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

Student Name: Om Singh
Roll Number: 210686
Date: November 17, 2023

QUESTION 1

Setting up the problem and notations:

We know that the loss (or objective) function that the K-means clustering algorithm optimises is given by:

$$\mathcal{L} = \sum_{n=1}^{N} \sum_{k=1}^{K} z_{nk} ||\mathbf{x}_n - \mu_{\mathbf{k}}||^2 \tag{1}$$

The only difference between the standard Lloyd's algorithm and in the suggested online algorithm is that in the suggested algorithm, we want to carry out stochastic gradient descent to update μ vectors, and that $\mathbf{x}_n$ is assigned to some cluster mean greedily.

The following notations have been used henceforth in the problem:

- The n^{th} training example is denoted by $\mathbf{x}_n$.
- At the end of iteration t, the i^{th} cluster mean shall be denoted by $\mu_i^{(t)}$.

Step-1: Choosing x_n randomly and assigning cluster to it greedily:

To perform this Step-1 of this online algorithm, do the following in every iteration (t), until convergence or until t_{max} is encountered:

1. Pick up one of the $\mathbf{x}_n's$ uniformly randomly.
2. Calculate the distance of this $\mathbf{x}_n$ from all the cluster means as found at the end of the $(t-1)^{th}$ iteration, and assign $\mathbf{x}_n$ to the cluster whose mean is the closest to $\mathbf{x}_n$ in terms of Euclidean distance.

   Mathematically,

$$\boxed{z_{nk'} = 1 \implies \|\mathbf{x}_n - \boldsymbol{\mu}_{k'}^{(t-1)}\| \leq \|\mathbf{x}_n - \boldsymbol{\mu}_{k}^{(t-1)}\| \quad \forall k \in \{1, 2, \ldots, K\}} \tag{2}$$

This is an example of a **greedy algorithm**, that makes only local decisions, and greedily assigns the randomly chosen $\mathbf{x}_n$ to the best possible cluster mean, which minimises the loss function in the current t^{th} iteration.

Step-2: Stochastic GD-based update for the cluster mean and intuition behind it:

Mathematical formulation:

Initially, we can initialise all the weights randomly or whatever we desire them to be. For Stochastic Gradient Descent, we need to calculate the partial derivative of the loss function $\mathcal{L}$ with respect to an arbitrary cluster mean $\mu_{\mathbf{k}}$, as our next step.

Taking the derivative, we get:

$$\frac{\partial \mathcal{L}}{\partial \mu_{\mathbf{k}}} = 2 \sum_{n=1}^{N} z_{nk} (\mu_{\mathbf{k}} - \mathbf{x}_n) \tag{3}$$

In the above case, we have absorbed the extra '-' sign inside the summation itself. So, the updates shall be the following:

$$\mu_{\mathbf{k}}^{(\mathbf{t})} = \mu_{\mathbf{k}}^{(\mathbf{t-1})} - 2\eta \cdot z_{nk} \cdot (\mu_{\mathbf{k}}^{(\mathbf{t-1})} - \mathbf{x}_n) \tag{4}$$

But, we know that z_{nk} is non-zero only for $k = j$, where j is the cluster ID assigned to $\mathbf{x}_n$ in step-1 of our online algorithm.

So, we only get a single update equation as:

$$\boxed{\mu_{\mathbf{j}}^{(\mathbf{t})} = \mu_{\mathbf{j}}^{(\mathbf{t-1})} - 2\eta \cdot (\mu_{\mathbf{j}}^{(\mathbf{t-1})} - \mathbf{x}_n)}, s.t. \|\mathbf{x}_n - \boldsymbol{\mu}_j^{(t-1)}\| \leq \|\mathbf{x}_n - \boldsymbol{\mu}_k^{(t-1)}\| \quad \forall k \in \{1, 2, \ldots, K\} \tag{5}$$

Intuition:

We see that this update is exactly what we desire!

It is because intuitively, if $\mu_{\mathbf{j}}^{(\mathbf{t-1})}$ is larger than $\mathbf{x}_n$, we have $\mu_{\mathbf{j}}^{(\mathbf{t-1})} - \mathbf{x}_n > 0$ and hence, overall $\mu_{\mathbf{j}}^{(\mathbf{t})}$ decreases. Similarly, if this difference is negative, the overall value increases for the assigned cluster mean. If it is exactly equal to 0, there is no change.

All these cases tend to minimise the distance between the assigned cluster mean and the input $\mathbf{x}_n$. So overall, in each step, the square of the Euclidean distance between the assigned cluster mean and the training input decreases, and everything else remains the same. Hence, overall, the loss function decreases, and hence gets optimised, which is exactly what we desired!

A great choice of learning rate / step size:

Just taking a look at the update equation, we can see that

$$\boxed{\eta = \frac{1}{2}} \tag{6}$$

works as a great choice for the learning rate / step size, despite being a constant step size. For this choice, the update equation assumes a very simple form:

$$\mu_{\mathbf{j}}^{(\mathbf{t})} = \mathbf{x}_n$$

. So, this choice of η is extremely efficient computationally, as we just need to make the closest cluster mean to $\mathbf{x}_n$ to be $\mathbf{x}_n$ itself.

