Q1 Solution

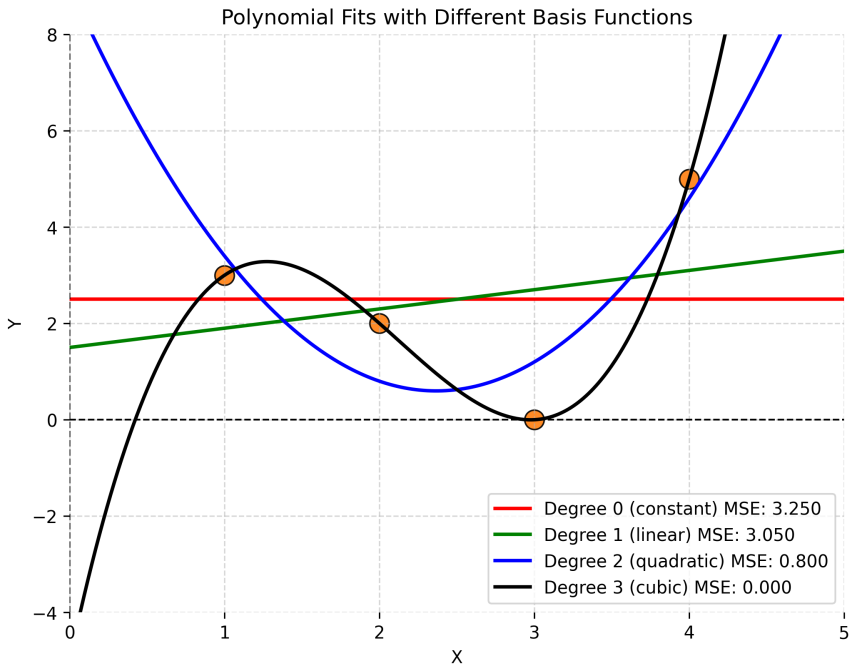

Figure 1: Polynomial fits using different basis functions. MSE values are indicated in the legend.

k	Fitted Equation	MSE
1	$\hat{y} = 2.5$	3.25
2	$\hat{y} = 1.5 + 0.4x$	3.05
3	$\hat{y} = 9 - 7.1x + 1.5x^2$	0.80
4	$\hat{y} = -5 + 15.17x - 8.5x^2 + 1.33x^3$	5.98×10^{-25}

Table 1: Fitted polynomial equations for different values of k and their corresponding mean squared error (MSE).

In Figure 1, we show polynomial fits of varying degrees to the data. As degree k increases, the fitted curve better approximates the data points, as reflected by the decreasing mean squared error (MSE) in Table 1. For k=4, the MSE is extremely small, indicating that the polynomial captures all points in the data

Q2 Solution

(a)(i) Let us define where

$$g_{0.07}(x) := \sin^2(2\pi x) + \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, 0.07^2)$$

We sample the inputs x_i independently from a uniform distribution over $[0, 1]$, 30 times, to produce the following data set.

$$S_{0.07,30} = \{(x_i, g_{0.07}(x_i))\}_{i=1}^{30}.$$

The function $\sin^2(2\pi x)$ in the range $0 \leq x \leq 1$, along with the points of the dataset $S_{\sigma,30}$ superimposed, is shown below:

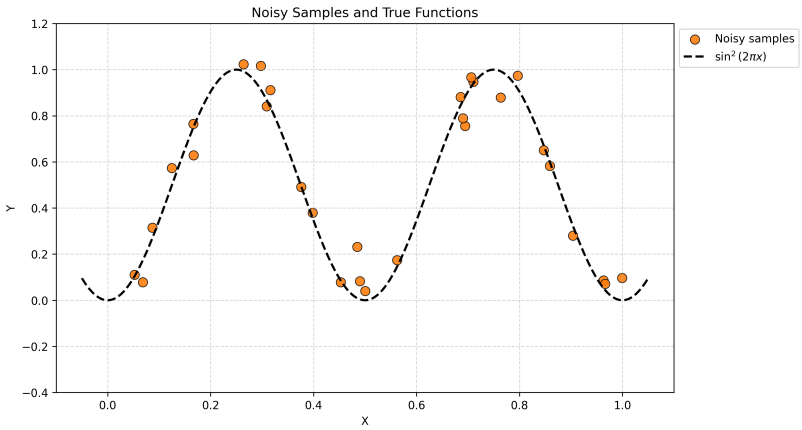

Figure 2: Noisy samples of $g_{0.07}(x) = \sin^2(2\pi x) + \varepsilon$ over $[0, 1]$.

(a)(ii) In this part, we fit the data with a polynomial basis of dimension k=2, 5, 10, 14 and 18. The following plot shows the fitted curves superimposed over the data points.

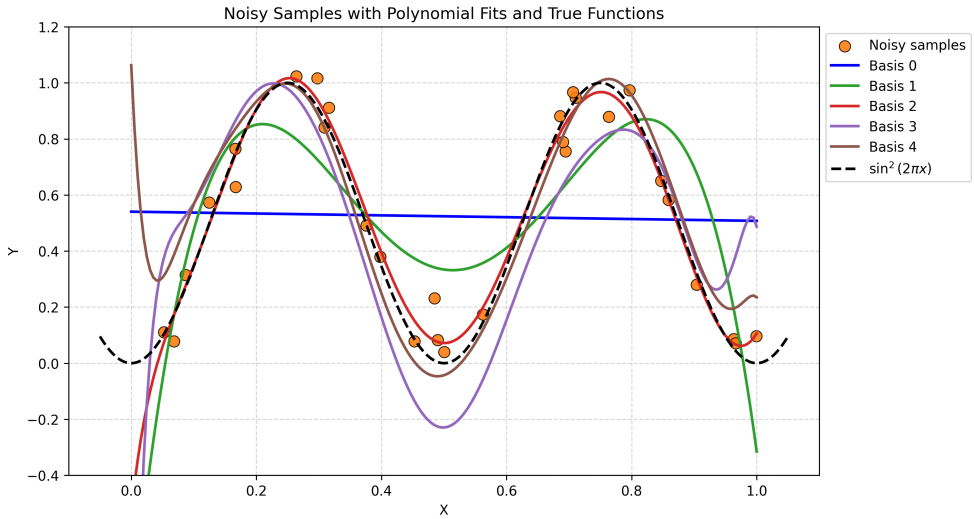

Figure 3: Noisy samples of $g_{0.07}(x)$ with polynomial fits

(b) and (c)

In these parts, firstly we compute the mean squared error (MSE) evaluated on the training data set, by fitting the data set S using polynomial basis functions of dimension k, denoted $tse_k(S)$ for $k = 1, 2, \ldots, 18$.

