given that we have data points that belong to two classes - let assume them the positive class and the negative class. Now, we want to find a direction (represented by the vector $\mathbf{w}$) in which, when you project these points, the means of the two classes are far apart, and the points within each class are as close as possible.

The objective function $J(\mathbf{w})$ is a way to quantify how well this separation and closeness are achieved. Here's a simplified explanation:

Numerator (Top part):

$$\|\mathbf{w}^T\boldsymbol{\mu}_1 - \mathbf{w}^T\boldsymbol{\mu}_{-1}\|^2$$

measures the difference between the projected means of the positive and negative classes. $\|\cdot\|^2$ takes the square of this difference. The larger this value, the better the separation of means.

Denominator (Bottom part):

$$\mathbf{w}^T\mathbf{S}_1\mathbf{w} + \mathbf{w}^T\mathbf{S}_{-1}\mathbf{w}$$

represents the dispersion or spread within each class after projection. Minimizing this term means that, within each class, the points are as close to each other as possible.

Overall Objective:

$$J(\mathbf{w}) = \frac{\|\mathbf{w}^T\boldsymbol{\mu}_1 - \mathbf{w}^T\boldsymbol{\mu}_{-1}\|^2}{\mathbf{w}^T\mathbf{S}_1\mathbf{w} + \mathbf{w}^T\mathbf{S}_{-1}\mathbf{w}}$$

combines these two aspects. Maximizing $\|\mathbf{w}^T\boldsymbol{\mu}_1 - \mathbf{w}^T\boldsymbol{\mu}_{-1}\|^2$ ensures good separation between means, and minimizing $\mathbf{w}^T\mathbf{S}_1\mathbf{w} + \mathbf{w}^T\mathbf{S}_{-1}\mathbf{w}$ ensures tight clusters within each class.

So, the goal is to find the vector $\mathbf{w}$ that maximizes the separation of class means while minimizing the dispersion within each class. It's a trade-off between keeping means far apart and keeping points within each class close together. This optimization helps in finding the best direction to project the data for effective classification.

Introduction to ML (CS771), Autumn 2023
Indian Institute of Technology Kanpur
Homework Assignment Number 2

Student Name: Ruby Prajapati
Roll Number: 231110042
Date: November 16, 2023

QUESTION 3

We're dealing with a matrix S, which is computed as $\frac{1}{N}(XX^T)$, where X is a matrix with data points as columns. The problem involves understanding the eigenvectors of this matrix. Given a covariance matrix $S = \frac{1}{N}(XX^T)$, consider an eigenvector v associated with the matrix, satisfying the following equation where the eigenvalue associated with v is λ:

$$Sv = \lambda v$$

Now, substituting the given covariance matrix:

$$\frac{1}{N}(XX^T)v = \lambda v$$

Multiplying both sides by X^T:

$$\frac{1}{N}(X^TX)(X^Tv) = \lambda(X^Tv)$$

Now, substitute $u = X^Tv$, and the equation becomes:

$$\frac{1}{N}(X^TX)u = \lambda u$$

This is a reformulation of the eigenvalue equation using the given covariance matrix. Multiply by X^T:

$$\frac{1}{N}(XX^T)(XX^Tv) = \lambda(XX^Tv)$$

Now, substitute $u = X^Tv$, and the equation becomes:

$$\frac{1}{N}(X^TX)u = \lambda u$$

Substituting $u = X^Tv$, we get the equation in terms of u:

$$\frac{1}{N}(X^TX)u = \lambda u$$

This equation represents the relationship between the covariance matrix and the transformed variable u, related to the original eigenvector v through the transpose of the data matrix X.

Complexity Comparison:

When $D < N$, the time taken for finding Principal Components is:

- For constructing the matrix $X^T X$, this algorithm takes time $O(D^2 N)$.
- Eigenvectors of $X^T X$ can be computed in time $O(D^3)$.

But here, we have given that $D > N$. In this case, the time complexity for finding Principal Components will be:

- For constructing the matrix XX^T, this algorithm takes time $O(N^2 D)$.
- Eigenvectors of XX^T can be computed in time $O(N^3)$.

In summary, the solution explains the relationship between eigenvectors and the given matrix S, introduces a substitution to simplify the expression, and discusses the computational complexity of finding eigenvectors in this specific case.

Introduction to ML (CS771), Autumn 2023
Indian Institute of Technology Kanpur
Homework Assignment Number 2

Student Name: Ruby Prajapati
Roll Number: 231110042
Date: November 16, 2023

QUESTION

4

My solution to problem 4

Part 1

Latent Variable Models for Supervised Learning

Consider learning a regression model given training data $\{(x_n, y_n)\}_{n=1}^N$, with $x_n \in \mathbb{R}^D$ and $y_n \in \mathbb{R}$. Let's introduce a latent variable z_n with each training example (x_n, y_n), where $z_n \in \{1, 2, ..., K\}$ denotes the cluster to which x_n belongs. We assume a multinoulli prior on z_n, i.e., $p(z_n) = \text{multinoulli}(z_n|\pi_1, ..., \pi_K)$.

The model has K weight vectors $W = [w_1, w_2, ..., w_K]$. Given the cluster id z_n, we assume that $p(y_n|z_n, x_n, W) = \mathcal{N}(y_n|w_{z_n}^T x_n, \beta^{-1})$. Note that although there are K weight vectors in the overall model, only one of them is being used to model y_n depending on the value of z_n. Also, the model for the responses y_n is still discriminative, as the inputs are not being modeled.
(1) Explanation: The model associates each training example with a cluster through the latent variable z_n, allowing different weight vectors to be used based on the cluster assignment. This flexibility enables the model to capture diverse patterns in the data, making it more adaptable to complex relationships. Unlike the standard probabilistic linear model with a single weight vector, this approach provides a richer representation, potentially enhancing performance in scenarios where the data exhibits varied structures.

Part (2)

step 1:

The goal is to estimate the posterior distribution of latent variables Z given the observed data and current parameter estimates Θ.

Update latent variables Z:
For each training example n = 1, ..., N, update the cluster assignment zn:

$$p(z_n = k|\mathbf{x}_n, \mathbf{y}_n, \Theta) \propto \pi_k \mathcal{N}(\mathbf{y}_n|\mathbf{w}_k^T \mathbf{x}_n, \beta^{-1}) \tag{1}$$

$$p(z_n = k|\mathbf{x}_n, \mathbf{y}_n, \Theta) = \frac{\pi_k \mathcal{N}(\mathbf{y}_n|\mathbf{w}_k^T \mathbf{x}_n, \beta^{-1})}{\sum_{j=1}^K \pi_j \mathcal{N}(\mathbf{y}_n|\mathbf{w}_j^T \mathbf{x}_n, \beta^{-1})} \tag{2}$$

step 2:

The goal is to update the parameters Θ based on the observed data and the current estimates of latent variables Z.

Update for w_k:

$$w_k = \left(\sum_{n=1}^{N} \mathbb{I}(z_n = k) x_n x_n^T \right)^{-1} \left(\sum_{n=1}^{N} \mathbb{I}(z_n = k) y_n x_n \right)$$

Update for Parameters:

$$N_k = \sum_{n=1}^{N} z_{nk} \tag{3}$$

$$w_k = (X_k^T X_k)^{-1} X_k^T y_k \tag{4}$$

$$\pi_k = \frac{N_k}{N} \tag{5}$$

Special case when $\pi_k = 1/K$ for all k:

In this case, the update equation for each z_n becomes:

$$z_n = \arg\max_{z_n} \exp\left\{-\beta^2 (y_n - \mathbf{w}_{z_n}^T \mathbf{x}_n)^2\right\} \Big/ \sum_{l=1}^{K} \exp\left\{-\beta^2 (y_n - \mathbf{w}_l^T \mathbf{x}_n)^2\right\}$$

This is like Softmax classification.This update is equivalent to multi-output logistic regression.

