COURSEWORK 1

COMP0078: Supervised Learning (2024/2025)

SN: 20082011, University College London

20 November 2024

Table of Contents

1 | PART I

1.1 Linear Regression

In this section, we treat about **Linear Regression**, an important topic in Machine Learning.

1.1.1 Question 1

The purpose of this question is to fit the data set $\{(1,3),(2,2),(3,0),(4,5)\}$ using the polynomial bases of dimension $k = 1,2,3,4$.

1.1.1.1 Question 1.(a)

To fit the data set with the polynomial basis, for data represented as:

$$\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ..., (\boldsymbol{x}_m, y_m)\} \tag{1.1}$$

where $\boldsymbol{x} = (x_1, ..., x_n)$ is a vector in $\mathbb{R}^n$ and y is a real number, our goal is to calculate the k-dimensional vector $\boldsymbol{w} = (w_1, ..., w_k)$ which will minimize the formula:

$$(\Phi\mathbf{w} - y)^T(\Phi\mathbf{w} - y) = \sum_{t=1}^{m}\left(y_t - \sum_{i=1}^{k} w_i\phi_i(\mathbf{x}_t)\right)^2 \tag{1.2}$$

To do so, we use the polynomial basis $P_1 = \{\phi_1(x) = 1, ..., \phi_k(x) = x^{k-1}\}$, which in our case is $\{\phi_1(x) = 1, \phi_2(x) = x, \phi_3(x) = x^2, \phi_4(x) = x^3\}$.

The plot represented in 1.1 represents the curves of the polynomials calculated using the basis P_1[1]. We can already identify the best model to be the polynomial base of dimension $k = 4$.

1.1.1.2 Question 1.(b)

We would like to find the equations to the curves fitted for $k = 1,2,3$.

We are given the equation corresponding to the fitted polynomial of dimension $k = 4$:

$$f_4(x) = -5 + 15.17x - 8.5x^2 + 1.33x^3 \tag{1.3}$$

Using the coefficients calculated in question (a), we obtain the following equations for the expressions of f_k[2]:

$$f_1(x) = 2.5 \tag{1.4}$$
$$f_2(x) = 0.4x + 1.5 \tag{1.5}$$
$$f_3(x) = 1.5x^2 - 7.1x + 9 \tag{1.6}$$

[1] All the codes can either be found in the appendix or in the joint Jupyter notebooks.
[2] The coefficients are rounded up to 2 decimal numbers.

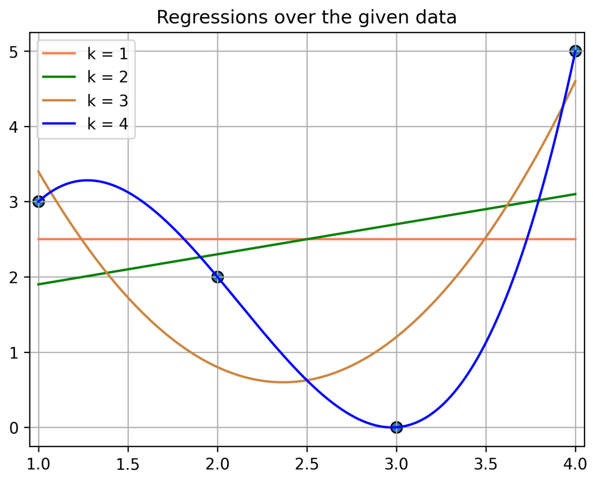


Figure 1.1: Regression on Data Set for Polynomial Bases of dimension $k = 1, 2, 3, 4$

1.1.1.3 Question 1.(c)

For each fitted curve, we obtained the following mean-squared errors depending on the dimension k, denoted as MSE_k:

$$\text{MSE}_1 = 3.25 \tag{1.7}$$
$$\text{MSE}_2 = 3.05 \tag{1.8}$$
$$\text{MSE}_3 = 0.80 \tag{1.9}$$
$$\text{MSE}_4 = 3.86\text{E-}26 \tag{1.10}$$
$$\tag{1.11}$$

1.1.2 Question 2

This part serves the purpose to illustrate the phenomena of *over-fitting*.

1.1.2.1 Question 2.(a)

We define:

$$g_\sigma(x) = \sin^2(2\pi x) + \epsilon \tag{1.12}$$

where the noise $\epsilon \sim \mathcal{N}(0, \sigma^2)$ is superimposed over the sinusoidal function in g_σ.
We define the data set:

$$S_{0.07,30} = \{(x_1, g_{0.07}(x_1)), ..., (x_{30}, g_{0.07}(x_{30}))\} \tag{1.13}$$

Question (i) We plotted the function $\sin^2(2\pi x)$ in the range $0 \leq x \leq 1$ with the points of data set S superimposed, represented in Figure 1.2.