The second part of the question involves computing the mean squared error (MSE) evaluated on the test data set. Train MSE is not sufficient to evaluate how good a model performs. To assess the generalization performance, we generate a test set of 1000 points as follows:

$$T_{0.07,1000} = \{(x_1, g_{0.07}(x_1)), \ldots, (x_{1000}, g_{0.07}(x_{1000}))\}.$$

For each polynomial dimension k, we construct the corresponding basis functions, use the learned weights to obtain the predicted values $\hat{y}$, and subsequently compute the test mean squared error (MSE). The figure below shows the natural logarithm of the test error, $\ln(\text{tse}_k(T))$, plotted against the model dimension k.

The figure below shows the natural logarithm of the train and test error, $\ln(tse_k(S))$ and $\ln(\text{tse}_k(T))$, plotted against the dimension of the model k.

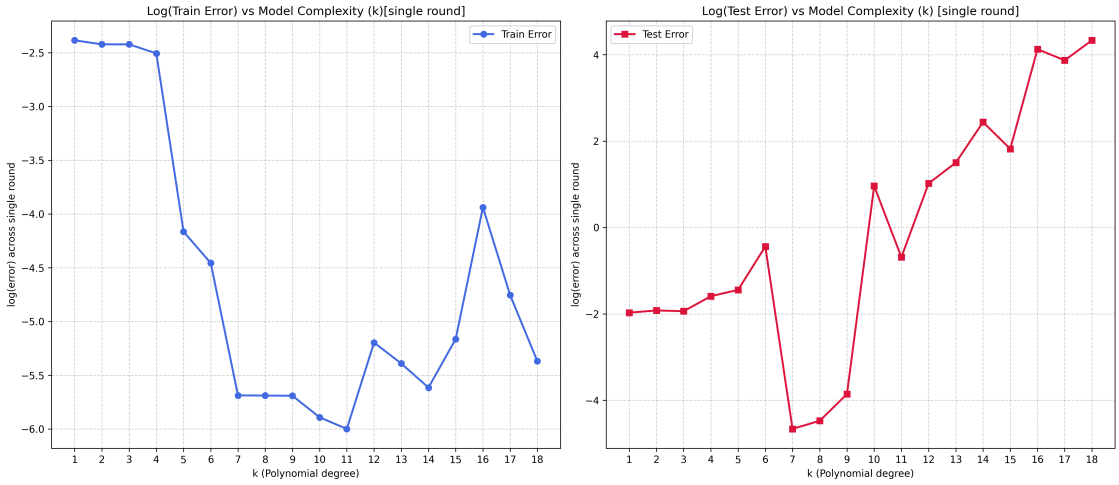

Figure 4: Natural logarithm of the Train and Test MSE against polynomial basis of dimension k.

From the graph above, we make the following observations:

- The Train MSE generally reduces up to $k = 10$ and then starts to show unstable oscillations.
- The Test MSE is similar for $k = 1$ and $k = 2$, beyond which it continues to increase with increasing dimension of the polynomial basis function. This shows that although we are able to reduce MSE on the train data when using higher order polynomial dimension, it doesn't generalize well to unseen data and our performance worsens instead of improving. This phenomenon is called overfitting.
- Different selection of points will give different Train and Test MSE curves.

(d) For any set of given random numbers, we will get slightly different training curves and test curves. In this part of the question, we repeat parts b) to c) 100 rounds instead of a single round, and average the train and test error across the 100 rounds for each dimension.

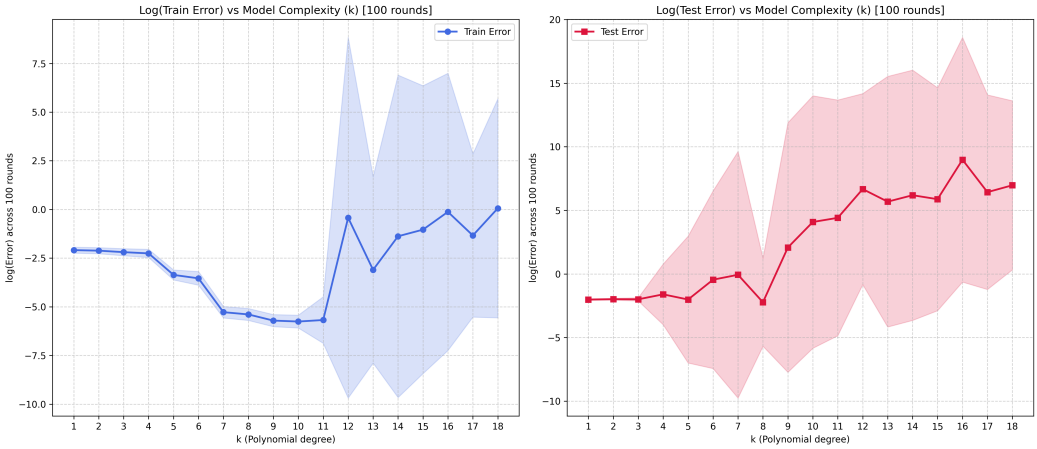

Figure 5: Natural logarithm of the average training and test MSE over a 100 rounds, plotted against the polynomial basis dimension k. Uses explicit inverse of $X^T X$

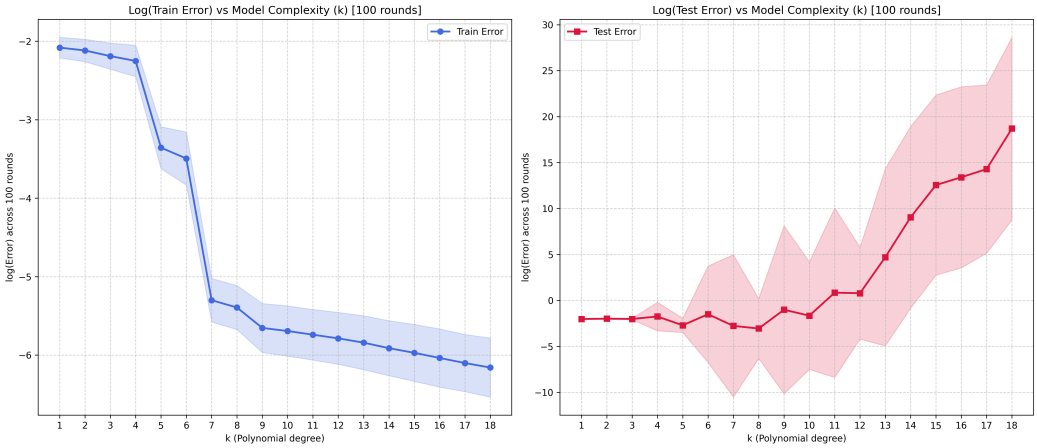

Figure 6: Natural logarithm of the average training and test MSE over 100 rounds after using SVD Decomposition and standardizing features to get the weight vector, plotted against polynomial basis dimension k.