Introduction to ML (CS771), Autumn 2023
Indian Institute of Technology Kanpur
Homework Assignment Number 2

Student Name: Ruby Prajapati
Roll Number: 231110042
Date: November 16, 2023

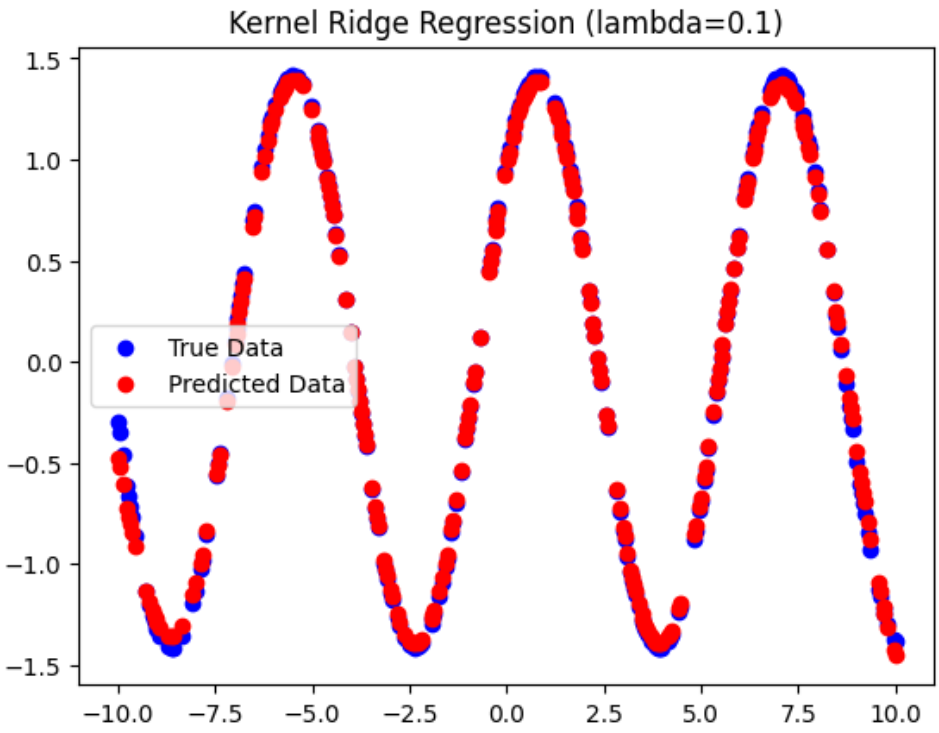
QUESTION 5

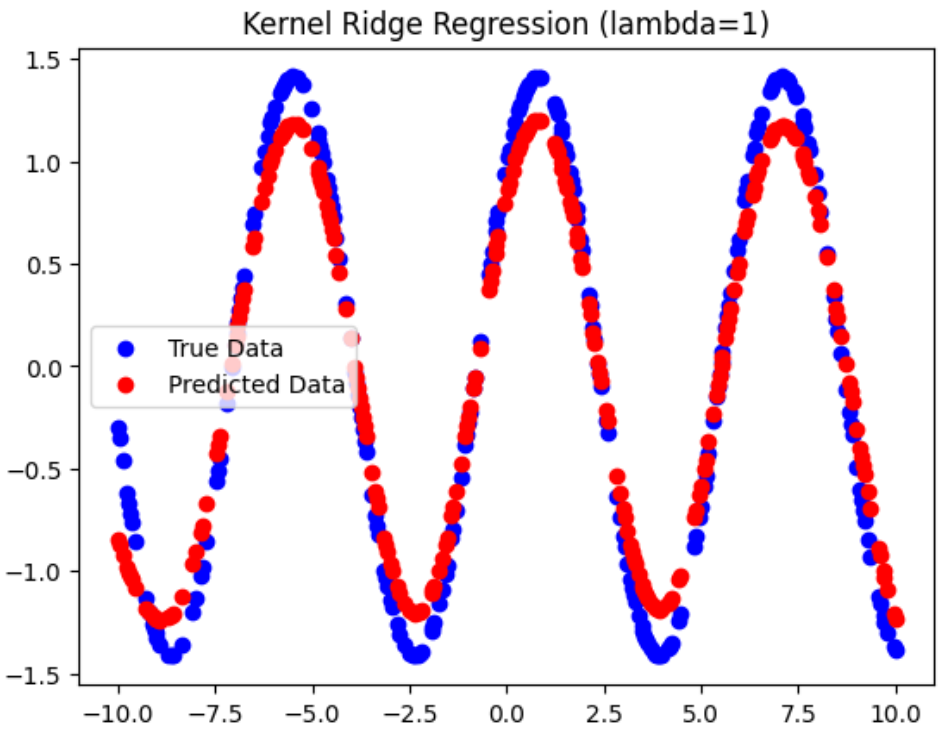
My solution to problem 5:

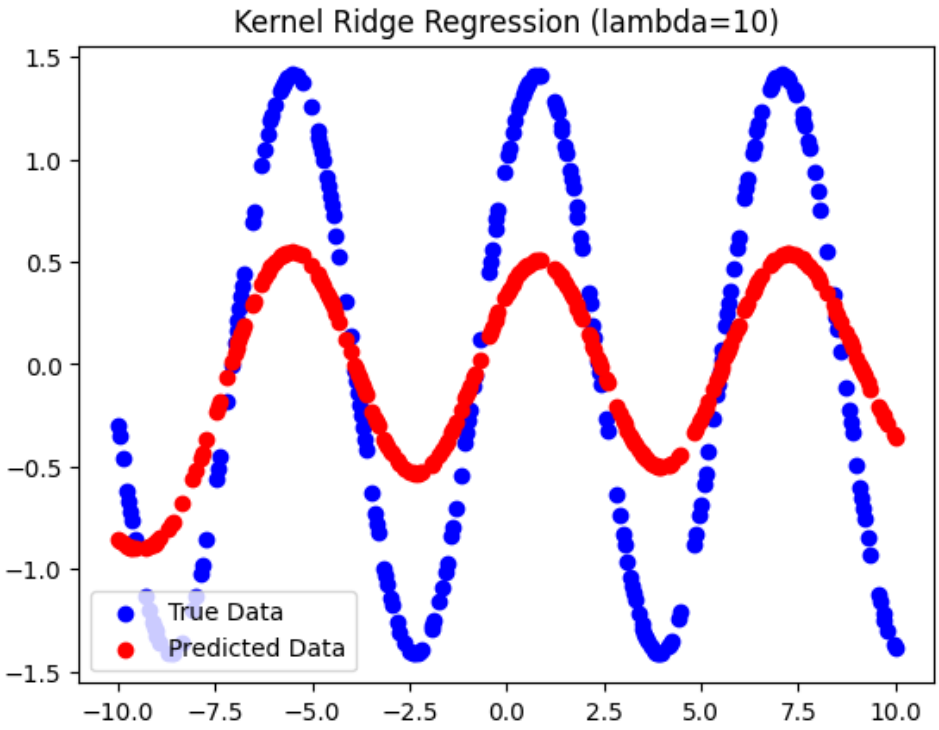
Part 1 :

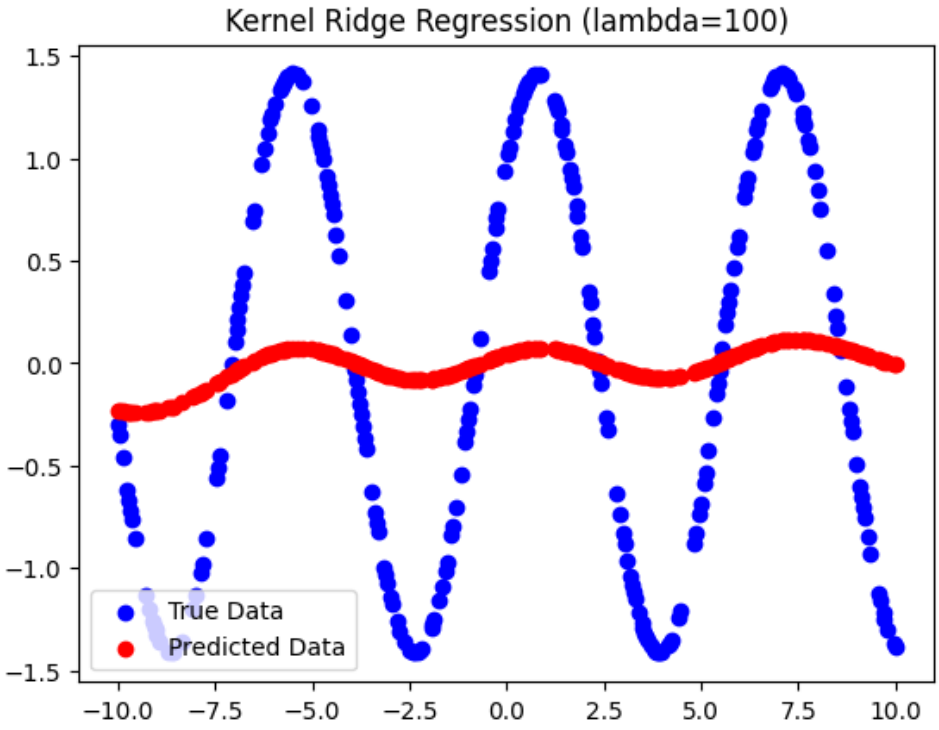
1.1

We have to implement kernel ridge regression in this part of question, train the model with the regularization hyper parameter lambda = 0.1 and then use the learned model to predict the outputs for the test data Compare the model's predictions with the true test outputs by plotting on a graph. and we have to repeat this exercise for lambda = 1, 10, 100. and also calculated root mean square.Following are the outputs of the implementation.