Question (ii) We fitted the data set with polynomial bases of dimension $k \in \{2, 5, 10, 14, 18\}$. The obtained plot is represented in 1.5. As we can see, the plot is hard to read. This is due to some overfitting observed already on the training set at dimension $k = 18$. This is not representative of the general error when fitting the train data for $k = 18$, but it is rather interesting for us to notice the model

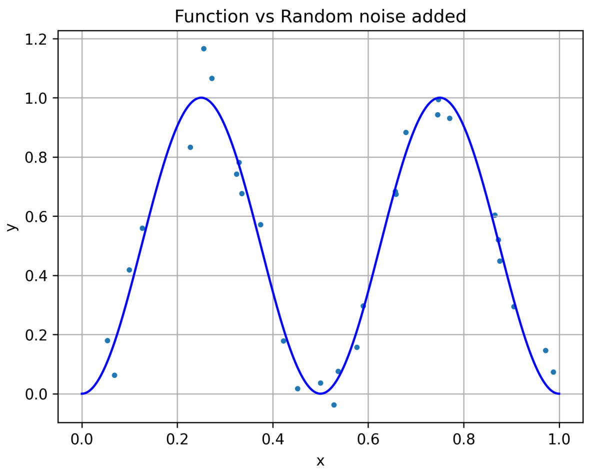


Figure 1.2: Function $\sin^2(2\pi x)$ and data set S

is depending a lot on the randomness introduced by the term ϵ. For the rest of the models fitted, the curve approximates pretty well the data.

Another example of fitted curves (with random new data) is displayed in 1.4. This time, we see data making more sense, and thus no overfitting for the training set.

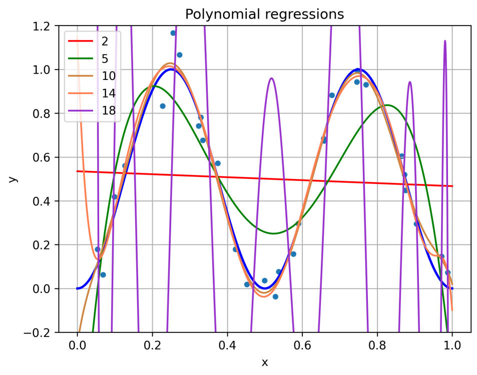


Figure 1.3: "Bad Quality" Data

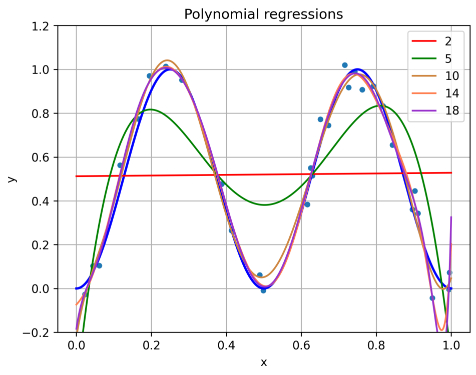


Figure 1.4: "Good Quality" Data

Figure 1.5: Curves fitted to the dataset for different k values; for two examples of random data

1.1.2.2 Question 2.(b)

We define the training error $te_k(S)$, defined by the MSE obtained when fitting polynomial functions of dimension k to the training data set S. By calculating these training errors, we obtain the plots represented in 1.6 and 1.7. In Figures 1.8 and 1.9 are represented the same but for data presenting better results. As expected, the "bad quality" data has a large MSE for $k = 18$, while the "good quality" data has a decreasing MSE as dimensions increase.