Summary of the entire algorithm:

1. Initialise K-cluster means randomly.
2. Pick one training example at a time uniformly randomly. Compute the Euclidean distance of this training example from all the K-cluster means.
3. Pick up the cluster mean which is closest to this training example. Update its value equal to the training example itself.
4. Move to (1) again until convergence or until a sufficient number of iterations have taken place, if employing dropout method.

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

Student Name: Om Singh
Roll Number: 210686
Date: November 17, 2023

QUESTION 2

Setting up the problem:

We are given labelled training data: $\{\mathbf{x}_n, y_n\}$, where $y_n \epsilon \{-1, +1\}$, where $\mathbf{x}_n \epsilon \mathbb{R}^D$. We need to determine a weight vector $\mathbf{w} \in \mathbb{R}^D$ such that is we project all the $\mathbf{x}_n's$ on this weight vector, the following are achieved:

- The distance between the means of the inputs from the two classes after projection becomes as large possible.
- The inputs within each class after projection become as close to each other as possible.

Methodology:

Consider the input vector $\mathbf{x}_n$. Now, points along $\mathbf{w}$ can be represented just using a scalar number, as the direction is implicit from the definition of the weight vector.

So, the projection of $\mathbf{x}_n$ along $\mathbf{w}$ can be written to be $\mathbf{w}^T\mathbf{x}_n$, which is a scalar. Now, consider what happens to the mean of all the inputs belonging to a particular class. Say the mean before projections was μ, and the new mean becomes μ'. Then,

$$\mu' = \frac{1}{N_c} \sum_{x_n : x_n \in c} w^T x_n = w^T \frac{1}{N_c} \sum_{x_n : x_n \in c} x_n = w^T \mu,$$

utilising the distributive property of matrices, and the definition of μ.

Now, as we ant to maximise the distance between the means of the two classes after employing the projection operation, we need to maximise $\mathbf{w}^T(\mu_+ - \mu_-)$. Simultaneously, we also need to minimise the inter-class variance. We also need a regularisation parameter, which shall serve as a model hyperparameter in this case, to show the extent of trade-off between the two!

Consider the following loss function:

$$\boxed{\mathcal{L}(\mathbf{w}) = \frac{1}{N_+} \sum_{\mathbf{x}_n : y_n = +1} (\mathbf{w}^T(\mathbf{x}_n - \mu_+))^2 + \frac{1}{N_-} \sum_{\mathbf{x}_n : y_n = -1} (\mathbf{w}^T(\mathbf{x}_n - \mu_-))^2 - \lambda(\mathbf{w}^T(\mu_+ - \mu_-))^2} \tag{7}$$

Here, μ_+ and μ_- are the **initial** mean vectors of the class of inputs having their labels as +1 and -1 respectively. Here, the loss function needs to be minimised with respect to the weight vector $\mathbf{w}$.

Conclusion (assuming optimisation using Gradient Descent Algorithm to Stochasic Gradient Descent Method):

So, the basic idea is to compute and store the vectors μ_+ and μ_- to start with. Then, at each stage, compute the gradient of the aforementioned loss function **with respect to the weight vector w** by either considering all the $\mathbf{x}_n's$ or by choosing a random example (stochastically) at each stage, and use the Gradient Descent or the Stochastic Gradient Method in order to minimise the loss function as mentioned above.

The hyperparameter λ can be determined through cross-validation. An appropriate initialisation of our weight vector $\mathbf{w}$ to start with is crucial too, as the loss function might be non-convex.

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

Student Name: Om Singh
Roll Number: 210686
Date: November 17, 2023

QUESTION 3

Mathematical formulation of the problem:

We have a $N \times D$ matrix $\mathbf{X}$, with $D > N$. We are given an eigenvector $\mathbf{v} \in \mathbb{R}^N$ of the matrix $\frac{1}{N}\mathbf{X}\mathbf{X}^T$, with some eigenvalue, say λ.

We want to use this information to find an eigenvector of the covariance matrix $\mathbf{S}$, which is defined as $\mathbf{S} = \frac{1}{N}\mathbf{X}^T\mathbf{X}$, in the Principal Component Analysis (PCA) algorithm.

Solving the problem:

Say the eigenvalue corresponding to $\mathbf{v}$ is λ. Then, we have:

$$\frac{1}{N}\mathbf{X}\mathbf{X}^T\mathbf{v} = \lambda\mathbf{v}$$

Multiplying both sides by $\mathbf{X}^T$, we get:

$$\frac{1}{N}\mathbf{X}^T\mathbf{X}\mathbf{X}^T\mathbf{v} = \lambda\mathbf{X}^T\mathbf{v}$$

. But we know that $\mathbf{X}^T\mathbf{X} = N\mathbf{S}$. So, we have:

$$\boxed{\mathbf{S}X^T\mathbf{v} = \lambda\mathbf{X}^T\mathbf{v}}. \tag{8}$$

Interestingly, this is an eigenvalue equation for the covariance matrix $\mathbf{S}$. So, for an eigenvector of the original covariance matrix, we can choose $\mathbf{u} = \mathbf{X}^T\mathbf{v}$. Also, the eigenvalue is still λ, and remains unchanged.

Concluding Remarks:

Therefore, if we know any eigenvector of the matrix $\frac{1}{N}\mathbf{X}\mathbf{X}^T$ having a particular eigenvalue, we can easily figure out an eigenvector having the same eigenvalue for the covariance matrix $\mathbf{S}$.

