

## 1. SGD for K-means Objective

- **Step 1 - Assign  $x_n$  "greedily" to the "best" cluster:** This is done by finding the cluster  $k$  that minimizes the distance  $\|x_n - \mu_k\|^2$ . In other words, we set  $z_{nk} = 1$  if  $k = \arg \min_j \|x_n - \mu_j\|^2$  and  $z_{nk} = 0$  otherwise. This step ensures that each data point is assigned to the cluster whose mean is closest to it, hence minimizing the within-cluster sum of squares.
- **Step 2 - The SGD-based cluster mean update equations:** are given by:

$$\mu_k = \mu_k - \eta(z_{nk}(\mu_k - x_n))$$

where  $\eta$  is the learning rate. This update makes sense intuitively because it moves the cluster mean  $\mu_k$  closer to  $x_n$  if  $x_n$  is assigned to cluster  $k$  (i.e., if  $z_{nk} = 1$ ), and does not change  $\mu_k$  if  $x_n$  is not assigned to cluster  $k$  (i.e., if  $z_{nk} = 0$ ). This is because the term  $(z_{nk}(\mu_k - x_n))$  becomes zero if  $z_{nk} = 0$ , leaving  $\mu_k$  unchanged. If  $z_{nk} = 1$ , the term becomes  $(\mu_k - x_n)$ , which is the direction in which  $\mu_k$  needs to move to get closer to  $x_n$ . The learning rate  $\eta$  controls the magnitude of this movement.

- **Step size:** The step size or learning rate  $\eta$  is a hyperparameter that needs to be chosen carefully. A common strategy is to start with a relatively large value of  $\eta$  and decrease it over time, often proportional to  $1/t$  where  $t$  is the iteration number. This is known as learning rate decay. This ensures that the algorithm makes large steps in the beginning when it is far from the minimum, and smaller steps as it gets closer to the minimum. This strategy can help speed up convergence and improve the final result. However, the optimal learning rate and decay schedule can depend on the specific problem and may require some trial and error to find.

This approach allows the K-means algorithm to adapt to new data points as they arrive, making it suitable for online learning scenarios. However, it's worth noting that the convergence of the online K-means algorithm can be slower than the batch version, especially if the data points are not independently and identically distributed. Also, the final solution can depend on the order in which the data points are processed. Therefore, it's often a good idea to shuffle the data points before running the online K-means algorithm. This approach ensures that the algorithm does not get stuck in a suboptimal solution due to the order of the data points.

*Student Name:* Sharvani Jadhav

*Roll Number:* 210960

*Date:* December 1, 2024

---

## 2. An Ideal Projection

The objective function that will achieve this is a combination of maximizing the distance between the means of the inputs from the two classes (inter-class scatter) and minimizing the distance within each class (intra-class scatter). This can be formulated as follows:

The inter-class scatter is defined as the squared distance between the means of the two classes after projection:

$$S_B = (\mu_+ - \mu_-)^2$$

where  $\mu_+$  and  $\mu_-$  are the means of the positive and negative classes after projection, respectively.

The intra-class scatter is defined as the sum of the variances of the two classes after projection:

$$S_W = \sum_{n:y_n=+1} (w^T x_n - \mu_+)^2 + \sum_{n:y_n=-1} (w^T x_n - \mu_-)^2$$

The objective function to be maximized is then the ratio of the inter-class scatter to the intra-class scatter:

$$J(w) = \frac{S_B}{S_W}$$

This objective function makes sense because it tries to find a projection direction  $w$  that maximizes the separation between the classes (as measured by the distance between their means) while minimizing the spread within each class (as measured by their variances). This is exactly what we want in order to achieve good class separability after projection. The solution to this problem gives the direction  $w$  that best separates the classes in the projected space.

---

### 3. Eigenchangers!

Given that  $v$  is an eigenvector of  $\frac{1}{N}XX^T$ , we have:

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

where  $\lambda$  is the eigenvalue corresponding to the eigenvector  $v$ .