The first graph on the previous page, Figure 5, shows that the training MSE decreases gradually with increasing k up to around $k = 12$, after which it starts to increase. This is because of numerical instability caused by ill-conditioning.

When we use high-degree polynomial basis functions such as $[1, x, x^2, \ldots, x^{k-1}]$, the columns of the design matrix with high dimension become imbalanced in scale. This not only makes the matrix difficult to invert accurately, but it also results in an extremely large coefficient being assigned to the higher order terms. As a result, even very small changes or rounding errors in the data can cause large changes in the fitted coefficients. This leads to unpredictable or unstable behavior in the model, which appears as a sudden increase or fluctuation in the training error.

To compute the weight vector, we currently use the following expression

$$\mathbf{w} = \left(\mathbf{X}_{\text{train}}^T \mathbf{X}_{\text{train}}\right)^{-1} \mathbf{X}_{\text{train}}^T \mathbf{Y}_{\text{train}}$$

Explanation of terms:

- $\mathbf{w}$: The **weight vector** or coefficients of the linear regression model.
- $\mathbf{X}_{\text{train}}$: The **training feature matrix**, where each row is a training example and each column is a feature.
- $\mathbf{Y}_{\text{train}}$: The **training target vector**, containing the true output values for the training data.

In our code, this is implemented as follows:

```
w = np.linalg.inv(X_train.T @ X_train) @ X_train.T @ Y_train
```

This involves computing the inverse of X_train_T @ X_train explicitly. To reduce the numerical instability, we use the following

```
w, residuals, rank, s = np.linalg.lstsq(X_train, Y_train, rcond=None)
```

Which uses SVD Decomposition to obtain the weight vector. By using the method, we are able to obtain the weight vector and reduce the rounding errors. This is numerically more stable, because it does not amplify the rounding errors as opposed to directly taking the inverse of $X^T X$.

Additionally, we also rescale the input features $[1, x, x^2, \ldots, x^{k-1}]$ to have zero mean and unit standard deviation. This transformation is applied to both the train and test design matrices as follows:

$$\mu = \text{mean}(X_{\text{train}}), \qquad \sigma = \text{std}(X_{\text{train}}),$$

$$X_{\text{train}}^{\text{norm}} = \frac{X_{\text{train}} - \mu}{\sigma}, \qquad X_{\text{test}}^{\text{norm}} = \frac{X_{\text{test}} - \mu}{\sigma}.$$

Here, X_{train} is a $30 \times k$ matrix and X_{test} is a $1000 \times k$ matrix, where k is the polynomial dimension. This scaling ensures that higher-order terms have comparable magnitudes, which improves the conditioning of the design matrix and results in more stable solutions.

It is important to note here that we do not standardize the first column, since it is used for the bias term (all ones).

Figure 6 shows the train and test error for dimensions $k = 1 \dots 18$, averaged over 100 rounds after using SVD decomposition and standardizing each column in the design matrix (except for the bias term) to have zero mean and unit variance.

After dealing with numerical instability issues, the conclusion we reach is that as the polynomial basis dimension increases, the train MSE generally reduces. The test MSE reduces slightly up til a dimension of 7, but then starts to increase. This phenomenon is known as overfitting, where the higher-order polynomial terms start fitting the noise in the training data rather than the underlying pattern, leading to poorer generalization on the test set.

Q3 Solution

This part of the question involves using the basis functions $\sin(k\pi x)$ for $k = 1, \ldots, 18$ to fit the training data generated by

$$S_{0.07,30} = \{(x_i, g_{0.07}(x_i))\}_{i=1}^{30}.$$

We will also evaluate the performance of the model on the test data generated by

$$T_{0.07,30} = \{(x_i, g_{0.07}(x_i))\}_{i=1}^{30}.$$

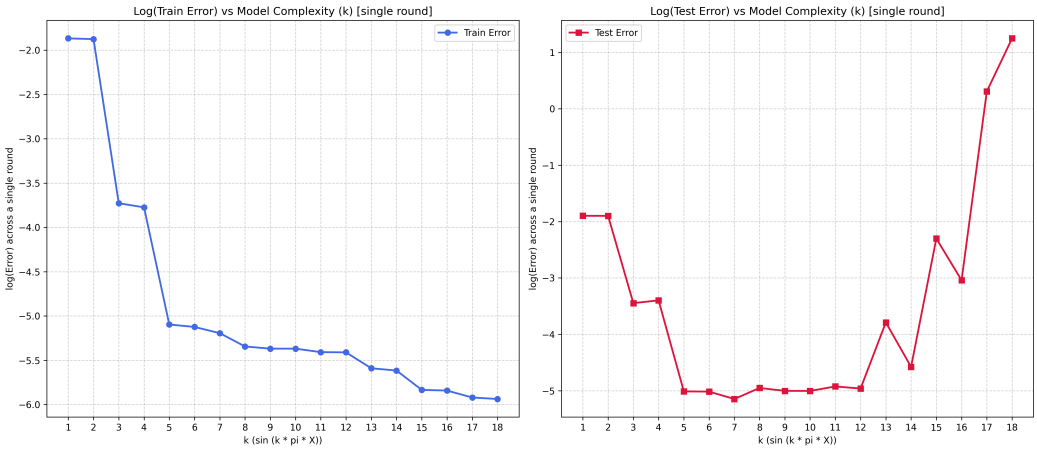

Figure 7: Natural logarithm of the average training and test MSE over a single round, plotted against sin basis dimension k.