Kernel Ridge Regression (lambda=1)
True Data
Predicted Data
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0


Kernel Ridge Regression (lambda=10)
True Data
Predicted Data
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0



Observation :

After the implementation we are getting this output, After the implementation, we are getting this output:

$$\lambda = 0.1 : \text{RMSE} = 0.0326$$

$$\lambda = 1 : \text{RMSE} = 0.1703$$

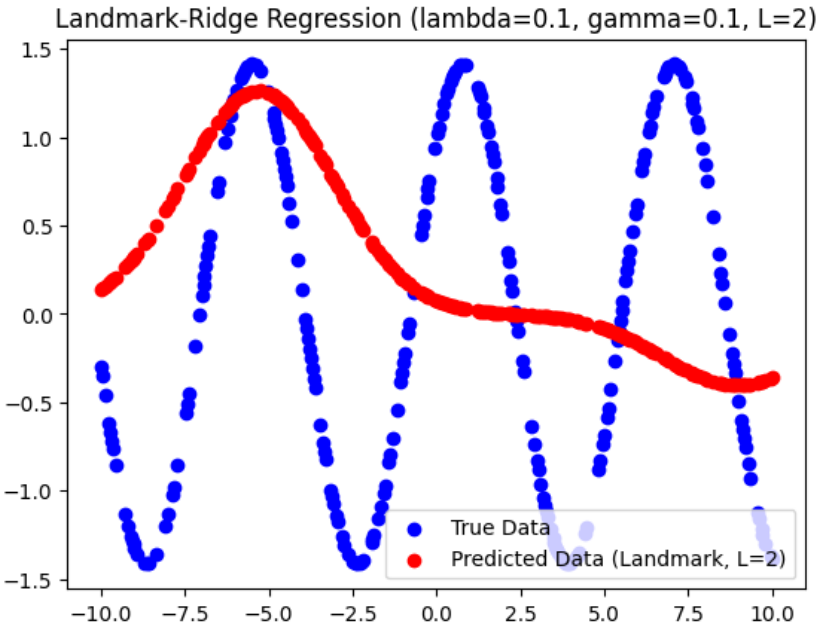
$$\lambda = 10 : \text{RMSE} = 0.6093$$

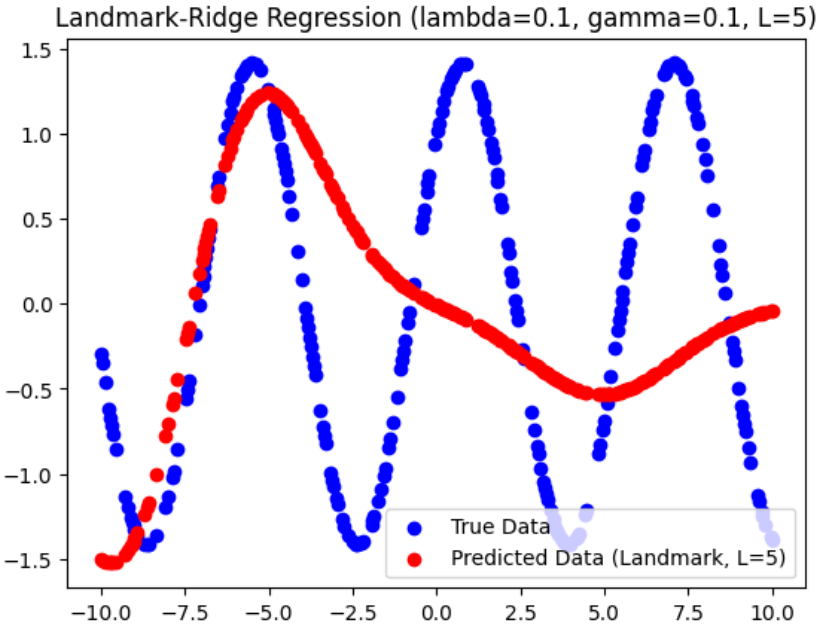
$$\lambda = 100 : \text{RMSE} = 0.9111$$

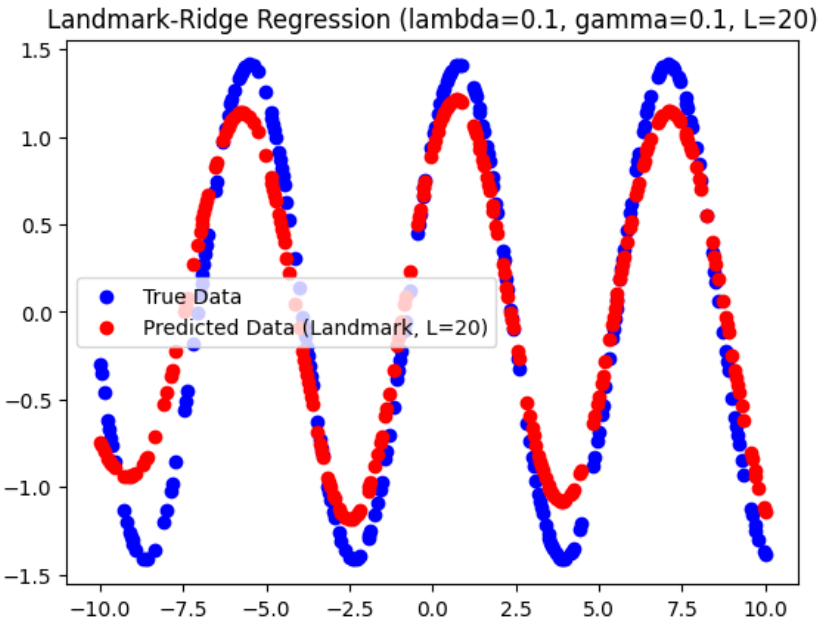
Easily, we can observe that as we are increasing the value of λ, the RMSE error at test time is increasing. Increasing the regularization parameter λ strengthens the penalty on model complexity, promoting simplicity. As λ rises, the model becomes more constrained, leading to underfitting and a higher root mean square error (RMSE). This indicates the model struggles to capture underlying patterns. It underscores the importance of choosing an optimal λ through techniques like cross-validation, striking a balance between fitting the training data and preventing overfitting. The observed trend suggests the current λ values may be too high for effective model performance. Adjustments and experimentation with different λ values are advisable to find the optimal regularization strength.