But, the matrix $\frac{1}{N}\mathbf{X}\mathbf{X}^T$ is a $N \times N$ matrix, and the covariance matrix $\mathbf{S}$ is a $D \times D$ matrix. As $D > N$, it is easier to find out the eigenvectors corresponding to some eigenvalue for the matrix $\frac{1}{N}\mathbf{X}\mathbf{X}^T$.

Advantages summary:

The advantage of this method over the traditional method of directly computing the eigenvectors of the covariance matrix works only in the case when $D < N$. In that case, finding eigenvectors $\mathbf{v} \in \mathbb{R}^N$ is much simpler computationally than finding out the eigenvectors for the covariance matrix. Further, once found, by doing a simple matrix multiplication of that $\mathbf{v}$ with $\mathbf{X}^T$, we end up getting the required eigenvector of the covariance matrix with the same eigenvalue, which is what we had set out to find!

So, computations (in terms of time complexity) become a lot more simpler, if $D >> N$. Also, it is more memory efficient too, as storing a $D \times D$ matrix is much more efficient in this case, compared to the $N \times N$ covariance matrix.

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

Student Name: Om Singh
Roll Number: 210686
Date: November 17, 2023

QUESTION 4

Basic formulation of the suggested model:

In the modified probabilistic linear regression model as suggested in the problem, each of the training data $\{\mathbf{x}_n, y_n\}$ has a latent variable $z_n \in \{1, 2, ..., K\}$ associated with it, denoting the cluster ID to which $\mathbf{x}_n$ belong to. Also, we assume a multinoulli prior on z_n, having the parameters π_1 through π_K.

Also, we assume K different weight vectors in the problem, one associated with each class. The model is overall discriminative in nature, as we assume no distribution on inputs. Based on the cluster ID, an appropriate weight vector is chosen from among the K-weight vectors to model the output y_n.

Part-1: Advantage over the standard probabilistic linear model:

To keep this section brief, the first point that I want to highlight is that in the standard model, the relationship between the input features and the output labels is uniform across all the data points. So, say we have inputs clustered around in different groups, out standard model shall try fitting a **single** linear curve to it, which won't work well.

In contrast to it, in the suggested model, we can easily cluster all the available data points into K-different clusters, and then fit a linear model through each one of those K-clusters separately, which is good for seeing patterns in the data, as well as reduces both the training and testing errors if data is in a particular format.

So, essentially, a difficult supervised learning problem has been broken down as a mixture of simpler models, thus making both training and testing much more efficient and reasonable through the combination of both clustering and regression.

An Alternating Optimisation (ALT-OPT) Algorithm to train the model:

The idea here is that as this is a multi-variable optimisation problem, it cannot be solved directly. Hence, we shall use the Alternating Optimisation Technique in every iteration. We shall first update the latent variables $\mathbf{Z} = \{z_1, z_2, ..., z_n\}$, and then choose its best value in order to update the global parameters $\Theta = \{(\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_K), (\pi_1, \pi_2, ..., \pi_K)\}$.

The algorithm suggested for that is as follows:

Algorithm:

In the t^{th} iteration, do the following steps until convergence or until t_{max} is reached:

1. Calculate the following for each n at iteration (t):

$$p(z_{nk}^{(t)} = 1 | \mathbf{y}_n, \mathbf{x}_n, \mathbf{w}_k^{(t-1)}, \Theta^{(t-1)}) = \frac{\pi_k^{(t-1)} \exp\left\{-\frac{\beta}{2}\left\|\mathbf{y}_n - (\mathbf{w}_k^{(t-1)})^\top \mathbf{x}_n\right\|^2\right\}}{\sum_{j=1}^{K} \pi_j^{(t-1)} \exp\left\{-\frac{\beta}{2}\left\|\mathbf{y}_n - (\mathbf{w}_j^{(t-1)})^\top \mathbf{x}_n\right\|^2\right\}} \quad (9)$$

2. For each n, find the index $j \in \{1, 2, ..., K\}$ such that:

$$\boxed{p(z_{nj}^{(t)} = 1|\mathbf{y}_n, \mathbf{x}_n, \mathbf{w}_j^{(t-1)}, \Theta^{(t-1)}) \geq p(z_{nk}^{(t)} = 1|\mathbf{y}_n, \mathbf{x}_n, \mathbf{w}_k^{(t-1)}, \Theta^{(t-1)}) \forall k \in \{1, 2, ..., K\}} \tag{10}$$

3. Set $\boxed{z_{nj}^{(t)} = 1 \text{ and } z_{nk}^{(t)} = 0} \forall k \neq j$. This completes the update of $\mathbf{Z}$ keeping the value of Θ fixed.
4. Now, keeping the value of $\mathbf{Z}$ fixed, update Θ as follows:

$$\boxed{\pi_k^{(t)} = \frac{1}{N} \sum_{n=1}^{N} z_{nk}} \forall k \in \{1, 2, ..., K\} \tag{11}$$

   This works, as we now update the new $\pi_i's$ based on the modifies values of the cluster IDs of the input features. Moving further, to update the weight vectors at iteration (t), we can use the following update equation:

$$\boxed{\mathbf{w}_k^{(t)} = \left(\sum_{n=1}^{N} \mathbb{E}[z_{nk}^{(t)}] \mathbf{x}_n \mathbf{x}_n^\top \right)^{-1} \sum_{n=1}^{N} \mathbb{E}[z_{nk}^{(t)}] \mathbf{x}_n \mathbf{y}_n} \forall k \in \{1, 2, ..., K\} \tag{12}$$

   The way to get to this has been proved as a Lemma below.
5. Now, proceed to the next iteration, by updating $t \longleftarrow t+1$, and continue at step (1) until convergence or t_{max} is reached.