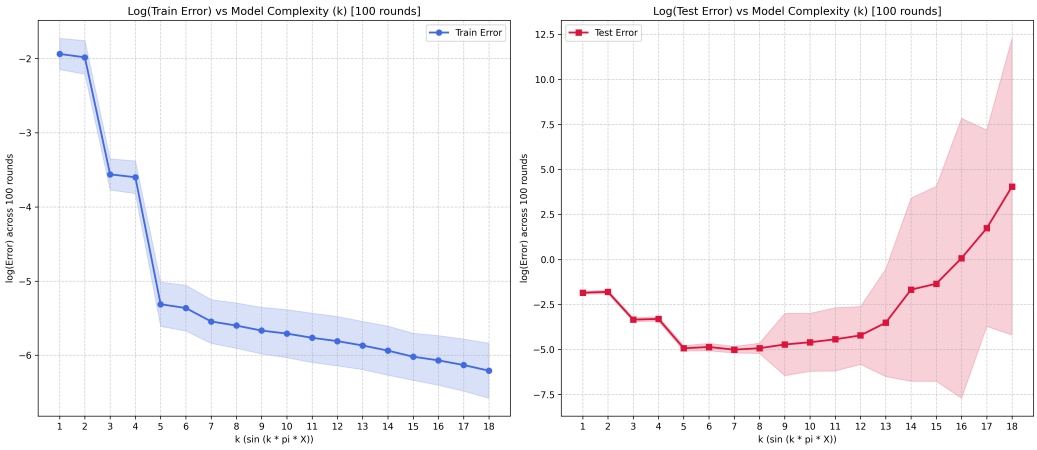

Figure 8: Natural logarithm of the average training and test MSE over a 100 rounds, by explicitly computing inverse, plotted against sin basis dimension k.

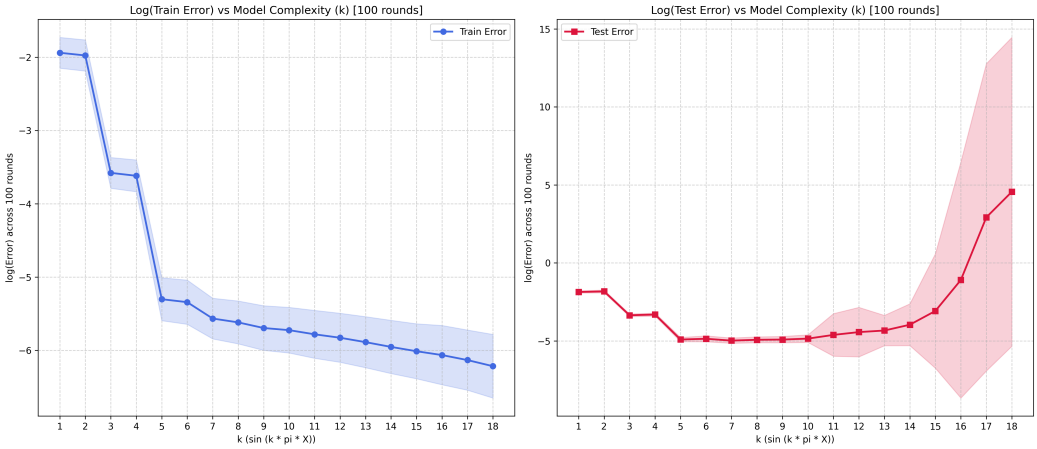

Figure 9: Natural logarithm of the average training and test MSE over a 100 rounds, by using SVD Decomposition, plotted against sin basis dimension k.

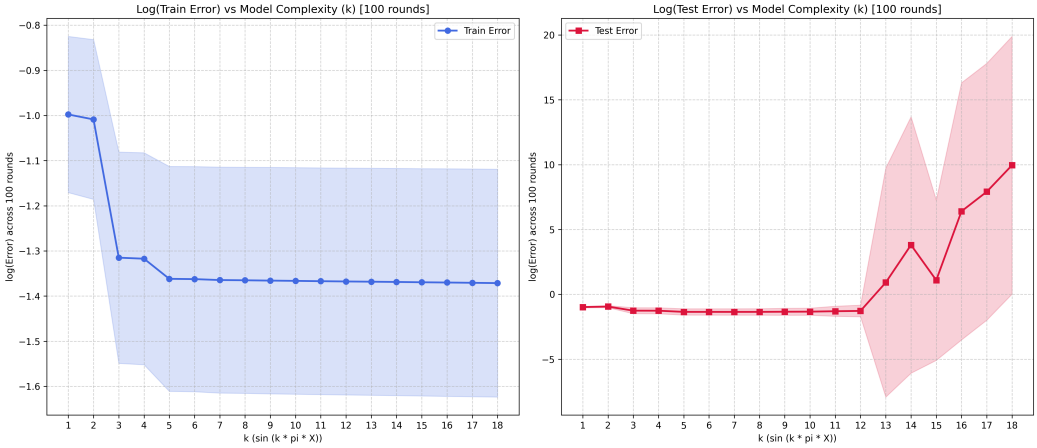

Figure 10: Natural logarithm of the average training and test MSE over a 100 rounds, by using SVD Decomposition and standardizing features, plotted against sin basis dimension k.

Observations

- For a single run, the train MSE curve remains consistent across repeated executions of the script. However, the test MSE curve becomes unstable and varies significantly for $k > 10$, suggesting that the model has overfit the training data. As a result, the test error becomes highly sensitive to the particular selection of data points in the test set.
- The Train MSE curve when using the sin basis function generally remains the same when computing the inverse of $X^T X$ explicitly and using SVD Decomposition. This is expected because the design matrix is not ill-conditioned, it is well-conditioned, i.e. each feature column lies in the same range between -1 and 1.
- The Test MSE decreases gradually decreases till k=5, remains the same between k=5 and k=10 and then starts to increase. This suggests that beyond $k > 10$, the model starts to fit noise and does not generalize well on unseen data.
- Standardizing the features worsens the Train MSE. This happens because firstly our train dataset only consists of a very limited number of data points (i.e. 30) and secondly, by forcing each column in our design matrix to have 0 mean, we shift some of the values in each column to become more negative. Whereas the original function $sin^2(x)$ lies between the range $0 <= f(x) <= 1$! To remedy the situation, we can either add a bias term in our basis function or normalize y.