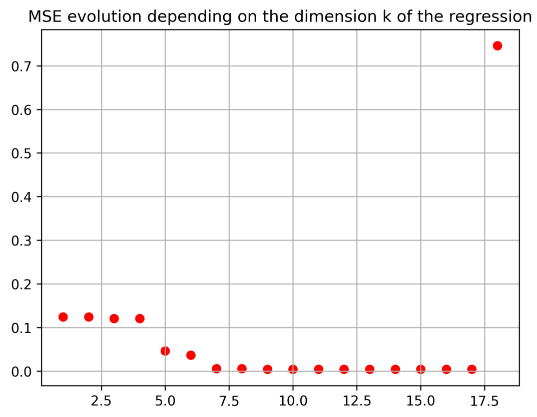


Figure 1.6: Training error te_k (MSE): : "Bad Quality" data

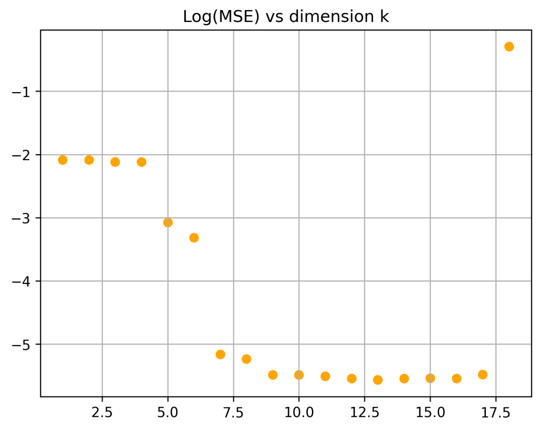


Figure 1.7: Natural Logarithm of te_k (MSE): "Bad Quality" data

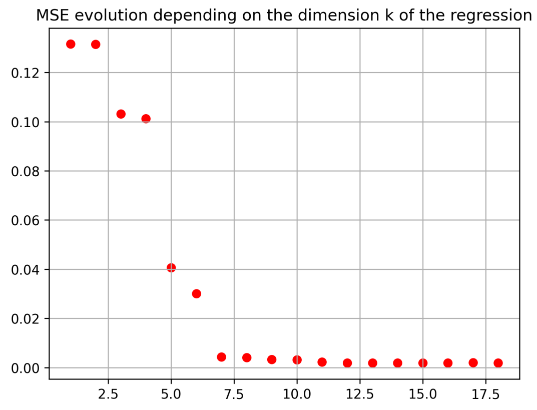


Figure 1.8: Training error te_k (MSE): "Good Quality" data

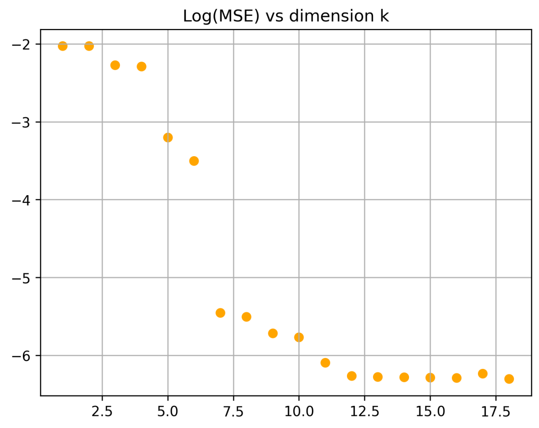


Figure 1.9: Natural Logarithm of te_k (MSE): : "Good Quality" data

Figure 1.10: Comparison of training errors and their natural logarithms for the dataset and new data.

1.1.2.3 Question 2.(c)

We now generate a test set, composed of 1000 points:

$$T_{0.07,1000} = \{(x_1, g_{0.07}(x_1)), \ldots, (x_{1000}, g_{0.07}(x_{1000}))\} \tag{1.14}$$

and we define $\text{tse}_k(S,T)$ as the test error (the MSE of the test set T, using the fitted polynomials of dimension k for data set S).

In Figure 1.11, the test set data is represented. Then, by using the fitted curves for the training data to estimate the test data, we obtain the curves shown in Figures 1.12 and 1.13.