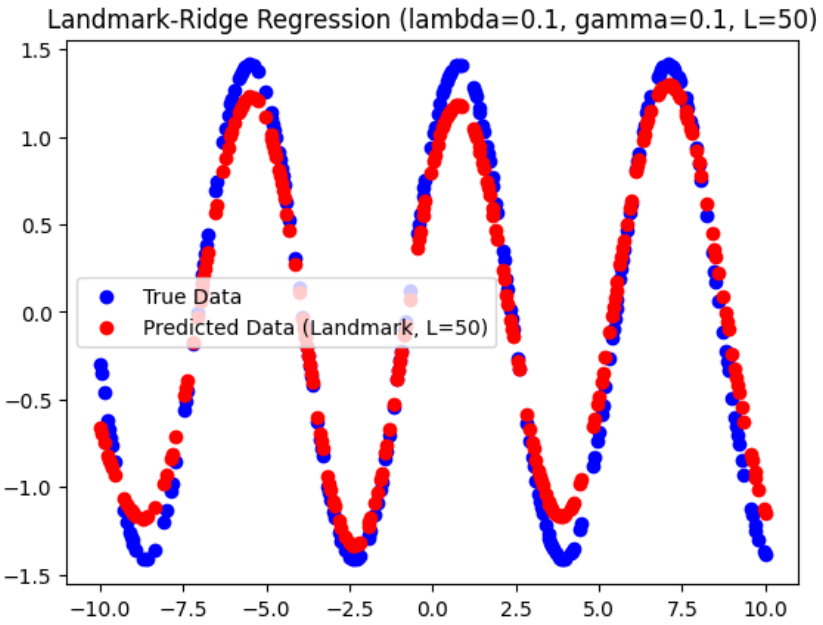
1.2

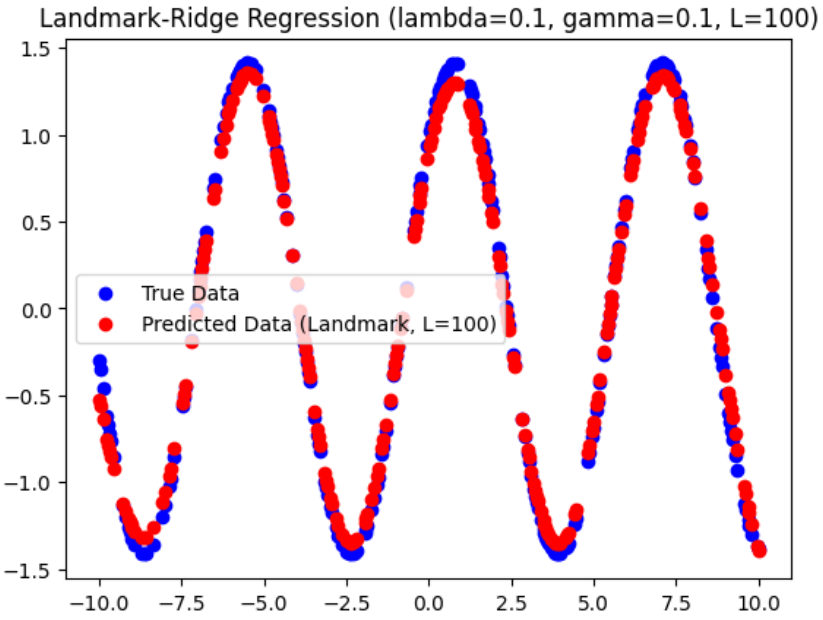
This Python code performs Landmark-Ridge Regression, a technique where we use randomly selected data points as landmarks to make predictions. It uses a mathematical function to measure the similarity between data points. The code tests different numbers of landmarks (L) and visualizes the predictions against true values for each case. The parameters lambda and gamma are set to control the model's behavior. By varying the number of landmarks, the code helps us see how this choice affects the accuracy of predictions in Ridge Regression. The plots show the impact of landmarks on the model's performance.






Landmark-Ridge Regression (lambda=0.1, gamma=0.1, L=20)
True Data
Predicted Data (Landmark, L=20)
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0





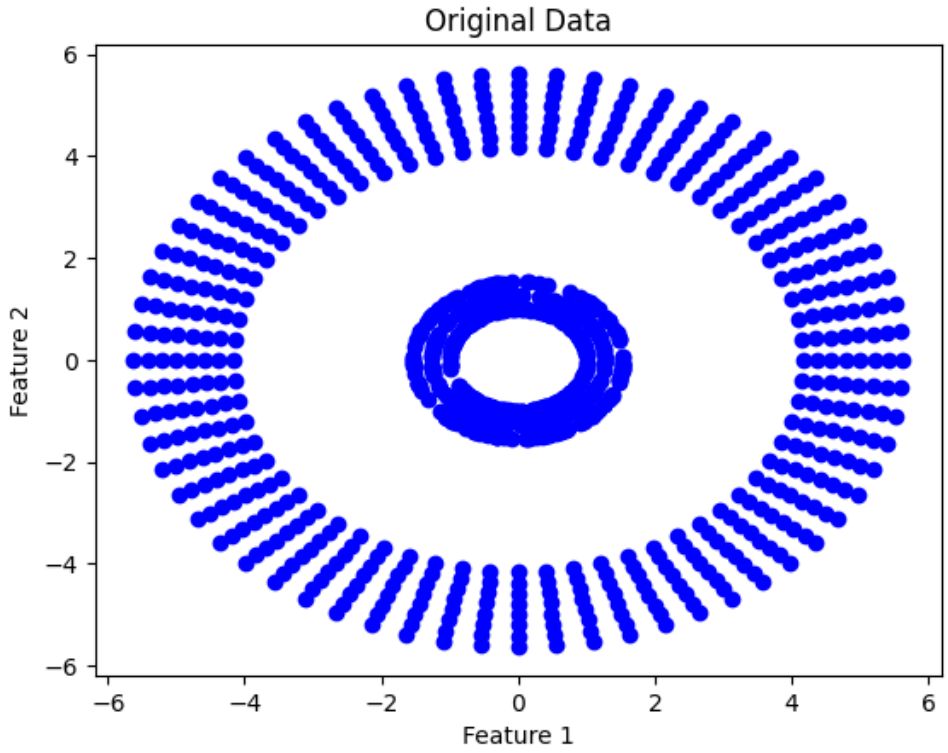
Observation :

As we are value of L this is predicting correct output , as the number of landmarks (L) increases, the Landmark-Ridge Regression tends to improve its predictions on the test data. This behavior is due to the increased flexibility and expressiveness of the model as more landmarks are introduced. Each landmark contributes to capturing different patterns in the data, allowing the model to adapt better to complex relationships. However, it's important to note that while increasing L can enhance performance, there's a trade-off, and too many landmarks may lead to over fitting. Optimal performance is often found through experimentation and validation.

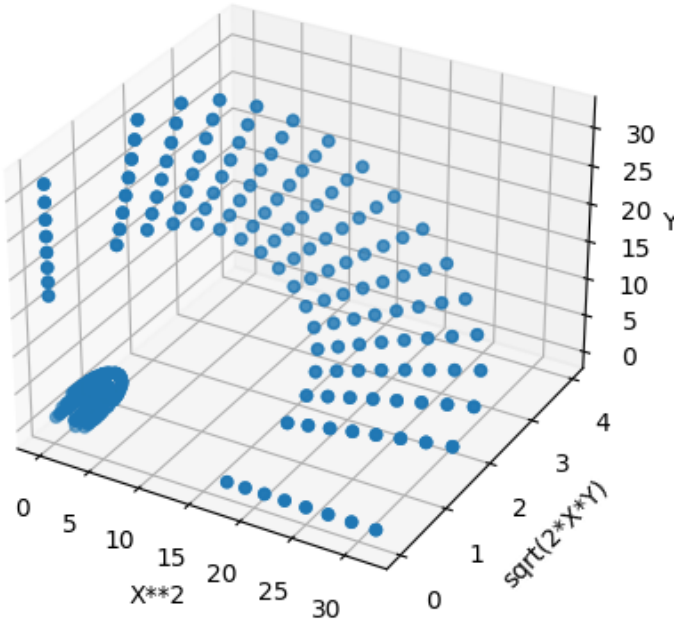
Part 2

2.1

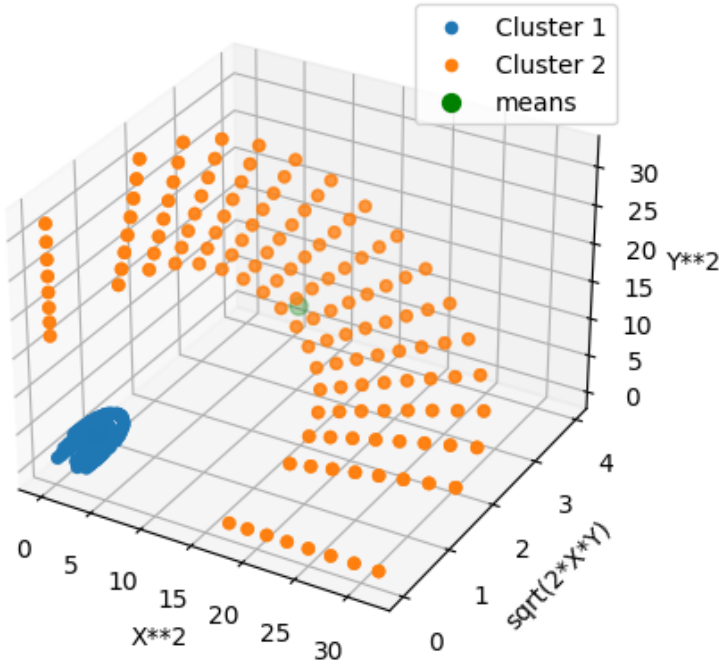
This Python code first visualizes 2D data and transforms it into a 3D feature space. It then performs K-Means clustering in the transformed space with two clusters. The algorithm iteratively assigns data points to the nearest centroid and updates the centroids until convergence. The code visualizes the clustered data in both 3D and 2D, highlighting the two clusters with distinct colors. Additionally, it shows the initial centroids in the 3D plot. Overall, the code demonstrates the process of K-Means clustering and visually presents the partitioning of the data into clusters along with the initial centroids.