Proving the weight update equation:

The weight update equation can be derived by maximising the expected log-likelihood with respect to the weights $\mathbf{w}_k$, while the latent variables $\mathbf{Z}$ are kept fixed.

In this case, the objective function is:

$$\mathbb{E}_{p(\mathbf{w}, \mathbf{Z}|\mathbf{y}, \mathbf{x}, \Theta)}[\log p(\mathbf{y}|\mathbf{Z}, \mathbf{x}, \mathbf{w})]$$

which has to be maximised, by suitably updating the weight vectors.

Posterior distribution in this case comes out to be:

$$p(\mathbf{w}, \mathbf{Z}|\mathbf{y}, \mathbf{x}, \Theta) = \frac{p(\mathbf{y}|\mathbf{Z}, \mathbf{x}, \mathbf{w}) \cdot p(\mathbf{w}|\Theta) \cdot p(\mathbf{Z}|\Theta)}{p(\mathbf{y}|\mathbf{x}, \Theta)}$$

So, substituitng this in out objective equation, we have:

$$\mathbb{E}_{p(\mathbf{w}, Z|\mathbf{y}, \mathbf{x}, \Theta)}[\log p(\mathbf{y}|Z, \mathbf{x}, \mathbf{w})] = \sum_{n=1}^{N} \sum_{k=1}^{K} \mathbb{E}[z_{nk}] \log p(\mathbf{y}_n|\mathbf{x}_n, \mathbf{w}_k)$$

Now, if we take its derivative with respect to $\mathbf{w}_k$ and set it to 0, we get:

$$\mathbf{w}_k^{(t)} = \left(\sum_{n=1}^{N} \mathbb{E}[z_{nk}^{(t)}] \mathbf{x}_n \mathbf{x}_n^\top \right)^{-1} \sum_{n=1}^{N} \mathbb{E}[z_{nk}^{(t)}] \mathbf{x}_n \mathbf{y}_n$$

as $p(\mathbf{y}_n|\mathbf{x}_n, \mathbf{w}_k) = \mathcal{N}(\mathbf{y}_n|\mathbf{w}_k^\top \mathbf{x}_n, \beta^{-1})$ as per the problem.

Q.E.D.

Please Turn Over (P.T.O.)

Intuition behind the updates in case of $\pi_k = \frac{1}{K} \forall k$:

If $\pi_k = \frac{1}{K} \forall k$, the cluster prior probabilities are uniformly distributed to start with.

The numerator of the expression survives and the denominator of the expression becomes 1:

$$p(z_{nk}^{(t)} = 1 | \mathbf{y}_n, \mathbf{x}_n, \mathbf{w}_k^{(t-1)}, \Theta^{(t-1)}) = \frac{1}{K} \exp \left\{ -\frac{\beta}{2} \left\| \mathbf{y}_n - (\mathbf{w}_k^{(t-1)})^\top \mathbf{x}_n \right\|^2 \right\}$$

This model reduces to a **Gaussian Mixture Model (GMM)** with equal cluster points in each cluster and isotropic Gaussian likelihood. Each cluster is now modeled as a spherical Gaussian distribution.

Final intuition behind updates:

$\mathbf{x}_n$ is assigned to the cluster whose weight vector is such that its dot product with $\mathbf{x}_n$ is closest to the actual label y_n. We check the dot product of $\mathbf{x}_n$ with all the K-weights in $\mathbf{W}$, and the weight which gives the dot product closest to y_n represents the weight of the cluster ID to which $\mathbf{x}_n$ shall be assigned to.

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

Student Name: Om Singh
Roll Number: 210686
Date: November 17, 2023

QUESTION 5

Programming Problem: Part-1:

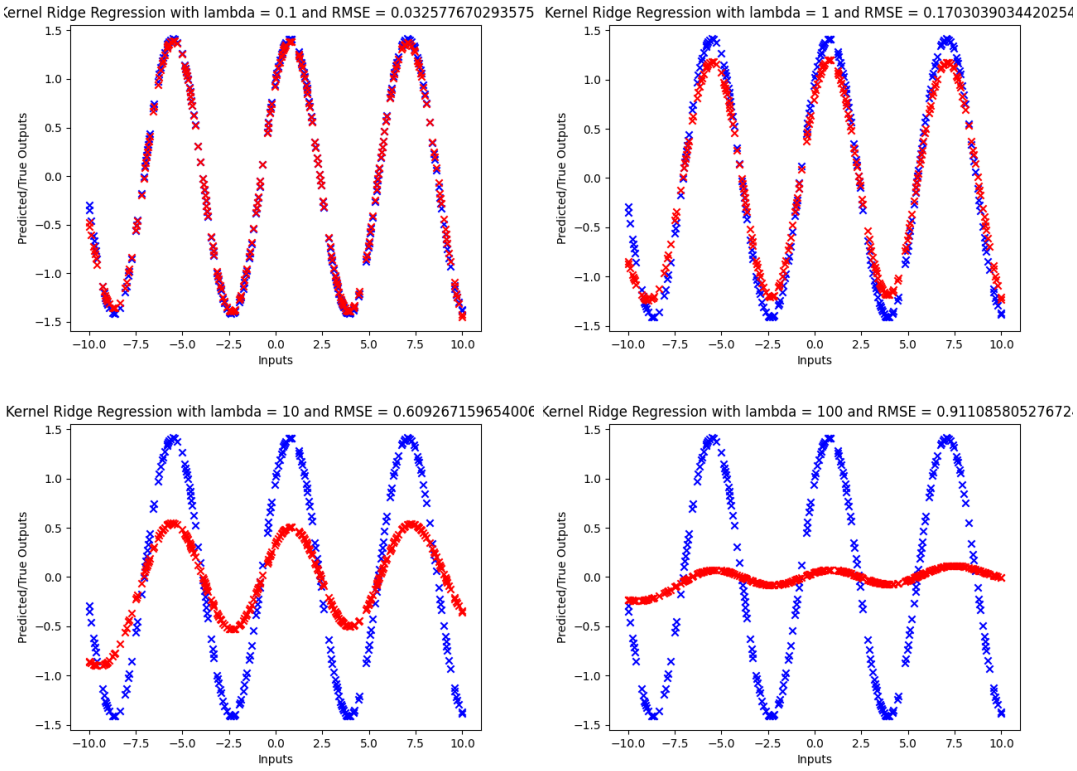
This part basically asks us to implement two regression models:

- Kernel Ridge Regression.
- Ridge Regression with Landmark-based features.

For both the models, RBF (radial basis function) kernel was used, with $\gamma = 0.1$.

Kernel Ridge Regression:

I tried out multiple λ values, as suggested in the problem statement. The plots obtained are the following. The RMSE values are mentioned on the plots themselves:



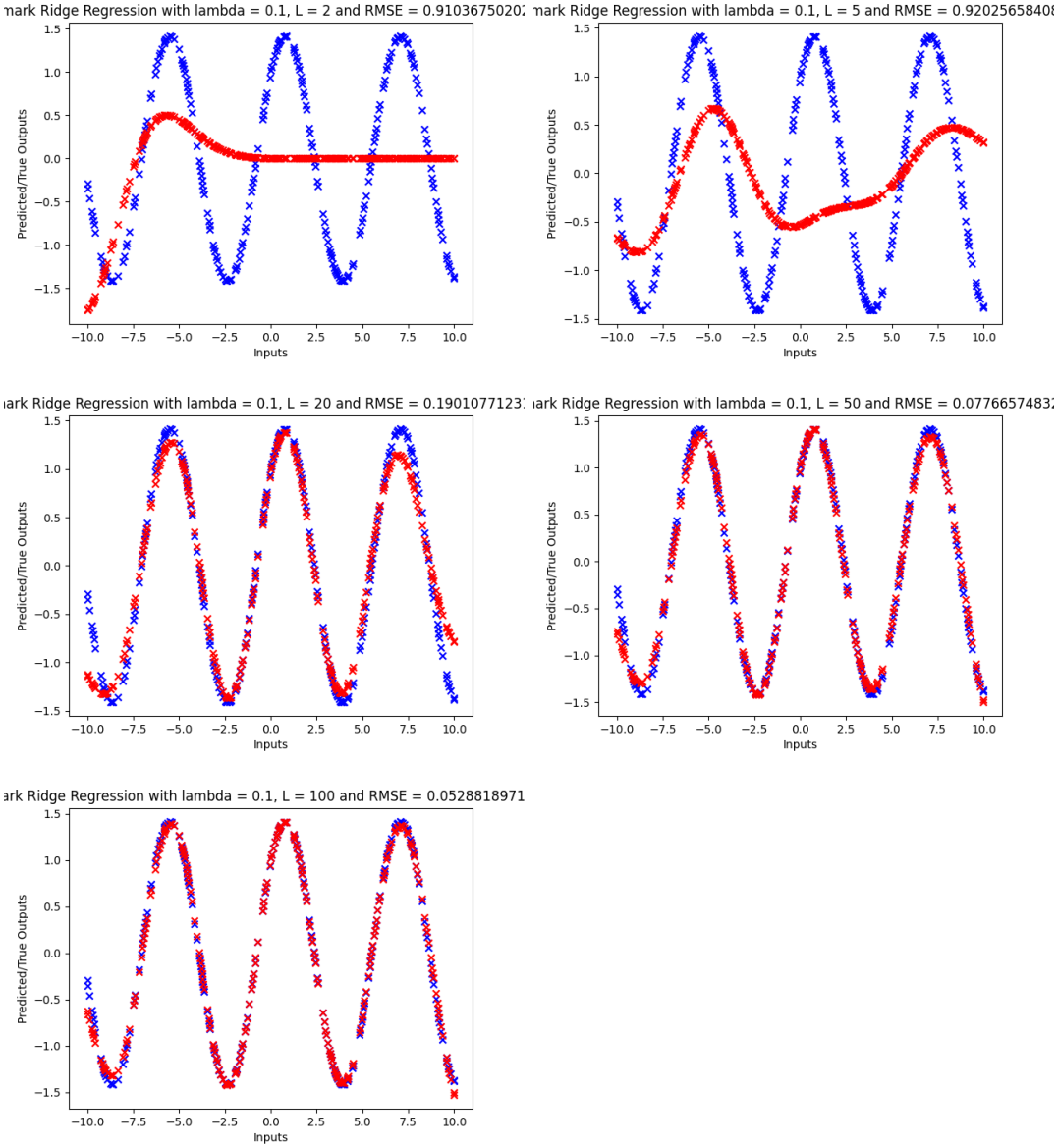
We can see here that as the value of regularisation parameter λ increases, root mean-squared error becomes more and more. The predicted outputs and the true outputs are in maximum agreement for smaller values of λ. The best agreement among the λ values we considered is for $\lambda = 0.1$. However, **we cannot make λ arbitrarily small, as we shall then encounter overfitting of the curve even for outliers**. In this case, the balance seems to be right for $\lambda = 0.1$.