Q4 Solution

(b) When X is a vector of ones of the same length as the training set, m, we have

$$X^\top X = m, \qquad (X^\top X)^{-1} = \frac{1}{m}.$$

Also,

$$X^\top y = \sum_{i=1}^{m} y_i.$$

Thus the least-squares solution is

$$w = (X^\top X)^{-1} X^\top y = \frac{1}{m} \sum_{i=1}^{m} y_i,$$

which is simply the sample mean of y. **(d)**

Method	**MSE (Train)**	**MSE (Test)**
Naive Regression	83.171 ± 2.846	87.127 ± 5.758
Linear Regression (attr. 1)	70.476 ± 2.846	74.885 ± 6.116
Linear Regression (attr. 2)	71.466 ± 2.739	77.831 ± 5.654
Linear Regression (attr. 3)	62.898 ± 3.356	68.541 ± 6.827
Linear Regression (attr. 4)	80.333 ± 2.825	85.267 ± 5.756
Linear Regression (attr. 5)	67.029 ± 2.956	73.296 ± 6.125
Linear Regression (attr. 6)	42.905 ± 3.978	45.434 ± 7.917
Linear Regression (attr. 7)	70.306 ± 3.148	76.983 ± 6.501
Linear Regression (attr. 8)	77.411 ± 3.048	83.000 ± 6.314
Linear Regression (attr. 9)	70.581 ± 3.081	75.681 ± 6.255
Linear Regression (attr. 10)	64.058 ± 3.240	69.967 ± 6.524
Linear Regression (attr. 11)	61.652 ± 3.190	65.077 ± 6.480
Linear Regression (attr. 12)	37.491 ± 1.477	40.663 ± 2.993
Linear Regression (all attributes)	**21.385 ± 1.800**	**25.880 ± 3.763**

Table 2: Train and test MSE for linear regression on the Boston Housing dataset, using individual attributes and all attributes.

Note that the results will change slightly each time that the notebook is run because shuffling happens randomly and the train and test splits might be different across the 20 trials.

Note, also that since the target variable and feature 12 are correlated, as shown in the scatter plot below, it makes sense that train and test MSE for feature 12 is low when performing linear regression using only that feature.

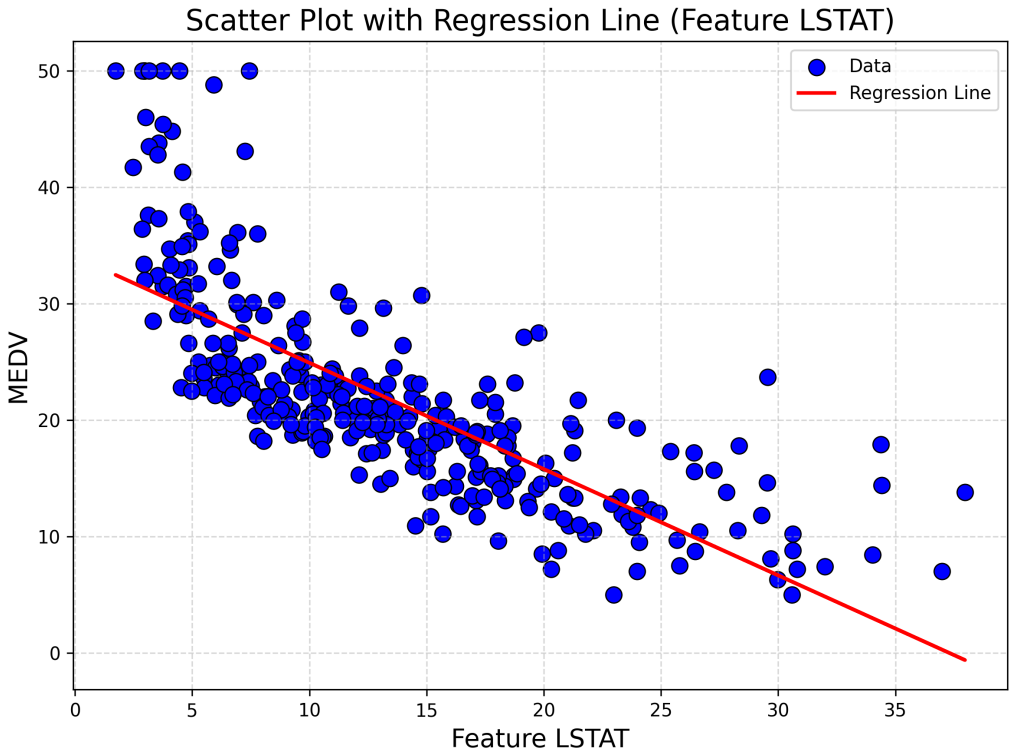

Figure 11: Scatter plot between the target variable and feature 12, along with the fitted line.

Q5 Solution

(a) After performing kernel ridge regression by running five-fold cross-validation on the training set (randomly chosen 2/3rd of the data), the following values of γ and σ perform the best:

$$\gamma = 2^{-27} \quad \sigma = 2^{8}$$

(b) The following are the plots of cross-validation errors as a function of γ and σ:

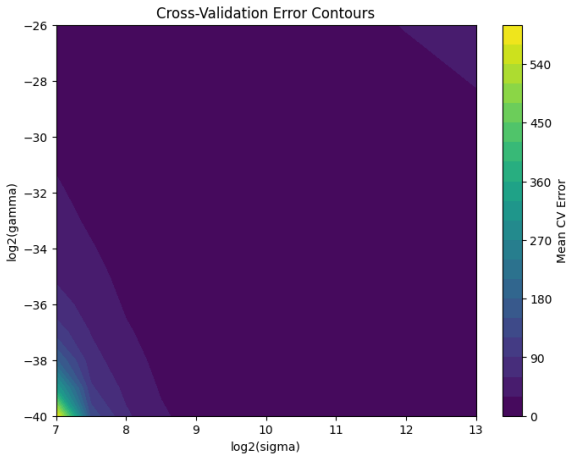

Figure 12: Contour Heatmap of CV error as a function of both γ and σ

For clarity, we remove the $\sigma = 2^7$ from the heatmap since there were some anomalously large values of cv error for this σ and visualize again:

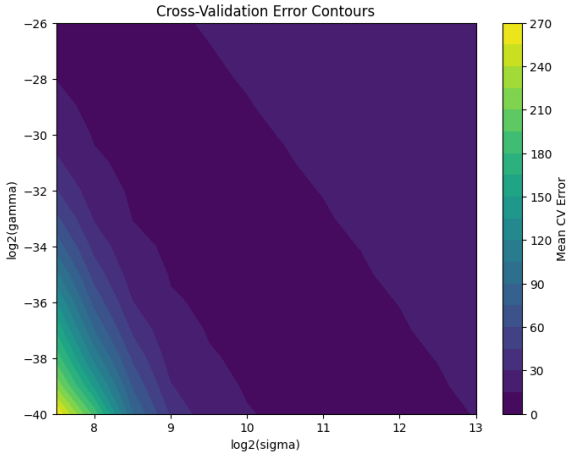

Figure 13: Contour heatmap of CV error with $\sigma = 2^7$ removed

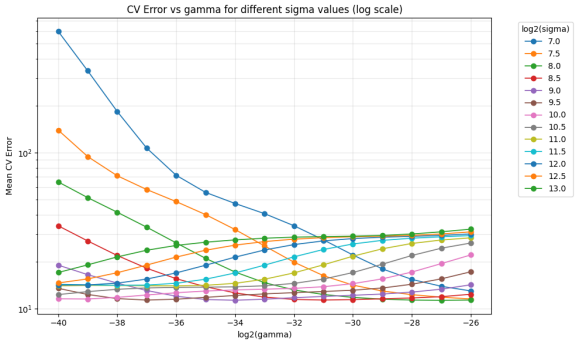

Figure 14: CV error vs γ for different values of σ

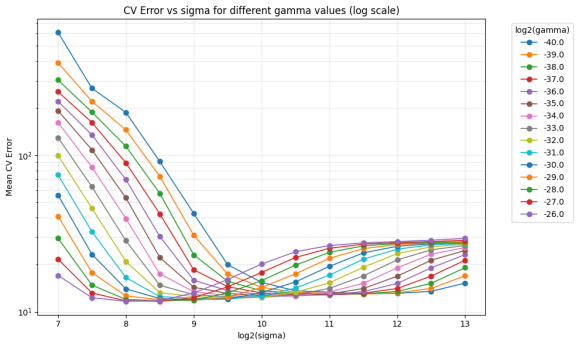

Figure 15: CV error vs σ for different values of γ

As can be seen from the plots, for low values of σ, the error is extremely high for small γ. This is due to overfitting. For small σ, the value of the Kernel $K(x_i, x_j) = exp(-\frac{||x_i - x_j||^2}{2\sigma^2})$ becomes very low for most data points outside the neighborhood of a point and the Kernel matrix is nearly diagonal. So, a point is "influenced" mostly by itself and a small neighborhood of points close to it. This causes overfitting unless the regularisation (γ) is sufficiently high. This is especially visible in the second plot.

As σ increases, the curves flatten out and the best values of σ and γ are found

For very high values of σ the error starts increasing with increasing σ. This likely indicates underfitting as, for very large σ almost all entries of the Kernel matrix start approaching 1, so the Kernel acts almost as a constant function, hence causing underfitting.

c. After calculating the best values of γ and σ using cross-validation the training set, we use the entire training and test set to calculate the train and test MSE. Calculating the train and test MSE with just one iteration gives us the following values:

$$\text{MSE Train} = 6.169$$

$$\text{MSE Test} = 14.472$$

(d)

Method	**MSE Train**	**MSE Test**
Naive Regression	83.171 ± 2.846	87.127 ± 5.758
Linear Regression (attr. 1)	70.476 ± 2.846	74.885 ± 6.116
Linear Regression (attr. 2)	71.466 ± 2.739	77.831 ± 5.654
Linear Regression (attr. 3)	62.898 ± 3.356	68.541 ± 6.827
Linear Regression (attr. 4)	80.333 ± 2.825	85.267 ± 5.756
Linear Regression (attr. 5)	67.029 ± 2.956	73.296 ± 6.125
Linear Regression (attr. 6)	42.905 ± 3.978	45.434 ± 7.917
Linear Regression (attr. 7)	70.306 ± 3.148	76.983 ± 6.501
Linear Regression (attr. 8)	77.411 ± 3.048	83.000 ± 6.314
Linear Regression (attr. 9)	70.581 ± 3.081	75.681 ± 6.255
Linear Regression (attr. 10)	64.058 ± 3.240	69.967 ± 6.524
Linear Regression (attr. 11)	61.652 ± 3.190	65.077 ± 6.480
Linear Regression (attr. 12)	37.491 ± 1.477	40.663 ± 2.993
Linear Regression (all attributes)	**21.385 ± 1.800**	**25.880 ± 3.763**
Kernel Ridge Regression	7.734 ± 1.271	13.163 ± 3.612

Table 3: Mean squared error (MSE) for training and test sets using linear regression on the Boston Housing dataset, applied to individual attributes and all attributes AND using Kernel Ridge Regression

Note that the results will change slightly each time that the notebook is run because shuffling happens randomly and the train and test splits might be different across the 20 trials.