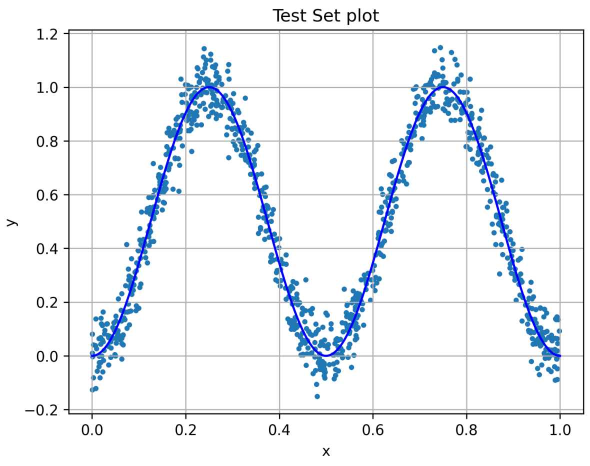


Figure 1.11: Test set data represented

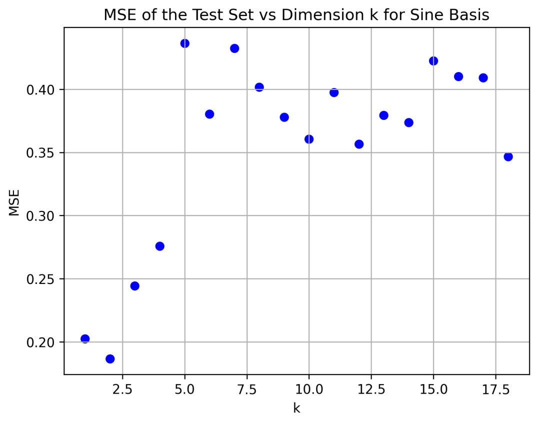


Figure 1.12: MSE of the Test Set

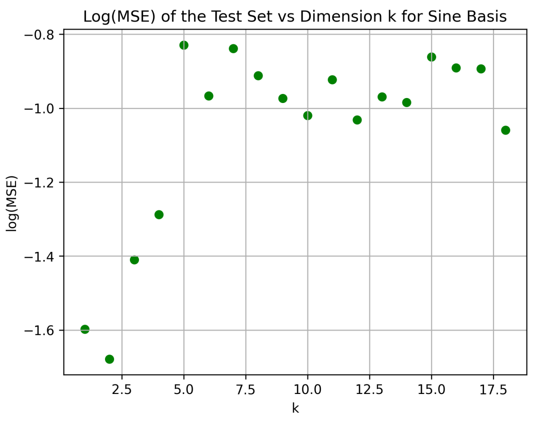


Figure 1.13: Log(MSE) of the Test Set

1.1.2.4 Question 2.(d)

We repeat the processes from the previous questions (b) and (c) over 100 runs, and averaged the results. The obtained data is shown in Figures 1.14 and 1.16, representing the averages for training and test sets respectively. Figures 1.15 and 1.17 depict the logarithm of these averages. As expected, it seems the average MSE for the training set is, on average, decreasing while that of the test set is increasing in an exponential manner (a straight curve in log representation).