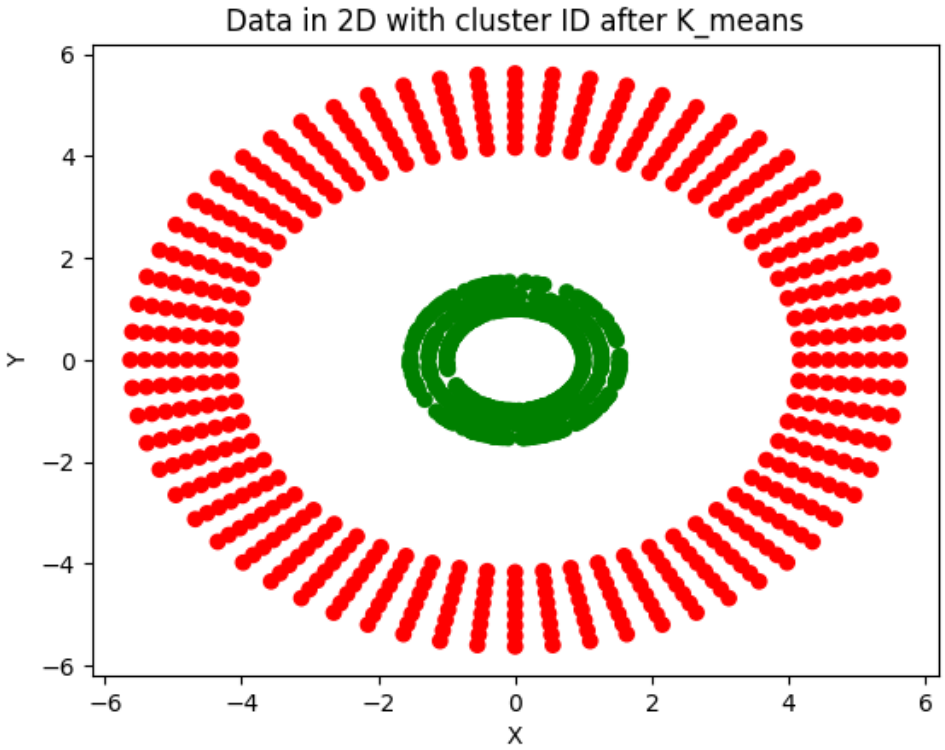


3D Data Plot without clustering



K-Means Clustering in 3D with 2 Clusters (Initialized Centroids)





Observations:

2D Original Data Plot:

- The initial plot of the original 2D data shows the distribution of points in the feature space.

3D Transformed Data Plot:

- After transforming the data into a 3D space, the plot illustrates the new feature space with axes corresponding to X^2, $\sqrt{2 \cdot X \cdot Y}$, and Y^2.

K-Means Clustering in 3D:

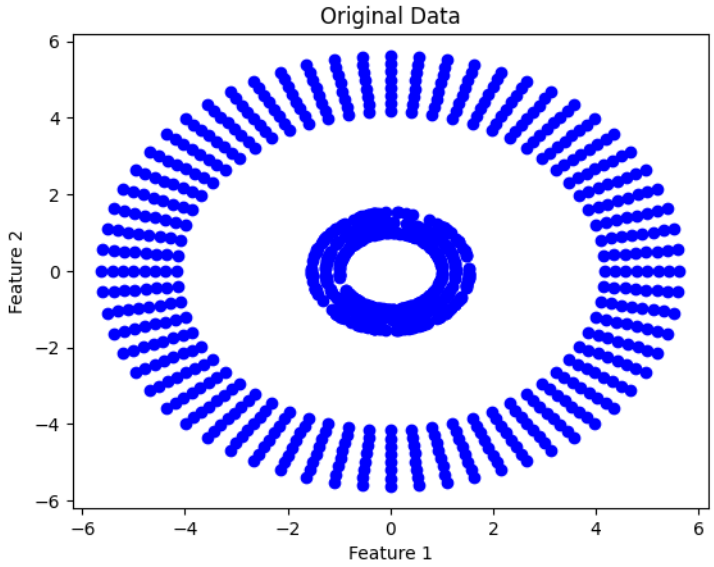
- The 3D plot after K-Means clustering with two clusters reveals how the algorithm partitions the data. Each cluster is assigned a distinct color, and the green markers represent the cluster centroids.

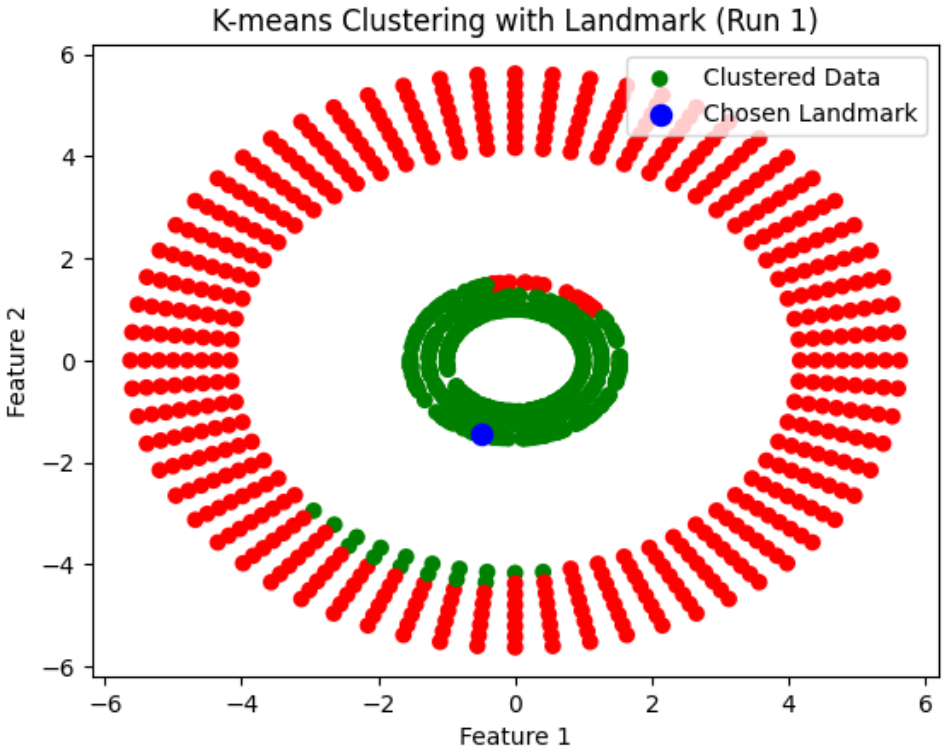
2D Data Plot with Cluster ID:

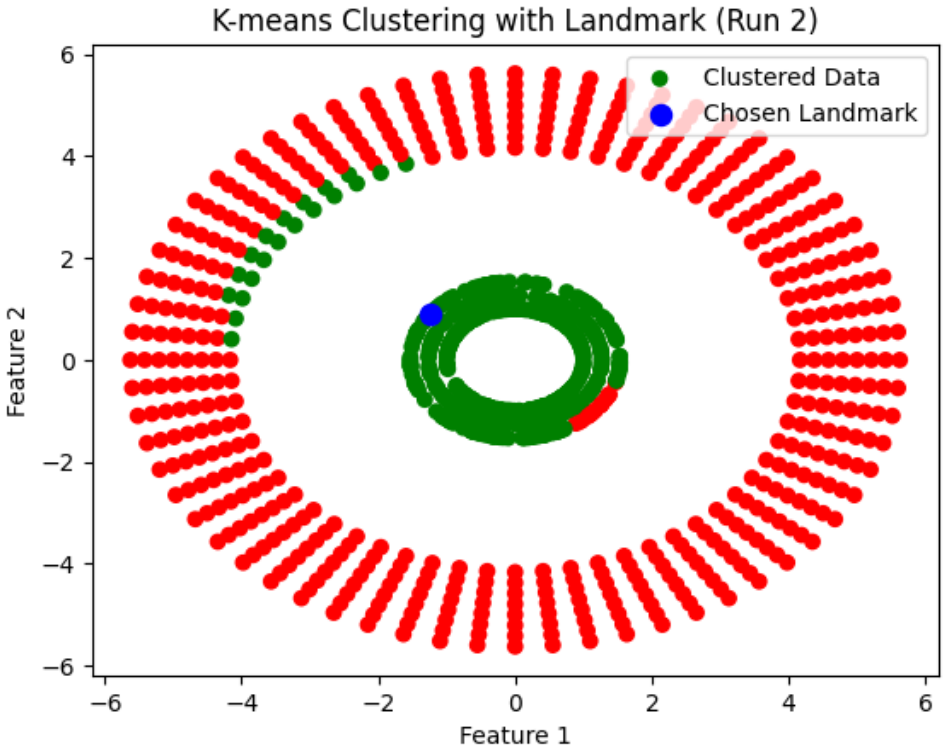
- The final 2D plot depicts the original data points colored according to their assigned clusters after K-Means clustering.