So, larger the emphasis (or regularisation in this case), weaker is the agreement on test-data, and higher is the RMSE (root-mean-squared error), upto a certain extent before the curves start overfitting.

Landmark-based Ridge Regression:

In this, we picked a different number of landmark points from the training set randomly, and trained our linear ridge regression model based on those. In the previous part, $\lambda = 0.1$ works the best. Hence, we keep this regularisation parameter fixed here.

The plots obtained for various values of L (the number of landmark points chosen) are as follows. The RMSE values have been included in the plots itself:



We can observe here that as we increase L, that is, as we increase the number of unformly randomly chosen landmark data points from the training set, the RMSE error decreases, and the plots agree with each other more and more. This is along expected lines, as more the number of training examples (or landmark points in the present case), the better shall the model learn. Hence, its test parameters too shall be better.

However, another thing to note is that **if we take a very high value of L, the curve will start to overfit!** For example, there is a very small decrease in the value of RMSE error as we go from L = 50 to L = 100. As they give similar errors, $L = 50$ **is preferred over** $L = 100$**, as for it, we have less chances of overfitting.**

Programming Problem: Part-2:

In this problem, our main target is to apply the K-means clustering algorithm when the data is such that the clusters aren't spherical and aren't linearly separable. Just to visualise the original dataset, I plotted it, and I have attached the plot below.

Now, we used the following two methods to apply the K-means clustering algorithm to this dataset:

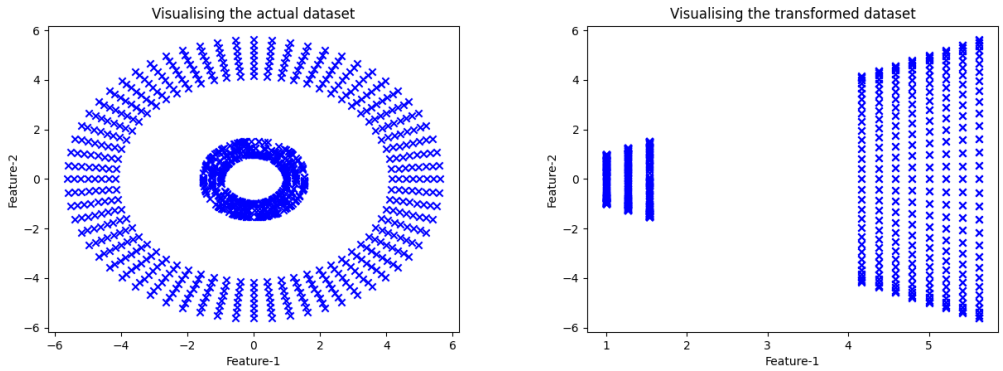
- Hand-crafted features.
- Kernalising using randomly-chosen landmark point.

Using Hand-crafted features:

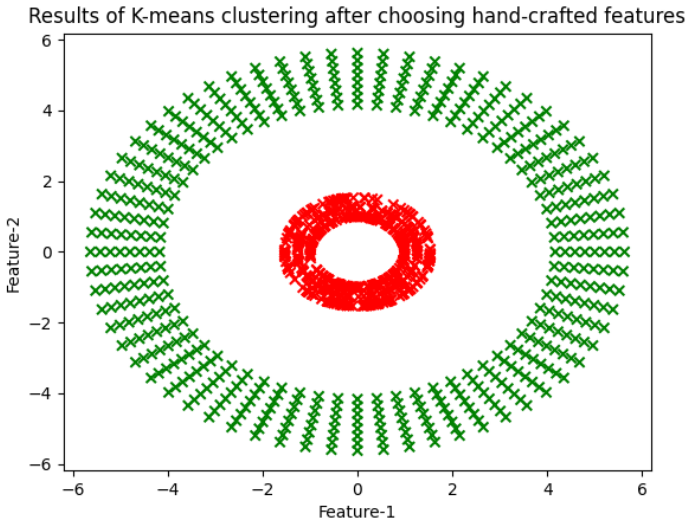
From the plot of the actual dataset above, it is quite clear that the data in the two clusters is centered about the origin point $(0, 0)$.

So, the **transformation that I thought of, which shall work well is to replace one of the coordinates [either x-coordinate or y-coordinate] with the Euclidean distance of the point from the origin**. We could also use the distance as the only coordinate, but anyway, it shall be more general to use the $(\text{distance}, y)$ as a pair after transformation.

Just to visualise the data after performing the transformation, I plotted it, and the plot obtained has been attached below.