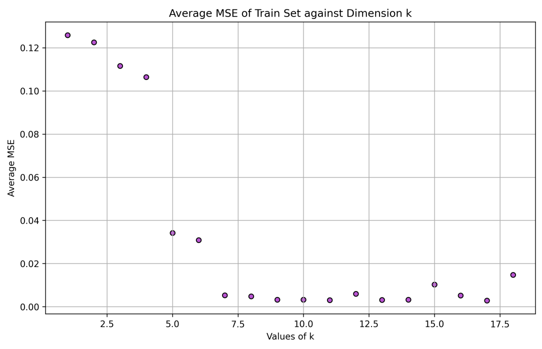


Figure 1.14: Averaged MSE of Training Set over 100 runs

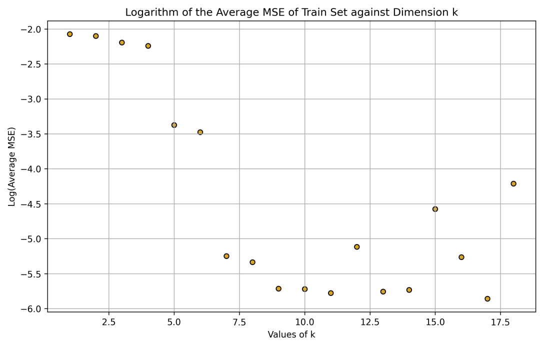


Figure 1.15: Logarithm of Averaged MSE of Training Set over 100 runs

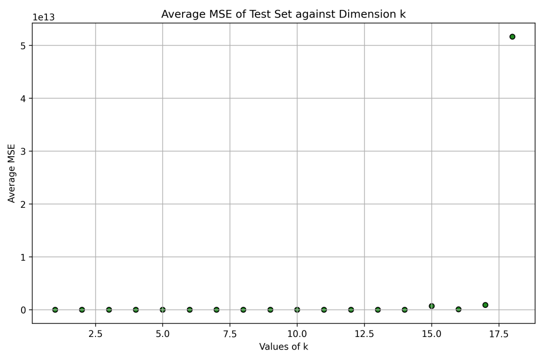


Figure 1.16: Averaged MSE of Test Set over 100 runs

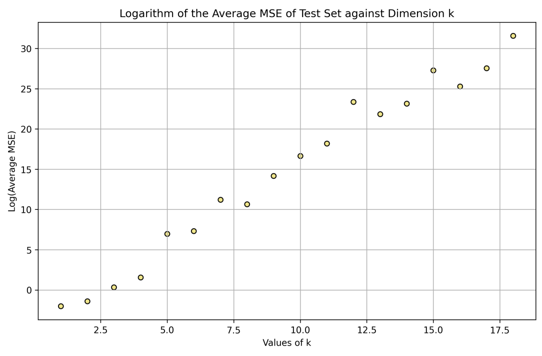


Figure 1.17: Logarithm of Averaged MSE of Test Set over 100 runs

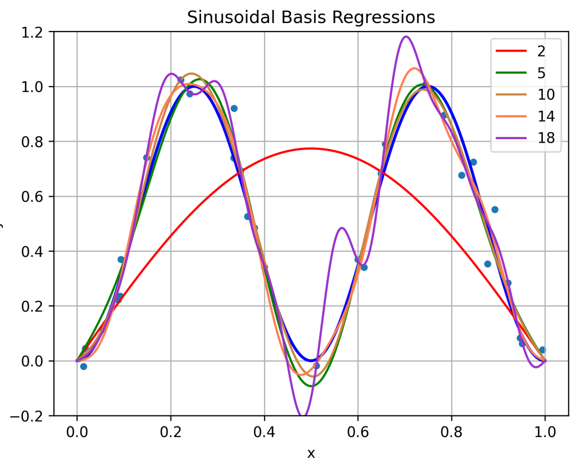


Figure 1.18: Regression using Sinusoidal Basis

1.1.3 Question 3

In this section, our goal was to use as new basis the following:

$$\{\sin(1\pi x), \sin(2\pi x), ..., \sin(k\pi x)\} \tag{1.15}$$

The difficulty resided in the inability to use the Vandermonde matrix in the python code, and thus having to rebuild ourselves our own matrix containing the $\sin(k\pi x)$ for each dimension k and input x.

By repeating the questions 2.(b) to 2.(d), we obtain the following results:

1. Figure 1.18 shows the obtained curves when fitting the data using the sinusoidal basis. Even at low dimensions, it seems the data was fitted with low error. This is because the original function over which the noise was added is a sinusoid.

2. Figures 1.19 and 1.20 show respectively the *mean-squared error* of the training set and the logarithm of the MSE of the training set when fitting sinusoidal basis. Compared to that of the polynomial basis with *"good quality" data* (Figures 1.8 and 1.9), we can observe 'less good' results, but less dependent on the dimension k (the data appears constant).

3. Figures 1.21 and 1.22 show the MSE for the Test Data fitted using the sinusoidal basis. We can observe, compared to that of the polynomial basis, the testing data has a low MSE and thus predicts with more accuracy than the polynomial basis the testing data.

4. Averaging the performances of the sinusoidal basis models fitted to the training and test data sets over 100 runs, we obtain the plots shown in Figure 1.26.

5. Taking the log of these averages, we obtain the plots displayed in Figure. These plots are easier to interpret than the ones from Figure 1.26. As we can see, in average, the test set fitting of the model has a performance slightly worse than that of the training data, but compared to the fitted curves for polynomial basis, we clearly have better results. There still seems to be overfitting obviously for higher dimensions k, due to the randomness introduced by the normal distribution but low compared to overfitting in polynomial basis.