We want to find an eigenvector  $u$  of  $S = \frac{1}{N}X^T X$ . Let's multiply both sides of equation (1) by  $X^T$ :

$$\frac{1}{N}X^T XX^T v = \lambda X^T v \quad (2)$$

Comparing equation (2) with the definition of an eigenvector, we can see that  $u = X^T v$  is an eigenvector of  $S$  with the same eigenvalue  $\lambda$ .

The advantage of this method is computational efficiency. When  $D > N$ , the dimensionality of  $S$  is  $D \times D$ , which can be quite large. On the other hand, the dimensionality of  $\frac{1}{N}XX^T$  is  $N \times N$ , which is smaller. Therefore, computing the eigenvectors of  $\frac{1}{N}XX^T$  can be much faster and less memory-intensive. This is particularly useful in Principal Component Analysis (PCA), where the goal is to reduce the dimensionality of a dataset while preserving the most important patterns or relationships between the variables. The eigenvalues give you the explained variance of the corresponding eigenvector. Thus, by sorting the eigenvectors in the decreasing order according to their eigenvalues, you can sort the principal components by importance order, and eventually remove the ones associated with a small variance. This is particularly useful in PCA, where the goal is to reduce the dimensionality of a dataset while preserving the most important patterns or relationships between the variables.

Student Name: Sharvani Jadhav

Roll Number: 210960

Date: December 1, 2024

---

#### 4. Probabilistic Formulation of Matrix Factorization with Side Information

1. **Explanation of the Model:** This model is a form of mixture of experts model where the data is assumed to be generated from one of  $K$  different linear models, each associated with a cluster. The cluster assignment is a latent variable  $z_n$ . The advantage of this model over a standard linear model is that it can capture more complex, non-linear relationships in the data by dividing the input space into regions (clusters) and learning a different linear model for each region.

#### 2. ALT-OPT Algorithm:

- **Update for  $Z$ :** For each data point  $n$ , we compute the posterior probability of each cluster  $k$  given the data and the current parameters, and assign the data point to the cluster with the highest posterior probability. The update equation for  $z_n$  is:

$$z_n = \arg \max_k \{ \log(\pi_k) + \log(\mathcal{N}(y_n | w_k^T x_n, \beta^{-1})) \}$$

- **Update for  $\Theta$ :** For each cluster  $k$ , we update  $w_k$  and  $\pi_k$  as follows:

$$w_k = \left( \sum_n I[z_n = k] \cdot x_n \cdot y_n^T \right) \cdot \left( \sum_n I[z_n = k] \cdot x_n \cdot x_n^T + \beta I \right)^{-1}$$
$$\pi_k = \frac{\sum_n \{I(z_n = k)\}}{N}$$

where  $I(z_n = k)$  is an indicator function that is 1 if  $z_n = k$  and 0 otherwise.

- If  $\pi_k = 1/K$  for all  $k$ , the update for  $z_n$  simplifies to choosing the cluster that maximizes the likelihood of the data point under the corresponding linear model:

$$z_n = \arg \max_k \{ \log(\mathcal{N}(y_n | w_k^T x_n, \beta^{-1})) \}$$

The updates are essentially implementing an Expectation-Maximization (EM) algorithm, where the E-step assigns each data point to the most probable cluster given the current parameters, and the M-step updates the parameters to maximize the likelihood of the data given the current cluster assignments.

## Problem 1

### Part (1)

As the regularization hyperparameter increases, the error also increases.

Lambda: 0.1, RMSE: 0.03257767029357357

Lambda: 1, RMSE: 0.1703039034420253

Lambda: 10, RMSE: 0.6092671596540066

Lambda: 100, RMSE: 0.9110858052767243

This phenomenon may be attributed to both the training and test sets being derived from the same sine curve without many outliers.

With less regularization, the model aligns more closely with the training data, thereby indirectly improving its fit to the test data.

### Part (2)

A lower value of  $L$  results in higher prediction error due to the consideration of fewer feature points.