As this is good enough, I applied the standard K-means clustering algorithm. The final clusters obtained out of it are the following:

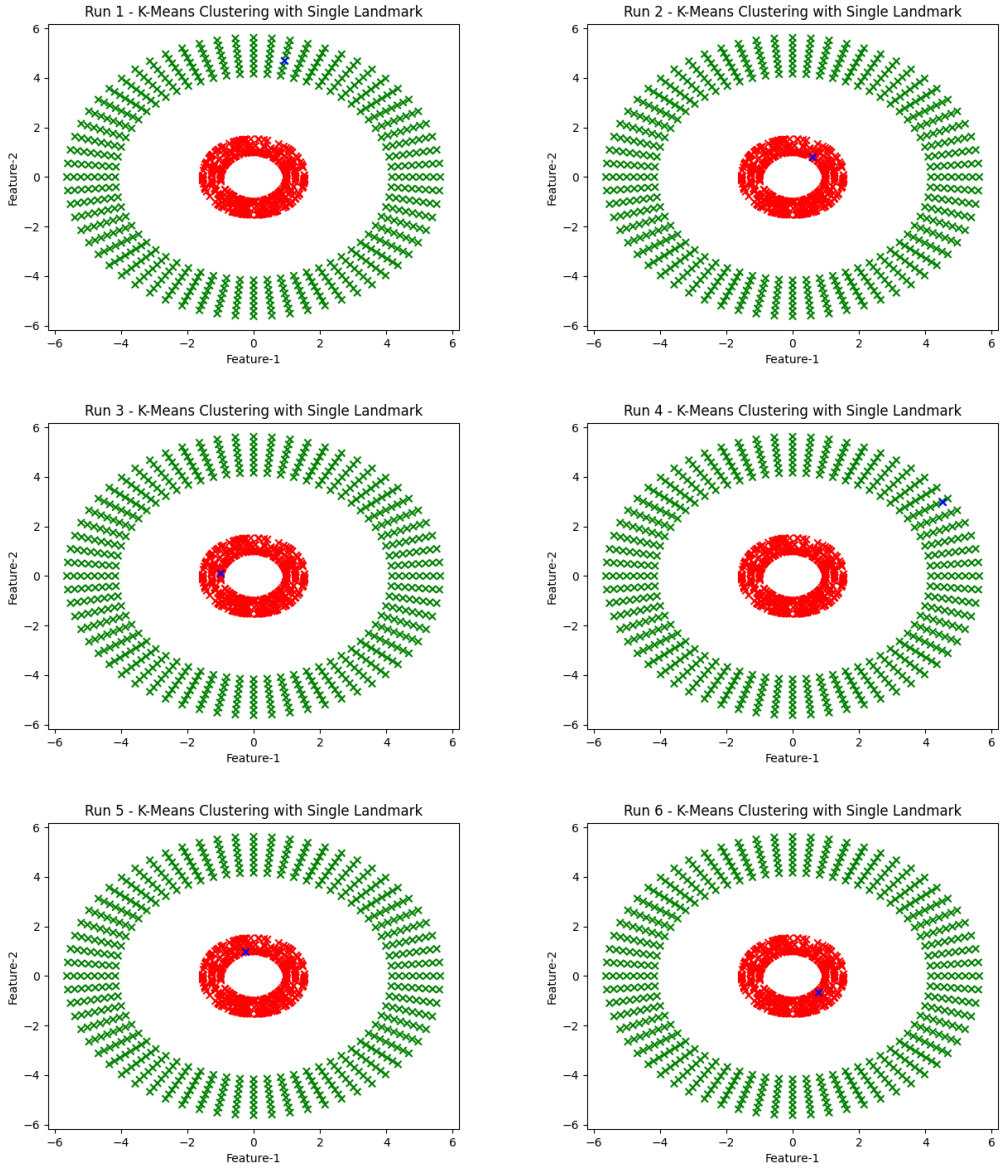


Using Kernels through randomly-chosen landmark point

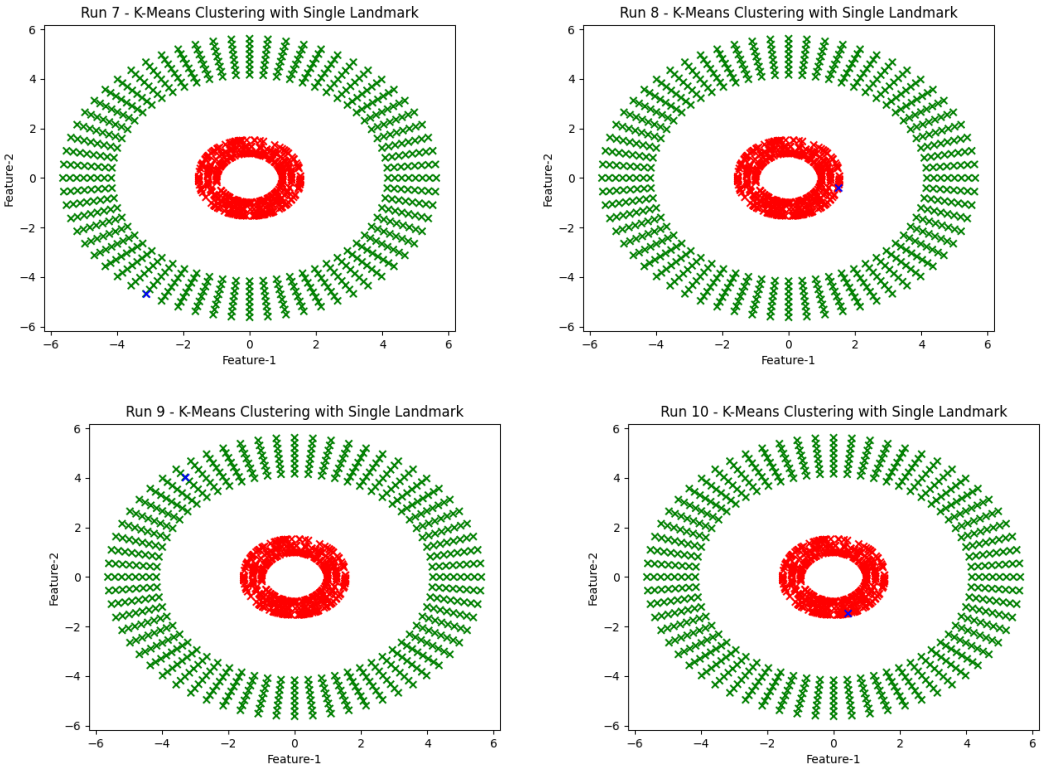
In this method, we picked **only one** landmark-point randomly in 10 different iterations. This landmark-point is the one which we believe contains good features for our implementation.