Q6: Generating the data and Decision Boundary

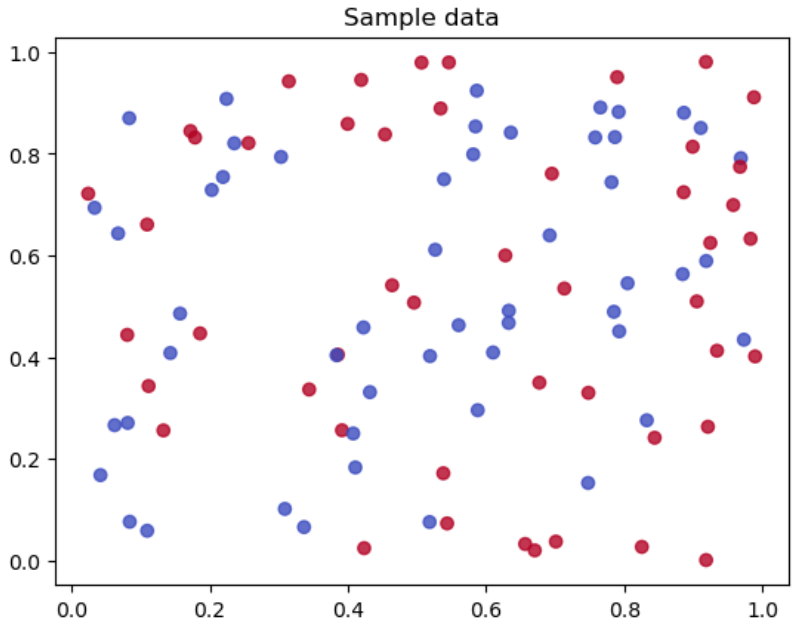

Figure 16: Generated Hypothesis

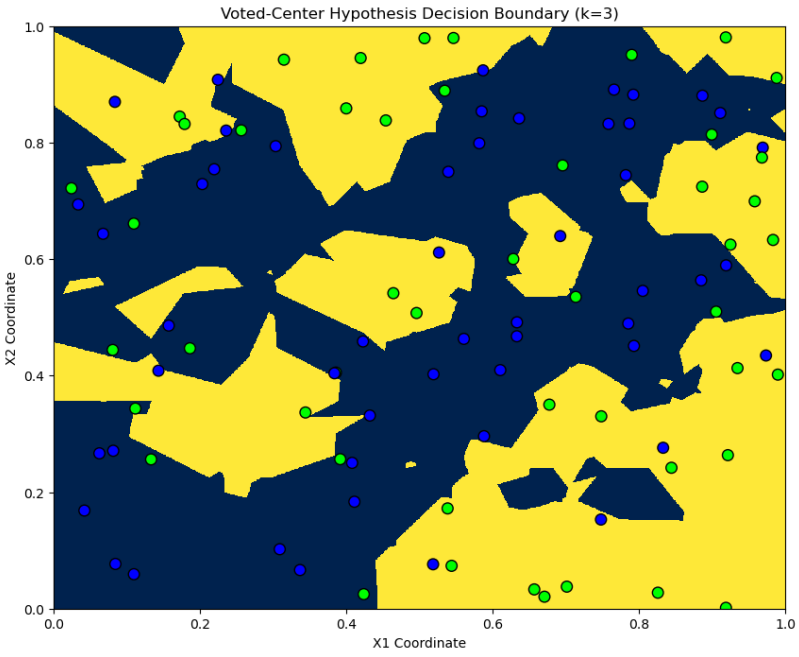

Figure 17: Decision Boundary

Q7: Estimated generalization error of k-NN as a function of k

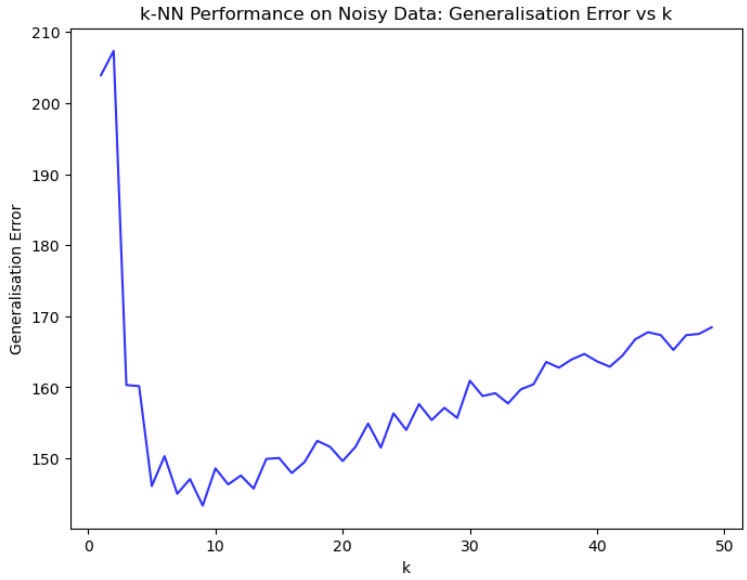

Figure 18: Average Optimal k vs m

7.1 Observations

1. The k-NN generalization error for $k = 1, 2$ is the highest among the values of k, which makes sense for $k < 3$ given we have noisy data. The algorithm is unable to correctly treat noisy data as the focus for small k is very 'local' - this leads to higher variance, due to overfitting on local information and thus being unable to offset 'bad' data with the 'good' surrounding data.

2. It is also worth noting that the generalisation error seems to be around the 20 percent mark, which corresponds to the level of noise introduced in test data generation from the hypothesis, suggesting the algorithm struggles with bad data, which further signifies overfitting.

3. As we go from k = 3 upwards, we are able to overcome local noisy data and make better predictions. This performance peaks around k = 7-9. This seems to be the sweet spot for, considering enough neighbors to balance potentially bad local data, against considering too large of a neighborhood.

4. As we go further beyond ($k \geq 10$), we see the impact of considering too large a neighborhood, with incorrect predictions creeping in more and more, because we are starting to take votes from a much wider space than required.

Q8: Determine optimal k as function of training dataset size (m)

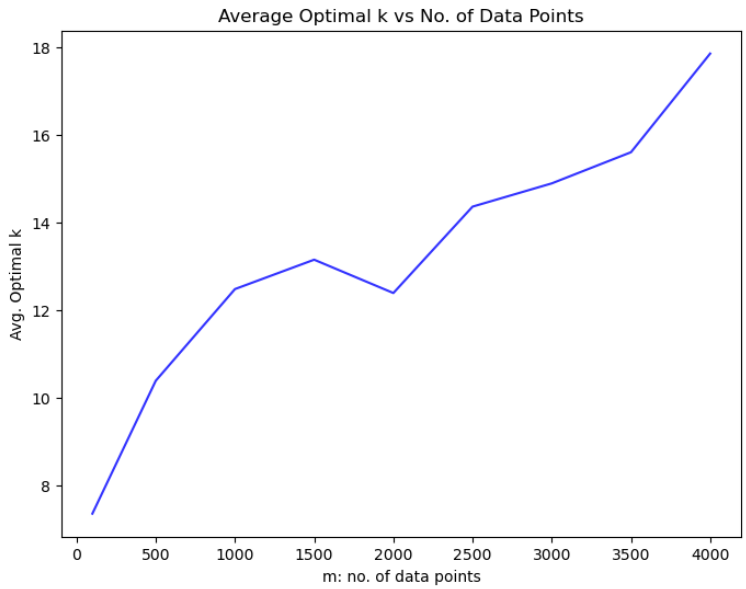

Figure 19: Average Optimal k vs $\sqrt{\mathbf{m}}$

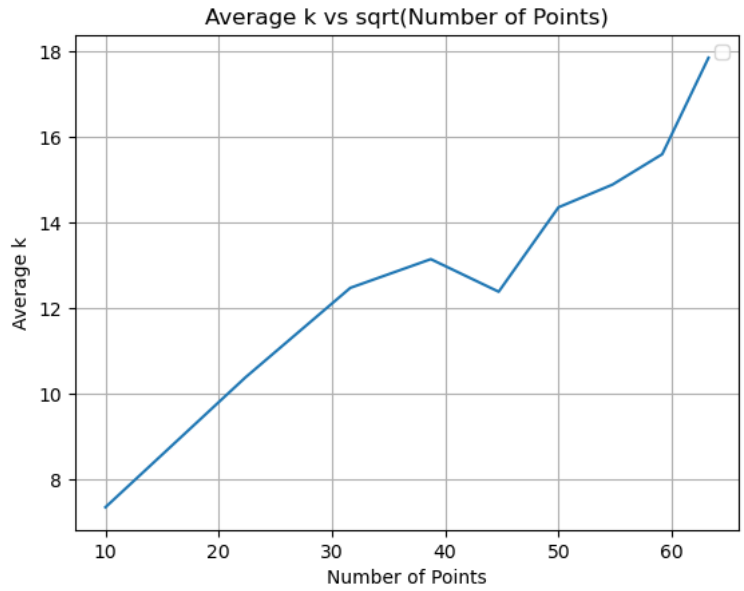

Figure 20: Rules for Optimal k vs m (courtesy: lecture slides)

8.1 Observations

1. As the number of data points increases, the (average) optimal value of k increases.

2. For smaller training data sets, the optimal k is lower, suggesting that a sparser set requires a smaller number of voting candidates to focus more on the surrounding, local neighbors - instead of straying too far out to find "k" neighbors.