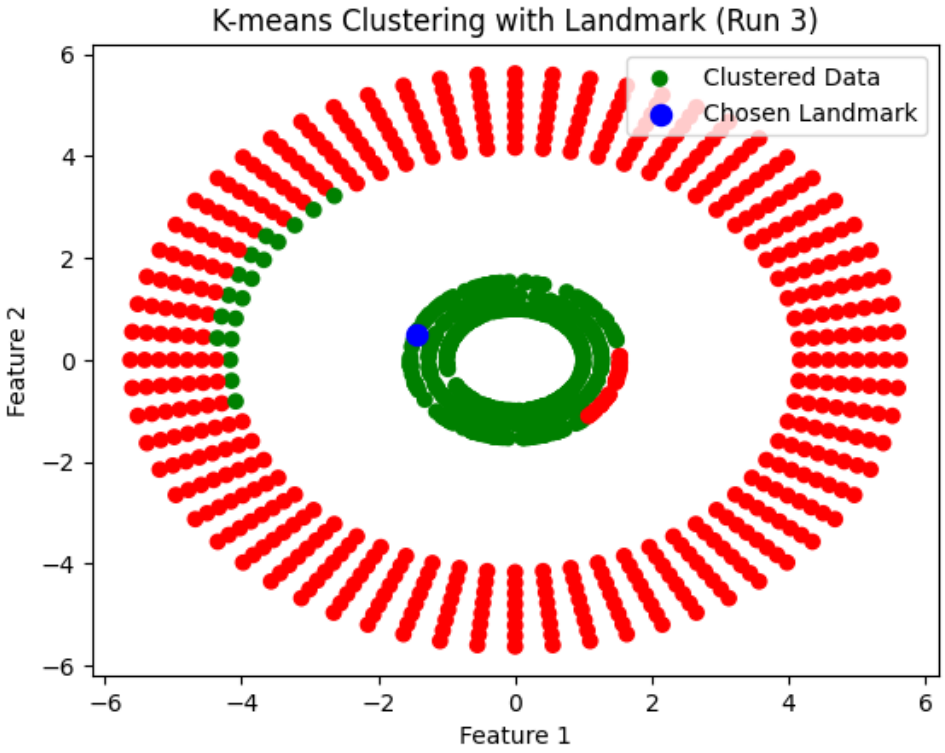
2.2

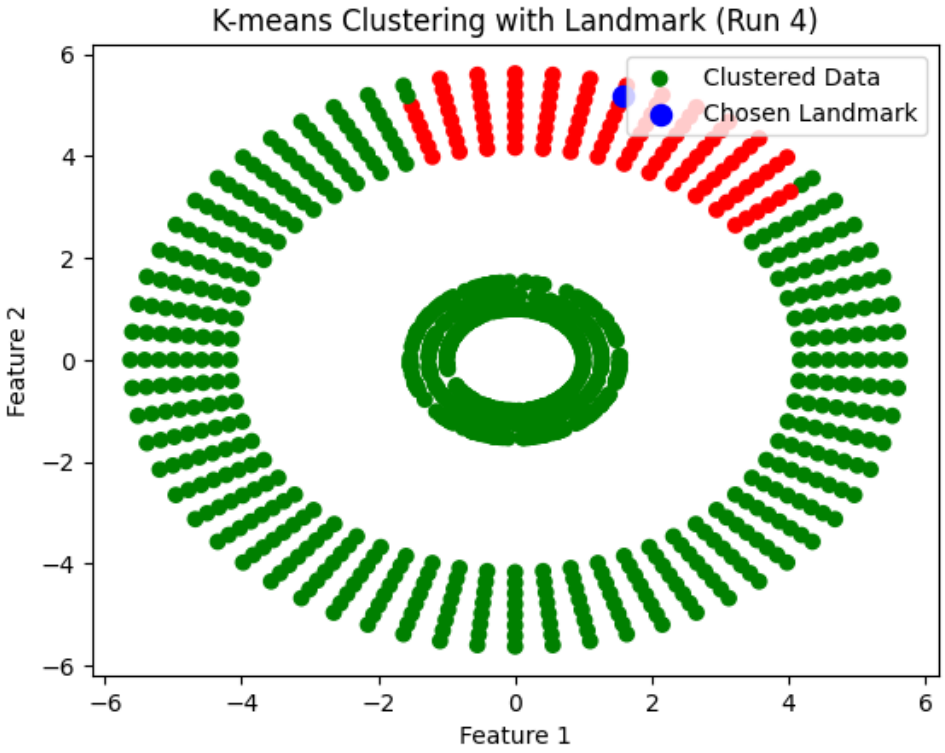
This Python code implements K-Means clustering with a randomly chosen landmark for multiple runs. In each run, it selects a random data point as a landmark, computes a Gaussian kernel for each data point based on the landmark, and applies K-Means clustering in the transformed space. The code visualizes the clustered data points in 2D, with different colors indicating distinct clusters. This process is repeated for multiple runs, providing insights into the variability of clustering results with different randomly chosen landmarks. The final plots illustrate the clustered data and the chosen landmark in each run.