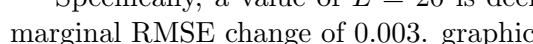
L: 2, RMSE: 0.9467228321571279

L: 5, RMSE: 0.7783258913843927

L: 20, RMSE: 0.13007396711435773

L: 50, RMSE: 0.07977418448460523

L: 100, RMSE: 0.032712050272949475

Specifically, a value of  $L = 20$  is deemed sufficient, as increasing it to 100 only leads to a marginal RMSE change of 0.003. 

## Problem 2

### Part (1)

The proposed approach is to cluster based on the distance of a point from its origin.

### Part (2)

RBF Kernel was kernel is used to kernelize the data. The  $L=1$  was used which seemed rather odd, but gave suprisingly accurate results in some of the iterations. The landmark when chosen closer to the origin gave a higher accuracy in clustering as compared to when away from origin. This can also be theoretically supported by the fact that the clusters are centered around origin and the label of the cluster depends on the distance of the point from origin, which is captured nicely by a landmark near origin.

0.3

### K-means Clustering Results with Hand-Crafted Features

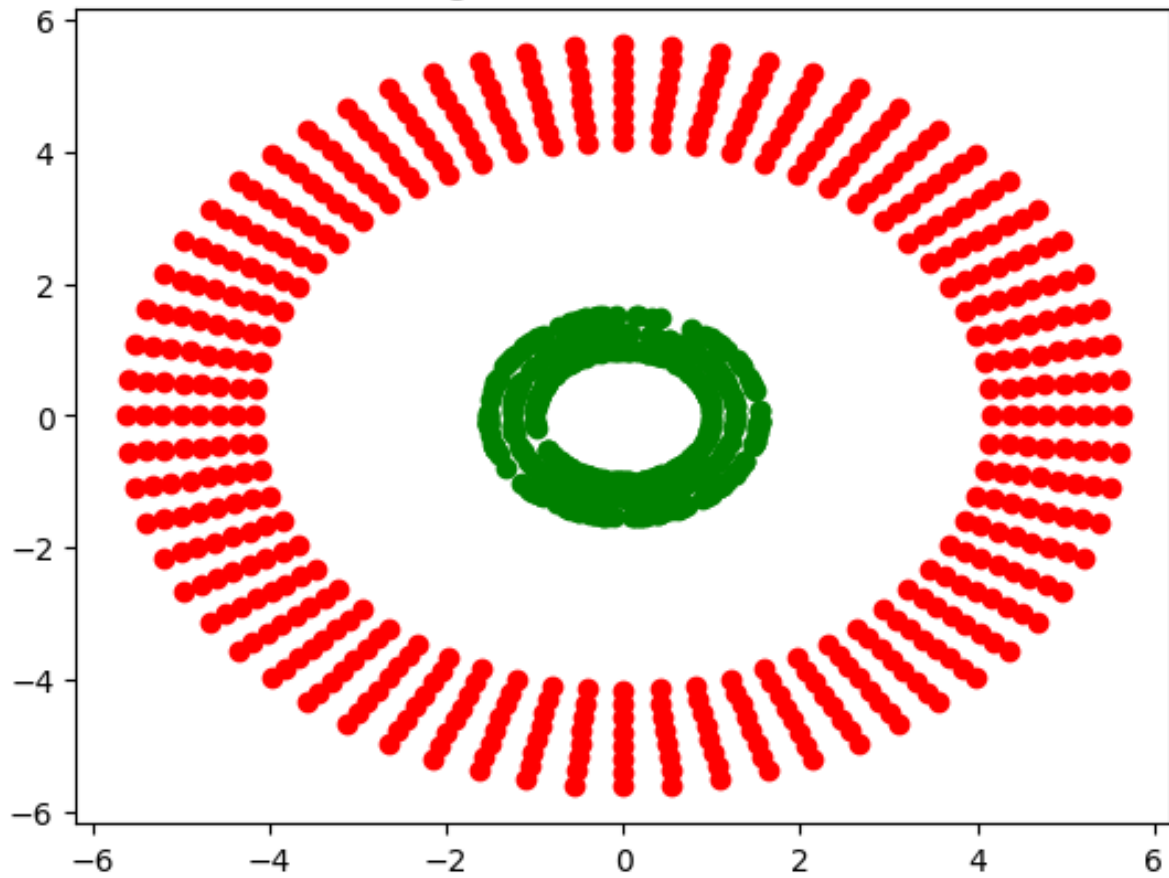
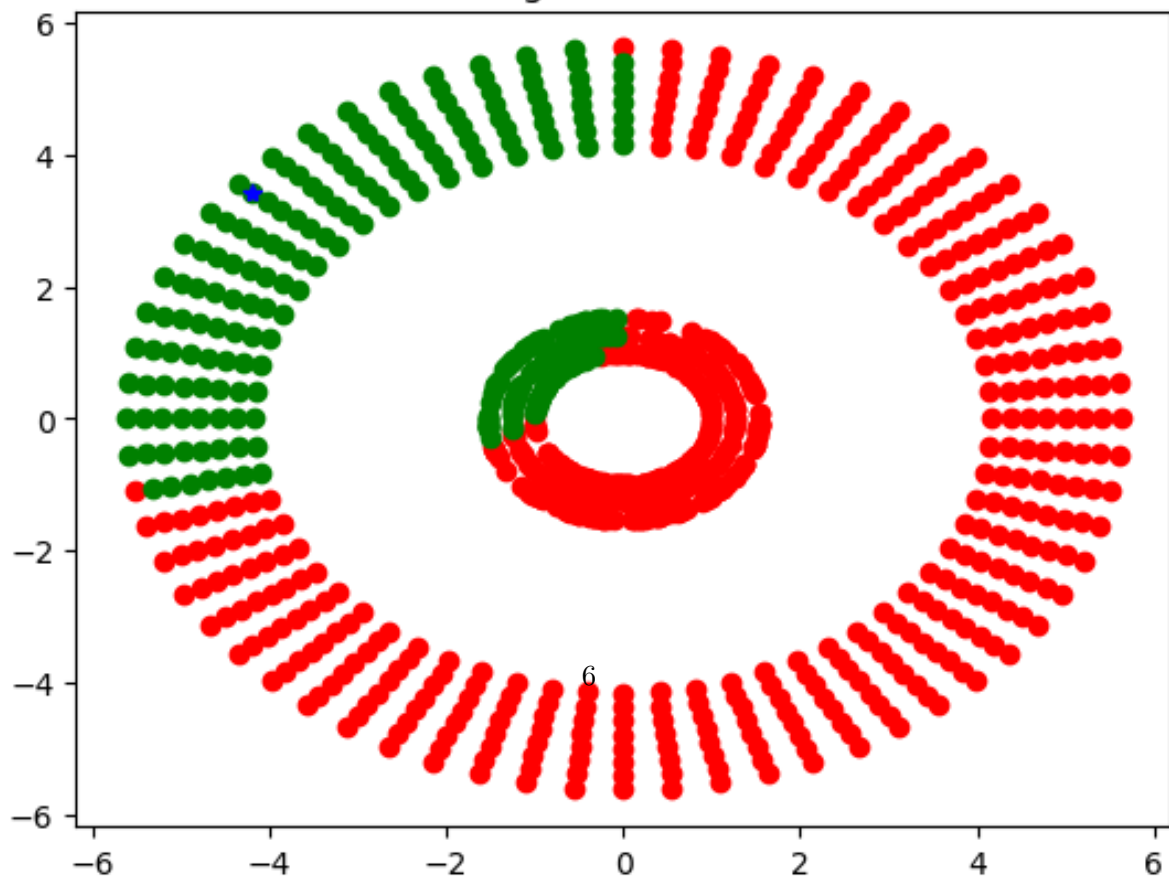


Figure 1: Image 1

0.3

### K-means Clustering Results with Kernel Method



### Problem 3

The PCA is linear, while t-SNE is non-linear. Hence, t-SNE gives better results. Also note that t-SNE is much computationally heavier as it takes a long time to be computed despite using optimized built-in python libraries, while PCA doesn't take much time to compute.