3. The optimal k grows as the size of the data set increases. This is expected because as we grow the number of data points, more and more training points are in a local area (denser space). To ensure we get all relevant but local information, we need to increase k so as to cover a reasonable surrounding area. Otherwise, we may miss out on relevant local information if k is too small. By increasing k, we can avoid overfitting due to looking at a subset of the relevant local information.

4. Finally, the rate at which optimal k increases with training data size (m) seems to be going down as we keep increasing m to larger values. This indicates a balance between (a) increasing k to get all the local, relevant information and (b) considering too many neighbors - the latter of which could lead to underfitting. This is perhaps changing the definition of 'local' as the data set grows larger.
Relevant rule discussed in lectures: snapshot added below

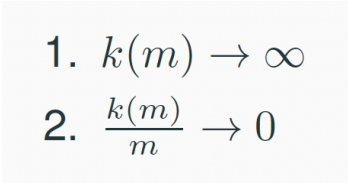

Figure 21: Rules for Optimal k vs m (courtesy: lecture slides)

5. Plotting k against different transformations of m, it was found that k vs sqrt(m) - as seen in the final plot - gives us a straight line, within the empirical bounds of the experiment. This suggests that the average optimal k increases as sqrt(m)

Q9 Solution

(a) For the function K to be a positive semi-definite kernel, we have to prove two things:

i) It is symmetric, i.e.,

$$K(x, z) = K(z, x)$$

This is obvious from the definition of K.

ii) The kernel matrix K is positive semi-definite, i.e. for any $a \in \mathbb{R}^n$:

$$a^T K a \geq 0$$

In other words,

$$\sum_{i,j} a_i a_j K(x_i, x_j) \geq 0$$

Given,

$$\begin{aligned}\sum_{i,j} a_i a_j K(x_i, x_j) &= \sum_{i,j} a_i a_j \left(c + \sum_{d=1}^{n} x_{i,d} x_{j,d} \right) \\ &= \sum_{i,j} c a_i a_j + \sum_{i,j} \sum_{d=1}^{n} a_i a_j x_{i,d} x_{j,d} \\ &= c \left(\sum_i a_i \right)^2 + \sum_{d=1}^{n} \left(\sum_i a_i x_{i,d} \right)^2 \geq 0\end{aligned}$$

The second term is always greater than or equal to zero (sum of squares). The first term $c(\sum_i a_i)^2 \geq 0$ when $c \geq 0$.

Thus, K is a positive semi-definite kernel when $c \geq 0$.

(b) Adding "c" introduces a **bias term** in least squares linear regression.

We can also see this considering the feature map for the kernel:

$$\Phi(x) = \begin{pmatrix} \sqrt{c} \\ x_1 \\ \vdots \\ x_n \end{pmatrix}$$

where $x = (x_1, \ldots, x_n)$.

So,

$$K(x, z) = \langle \Phi(x), \Phi(z) \rangle$$

Thus, Φ is a feature map for K, and we can see this is introducing a constant bias to least squares linear regression.

Introducing c does not impact the weights learned, only adds a constant bias term to the least squares linear regression

Q10: Gaussian RBF Kernel replicating 1-NN

Linear classifier:

$$g(t) = sign(f(t))$$

We are given f(t) as:

$$f(t) = \Sigma_i \alpha_i K(t, x_i)$$

, where x_i is the i-th data point in the training set.

Using the Gaussian Kernel and substituting, we have:

$$f(t) = \Sigma_i \alpha_i e^{-} \beta ||x_i - t||^2$$

The 1-NN classifier takes the point closest to the test data t (say, x_j) and predicts the class based on the label for that training data point x_j. Therefore, if f(t) resembles the 1-NN classifier (i.e. "consult with the closest data point"), g(t) will give the same output as the 1-NN classifier.

The 1-NN classifier can be written with $label(.) \in \{-1, 1\}$ as:

$$h(t) = label(\arg min_x \ ||x - t||^2)$$

Let α_i be the label of the corresponding data point x_i. We have:

$$g(t) = sign(\ \Sigma_i(\ label(x_i) * e^{-\beta||x_i - t||^2}\ )\ )$$

Assume we have m data points $\{x_1, x_2, ..., x_m\}$, we can write g(t) as:

$$g(t) = sign(\ (label(x_1) * e^{-\beta||x_1 - t||^2} + (label(x_2) * e^{-\beta||x_2 - t||^2} + ... + (label(x_m) * e^{-\beta||x_m - t||^2}\ )$$

Say the nearest neighbour is x_k. Thus, for the replication of 1-NN, we will have:

$$sign(\ (label(x_1) * e^{-\beta||x_1 - t||^2} + (label(x_2) * e^{-\beta||x_2 - t||^2} + ... + (label(x_m) * e^{-\beta||x_m - t||^2}\ ) = label(x_k)$$

This equality requires that the term with label(x_k) dominates the other terms. Therefore, we must have:

$$\frac{\text{coefficient of label}(x_i)}{\text{coefficient of label}(x_k)} \to 0 \ \forall \ i \in \{1, 2, ..., k-1, k+1, ..., m\}$$

For a general coefficient $e^{-\beta||x_i - t||^2}$, we need for all $i \in \{1, 2, ..., k-1, k+1, ..., m\}$, that:

$$\frac{e^{-\beta||x_i - t||^2}}{e^{-\beta||x_k - t||^2}} \to 0$$

$$e^{-\beta(||x_i-t||^2-||x_k-t||^2)} \to 0$$

Simplifying, we get:

$$-\beta(||x_i-t||^2-||x_k-t||^2) \to -\infty$$

$$\beta(||x_i-t||^2-||x_k-t||^2) \to \infty$$

Given all x_i except x_k are farther away from t, we have $||x_i-t||^2-||x_k-t||^2 > 0$. Further, unless the domain is infinite, we won't have $||x_i-t||^2-||x_k-t||^2 \to \infty$.

Let $||x_i-t||^2-||x_k-t||^2 = c$. We thus have:

$$c\beta \to \infty$$

$$\Rightarrow \beta \to \infty$$