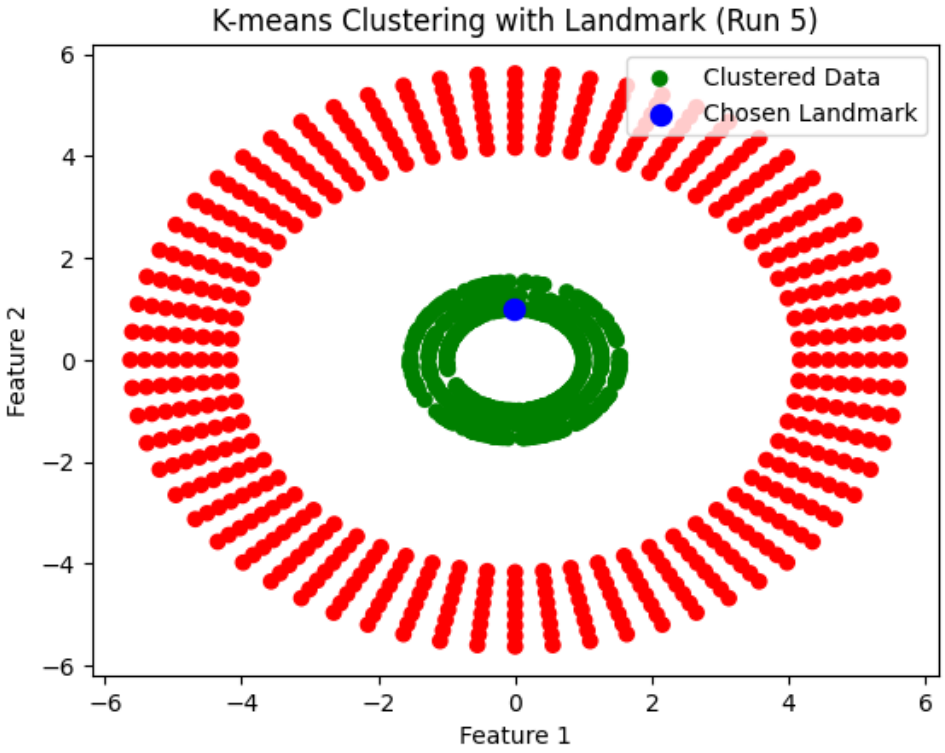


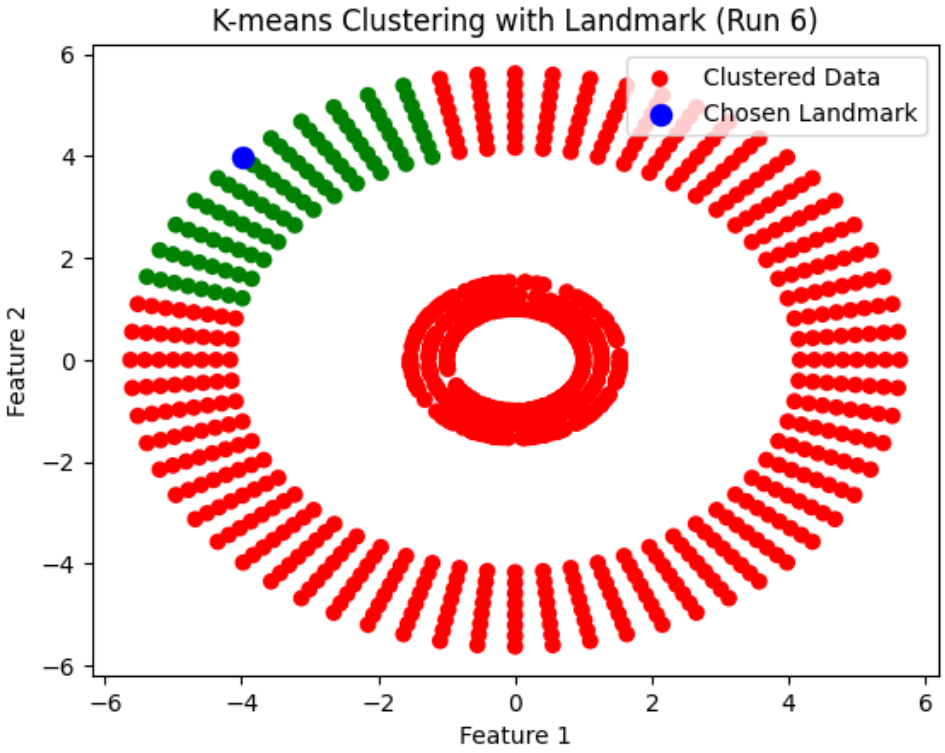


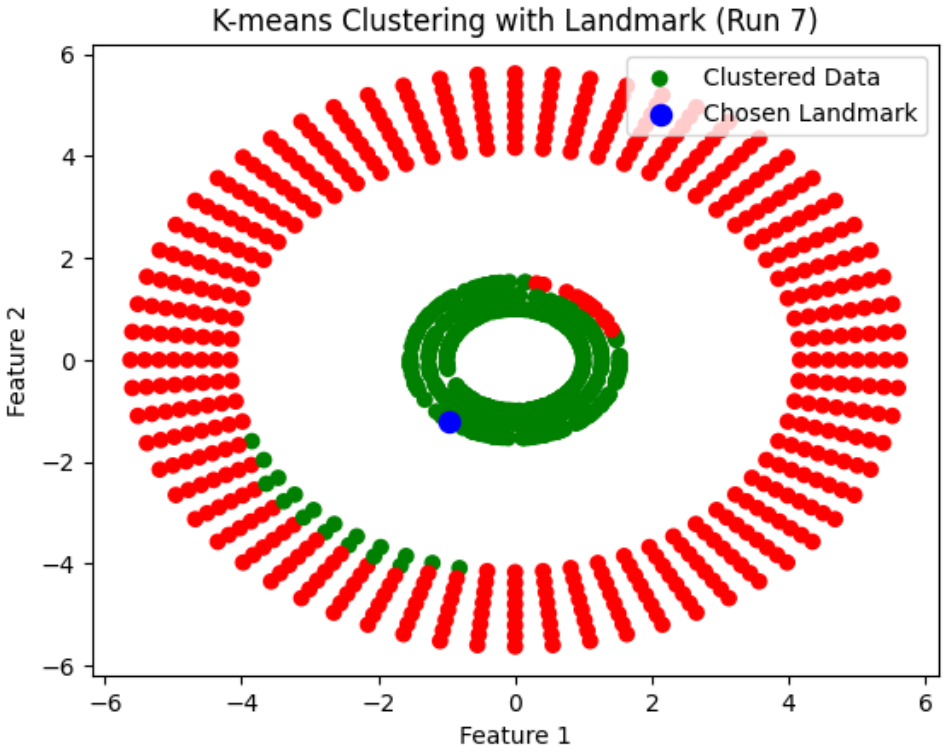

K-means Clustering with Landmark (Run 2)
Clustered Data
Chosen Landmark
Feature 2
Feature 1
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6

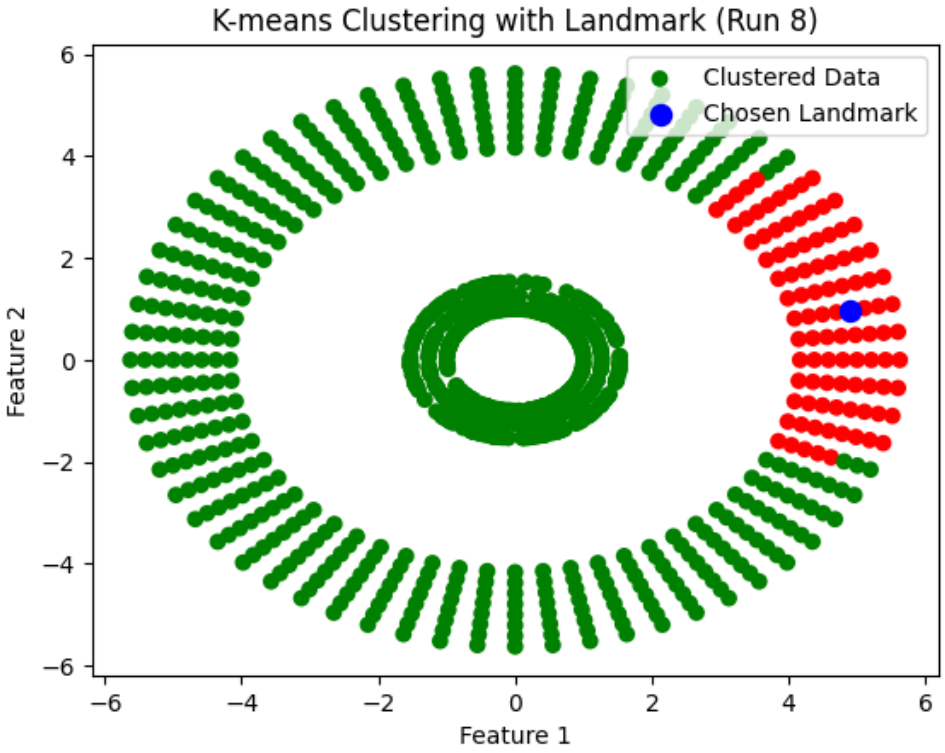

K-means Clustering with Landmark (Run 3)
Clustered Data
Chosen Landmark
Feature 2
Feature 1
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6

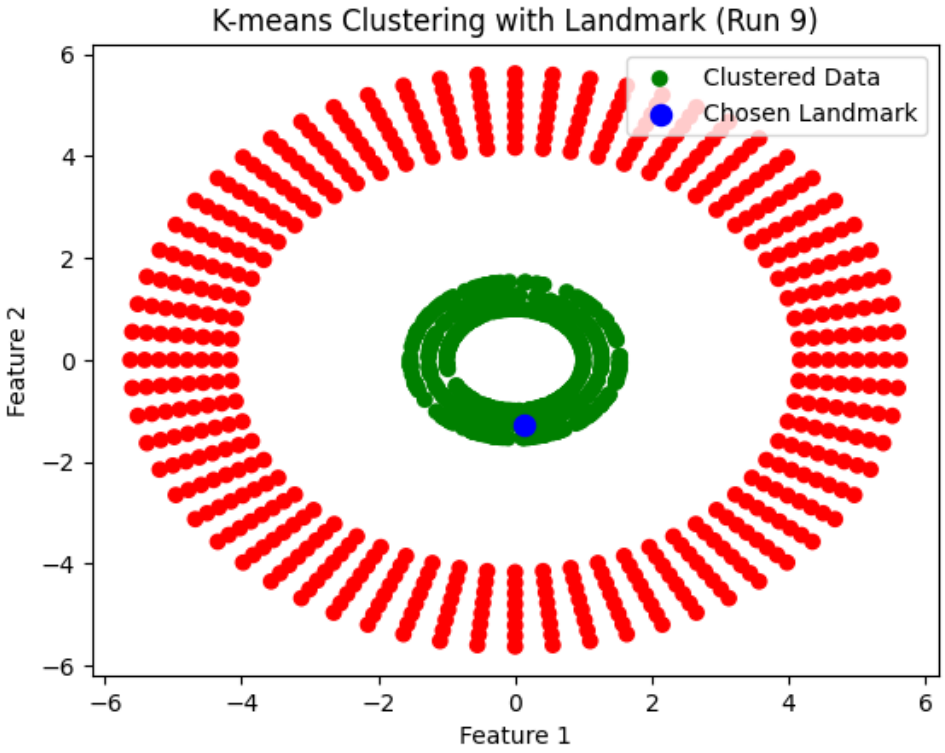

K-means Clustering with Landmark (Run 4)
Clustered Data
Chosen Landmark
Feature 2
Feature 1
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6