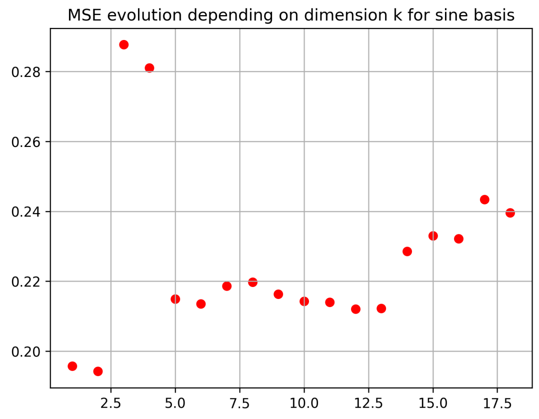


Figure 1.19: MSE of Training data using Sine Basis

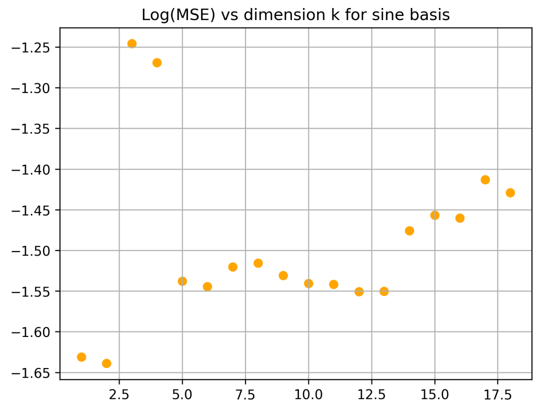


Figure 1.20: Log(MSE) of Training data using Sine Basis

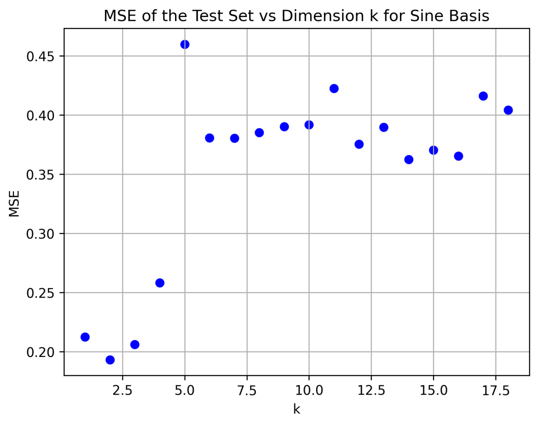


Figure 1.21: MSE of Test data using Sine Basis

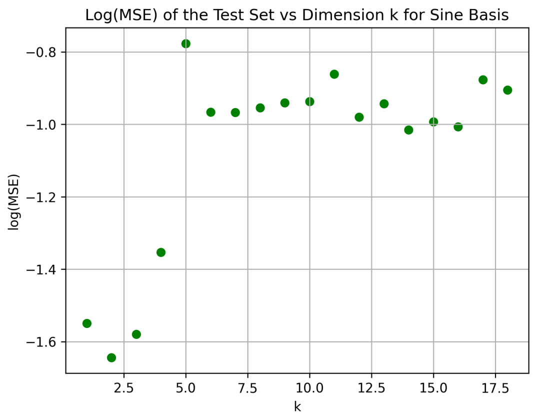
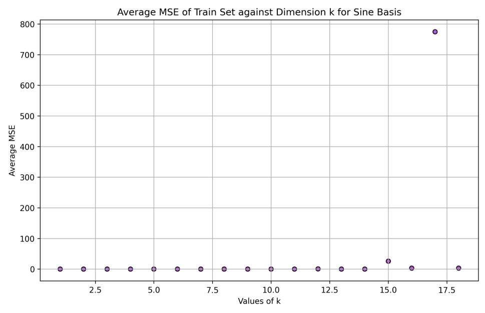


Figure 1.22: Log(MSE) of Test data using Sine Basis

Figure 1.23: Training and Test Results for Sinusoidal Basis



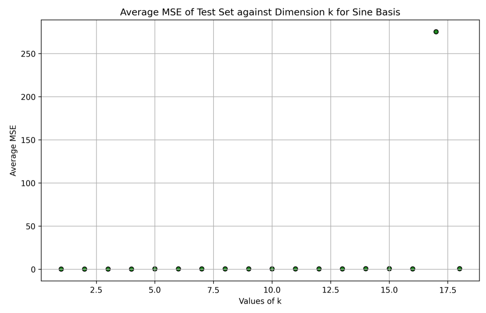


Figure 1.24: Train Set

Figure 1.25: Test Set

Figure 1.26: Average MSE over 100 runs for Sinusoidal Basis

1.2 Filtered Boston housing and kernels

In this section, we work on regressions to predict house prices in Boston through different methods.

1.2.1 Question 4

1.2.1.1 Question 4.(a)

The 'MEDV' column is the target data we want to predict. The first model we used to fit the dataset is a Naive Regression model, we consider it as the Baseline model. By generating a vector of **ones**, estimating the weights is comparable to calculating the mean of the training dataset. The obtained data is summarised in the table 1.1.

MSE of Train Dataset	MSE of Test Dataset
85.5273	82.2525
89.8753	73.5590
91.5707	70.0413
85.2202	83.1491
74.3401	104.7675
89.5198	74.1880
82.3891	88.8088
83.1239	87.1267
78.8620	95.8746
86.7034	79.8687
77.7355	98.3026
85.2413	82.9112
86.2899	80.7977
80.8245	91.7831
81.1635	91.1667
88.7343	75.7395
87.9955	77.5855
82.1255	89.1921
83.3750	86.6876
84.2246	84.9549

Table 1.1: Calculated MSE values of Train and Test datasets

By averaging these MSE, we obtain the following mean Mean Squared Errors for training and testing datasets respectively:

$$\text{MSE}_{train} = 84.24 \tag{1.16}$$

$$\text{MSE}_{test} = 84.94 \tag{1.17}$$

1.2.1.2 Question 4.(b)

After 20 runs, the average MSE of the train and test datasets are very close. The first factor from which we can interpret it is how the model does not relate the outputs of the data to their inputs. Therefore, it will approximately give the same value overall, because the weights in the model are simply the average of the training dataset. Therefore, since the draws of the train and test datasets are random, when performed a large number of times, the average of the MSE will be approximately the average value of the whole feature (which is the 'MEDV' column of the dataset, representing the Median value of owner-occupied homes in \$1000's, source: Boston Housing: Data Set Details). This can be seen as the Monte Carlo algorithm: an estimate, when performed a large enough number of times, can be seen as a good approximation of the true value (the average). This result can also be explained by the high bias and low variance of the Naive Regression model. The model simply calculates the average of the model and does not try to capture an interpretation of the model.