Fortunately, in my case, all 10 runs produced a valid clustering, differing only in cluster ID assignment. Each time, the landmark point chosen has been **highlighted with a blue cross**.

The clustering obtained alongwith their respective runs are the following:



Continued on the next page..



Justification:

In this case, as the random landmark point is a point from the dataset itself, and the dataset has less (or no!) outliers, the choice of the randomly-chosen data point mostly doesn't matter.

In general, if the landmark point is well-placed, and captures the essence of the data, we get a good clustering. **Bad clustering may occur if the landmark point is poorly placed, like if the landmark point is an outlier, or in a region where the two clusters overlap significantly, or if it is there in a sparse region, cluster tail or even in some transitional region**.

However, in the present case, such situation doesn't take place, as the data is quite uniform.

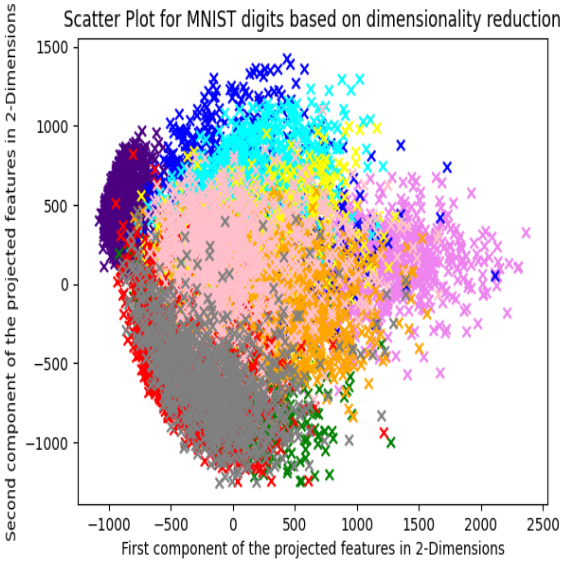
Programming Problem: Part-3:

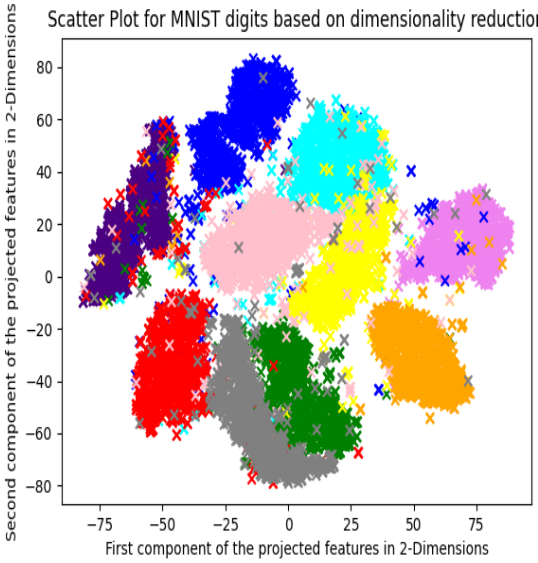
In this part, we were required to compare the results obtained by dimensionality reduction to 2-dimensions using PCA (Principal Component Analysis) and tSNE (t-Distributed Stochastic Neighbour Embedding) for labelled digit data, obtained from MNIST digit data file.

The plots obtained are as follows:

PCA (Principal Component Analysis) (on the left)

tSNE (t-Dist. Stochastic Neighbour Embedding) (on the right)





The color coding used in the problem is the following:

0 - Violet, 1 - Indigo, 2 - Blue, 3 - Cyan, 4 - Green, 5 - Yellow, 6 - Orange, 7 - Red, 8 - Pink and 9 - Gray.

Observations and Concluding comments:

It is clear from the plots that in PCA, classes aren't properly separated, and they often overlap [a pair of classes aren't linearly separable from one another].

On the other hand, in the case of tSNE, the individual scatters of different digits are much better clustered and are distinct from one another. So, for the problem at hand of dimensionality reduction, it is much more worthwhile to choose tSNE as the method of choice than using PCA!

The major reason behind this difference is that PCA is a linear dimensionality-reduction technique. Hence, it doesn't capture non-linear relationships between the digits well. tSNE meanwhile is a non-linear dimensionality-reduction technique, and hence is much more suited for this particular problem.