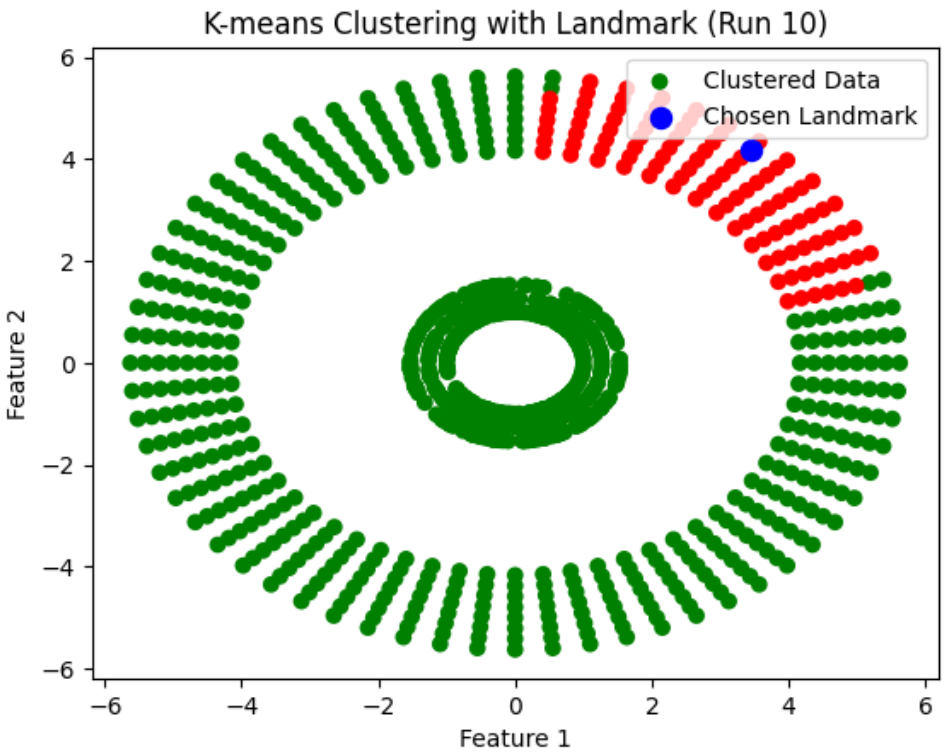

K-means Clustering with Landmark (Run 5)
Clustered Data
Chosen Landmark
Feature 2
Feature 1
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6


K-means Clustering with Landmark (Run 6)
Clustered Data
Chosen Landmark
Feature 2
Feature 1
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6


K-means Clustering with Landmark (Run 7)
Clustered Data
Chosen Landmark
Feature 2
Feature 1
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6


K-means Clustering with Landmark (Run 8)
Clustered Data
Chosen Landmark
Feature 2
Feature 1
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6


K-means Clustering with Landmark (Run 9)
Clustered Data
Chosen Landmark
Feature 2
Feature 1
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6



Observation :

The clustering results vary with different randomly chosen landmarks, illustrating the sensitivity of K-Means to the initial selection. The choice of landmark influences the shape and boundaries of the clusters. Multiple runs highlight the variability in clustering outcomes, providing insights into the impact of landmark selection on the K-Means algorithm.

Part 3 :

This Python code performs dimensionality reduction and visualization on the MNIST dataset. It loads the dataset containing images of digits and their corresponding labels. First, it uses Principal Component Analysis (PCA) to reduce the data to two dimensions and visualizes the digit clusters in a scatter plot. Then, it applies t-Distributed Stochastic Neighbor Embedding (t-SNE) for another two-dimensional projection, offering an alternative visualization. Each digit class is represented by a distinct color, providing insights into the spatial distribution and relationships between digits in the reduced-dimensional space. The code helps explore and understand the structure of the high-dimensional MNIST dataset.

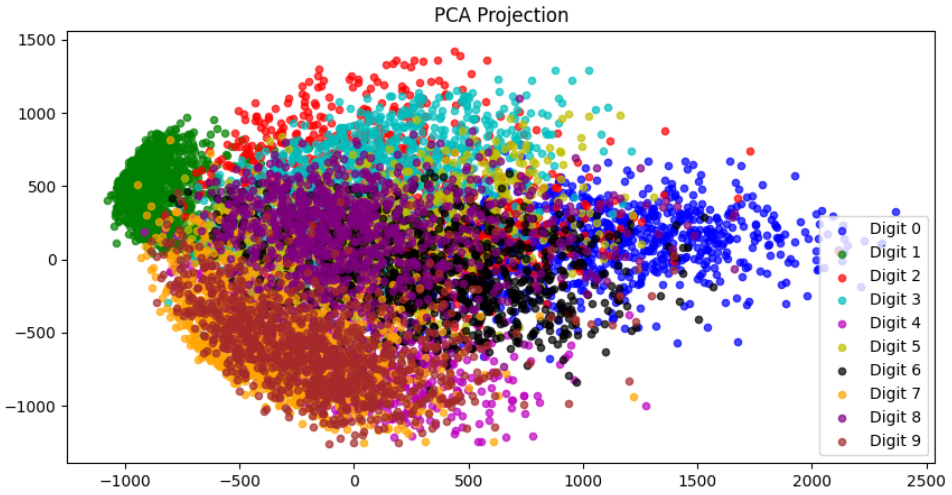


Figure 1: PCA

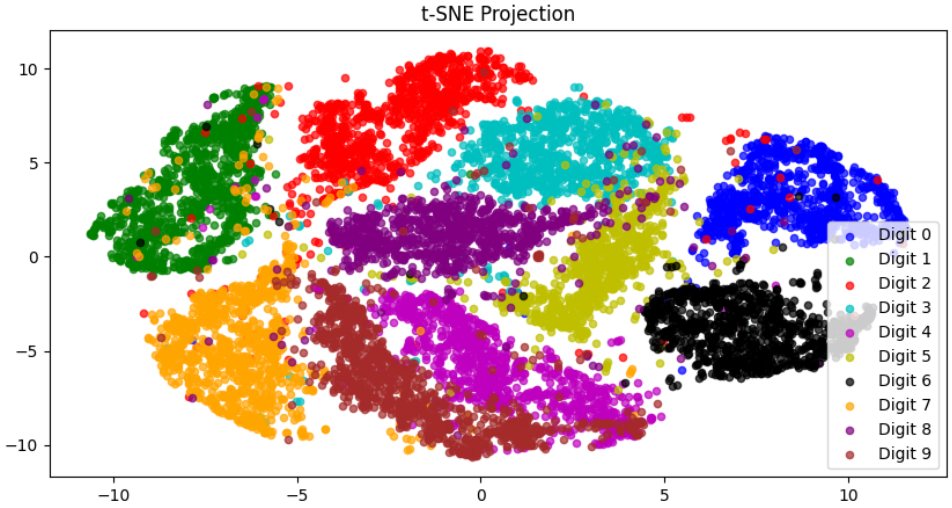


Figure 2: T-SNE

Observation :

PCA Projection:

The PCA projection visualizes the MNIST digits in a two-dimensional space. Digits with similar shapes are grouped together, reflecting structural similarities. Some overlap between digit clusters is observed.

t-SNE Projection:

t-SNE provides an alternative projection, emphasizing local relationships between data points. Digit clusters are well-separated, revealing more intricate structures and reducing overlap. Each digit class forms a distinct cluster, demonstrating the effectiveness of t-SNE in preserving local similarities.