1.2.1.3 Question 4.(c)

For the second method, we performed linear regression using a single feature different from the target feature. The obtained averaged MSE obtained are displayed in Figure 1.29. As we can see, most of the features seem to have similar MSE (around ~ 70) for both the training and test sets, but two features seem to do better for the predictions of the model, the features of indexes 6 and 12 corresponding respectively to:

1. the 'RM' column : average number of rooms per dwelling.
2. the 'LSTAT' column : the % of lower status of the population.

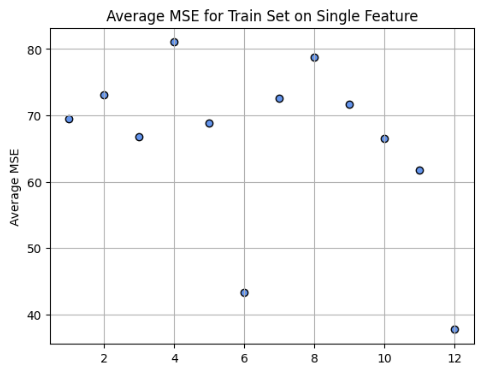


Figure 1.27: Train Set

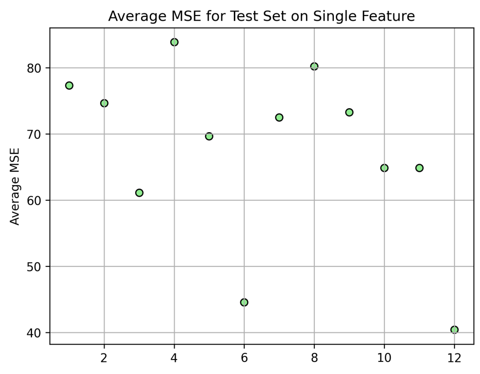


Figure 1.28: Test Set

Figure 1.29: Average MSE over 20 iterations for Single Feature Regression

1.2.1.4 Question 4.(d)

In this question, we studied the third type of model of which we wanted to observe the performance: the **Linear Regression using all attributes**.

The obtained results for the MSE of this model's training dataset and test dataset are displayed in 1.2.

Averaging these obtained MSE, we obtain the following means for training and testing datasets using the model:

$$\text{MSE}_{train} = 22.25 \tag{1.18}$$

$$\text{MSE}_{test} = 24.16 \tag{1.19}$$

Comparing it to all the previous models, we clearly observe a better performance using all the features to predict the target variable. Figure 1.32 summarize the foundings of Question 4.

MSE of Train Data	MSE of Test Data
23.6204	23.6204
24.6391	24.6391
22.5443	22.5443
20.7935	20.7935
25.8196	25.8196
23.7221	23.7221
19.4884	19.4884
25.5442	25.5442
22.2756	22.2756
22.5131	22.5131
19.5914	19.5914
19.9263	19.9263
22.4951	22.4951
20.7108	20.7108
17.9075	17.9075
19.8236	19.8236
23.5142	23.5142
21.9494	21.9494
23.3848	23.3848
24.6437	24.6437

Table 1.2: MSE values for Train and Test datasets using all features linear regression.

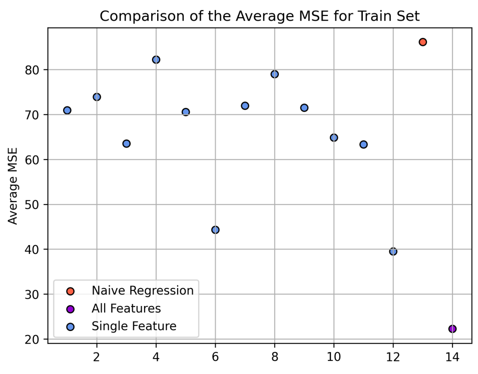


Figure 1.30: Train Set

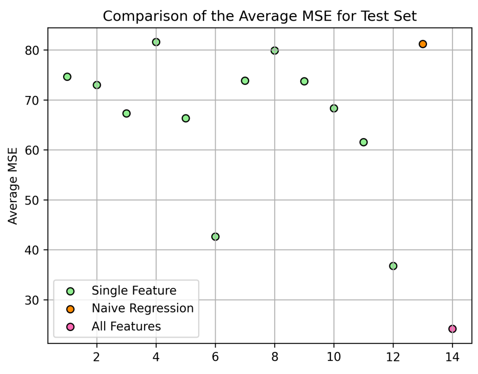


Figure 1.31: Test Set

Figure 1.32: Average MSE over 20 iterations for each model

1.3 Kernelised Ridge Regression

1.3.1 Question 5

In this section, we perform the kernel ridge regression over the data set using a Gaussian Kernel and evaluate its performance.

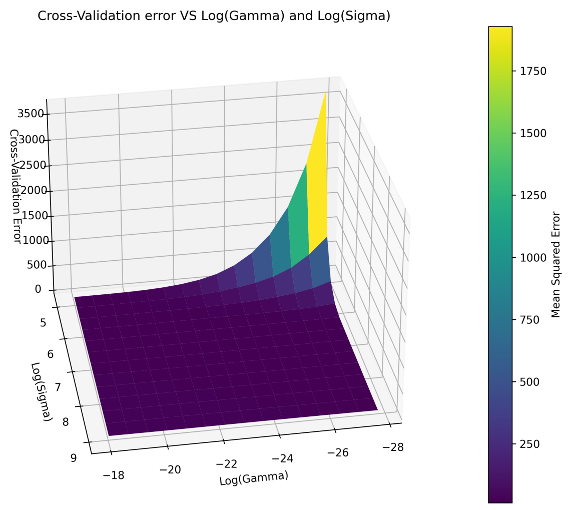


Figure 1.33: 3D surface plot of the Cross Validation

1.3.1.1 Question 5.(a)

The best parameters obtained from this step were:

$$\sigma_{best} = 724.08 \tag{1.20}$$

And

$$\gamma_{best} = 9.31\text{E-}10 \tag{1.21}$$

1.3.1.2 Question 5.(b)

By performing the cross-validation error on all parameters σ and γ given, and using a surface plot, we obtain the plot displayed in Figure 1.33. The plot shows the logs of the γ and σ values on the x and y axis, because we could not see to the naked eye the changes in the values depending on the parameters.

1.3.1.3 Question 5.(c)

By performing the kernel ridge regression using all features except the target feature, we obtain the following best parameters, training error and testing error:

$$\sigma_{best} = 724.08 \tag{1.22}$$

And

$$\gamma_{best} = 9.31\text{E-}10 \tag{1.23}$$

And the corresponding training and test errors obtained:

$$\text{Error}_{training} = 8.23 \tag{1.24}$$
$$\text{Error}_{testing} = 11.70 \tag{1.25}$$
$$\tag{1.26}$$

1.3.1.4 Question 5.(d)

By performing this step, the obtained comparison of results is summarised in Table 1.3. Figures 1.36 show the table's data displayed as a box-plot, which can be easier to interpret. What we can conclude is that the Kernel Ridge Regression model outperforms all the other models for the prediction of the houses price. However, on the training set, we can observe a large variance (almost larger than all the other models). The accuracy's of the model is hence variable depending on the trial.

Method	**MSE Train**	**MSE Test**
Naive Regression	85.93 ± 4.79	81.57 ± 9.58
Linear Regression (Attribute 1)	70.68 ± 4.51	74.70 ± 9.44
Linear Regression (Attribute 2)	73.06 ± 4.79	74.76 ± 9.69
Linear Regression (Attribute 3)	63.57 ± 4.44	67.22 ± 8.95
Linear Regression (Attribute 4)	80.23 ± 4.57	85.69 ± 9.17
Linear Regression (Attribute 5)	70.51 ± 4.01	66.24 ± 8.02
Linear Regression (Attribute 6)	43.07 ± 3.76	45.12 ± 7.54
Linear Regression (Attribute 7)	70.17 ± 4.97	77.48 ± 10.16
Linear Regression (Attribute 8)	79.29 ± 6.35	79.37 ± 12.84
Linear Regression (Attribute 9)	71.27 ± 4.89	74.33 ± 9.82
Linear Regression (Attribute 10)	65.12 ± 5.49	67.87 ± 11.24
Linear Regression (Attribute 11)	62.41 ± 4.43	63.72 ± 9.24
Linear Regression (Attribute 12)	37.67 ± 2.55	40.39 ± 5.33
Linear Regression (All Attributes)	21.69 ± 1.61	25.15 ± 3.41
Kernel Ridge Regression	8.34 ± 1.58	15.15 ± 12.44

Table 1.3: Mean Squared Error (MSE) results for train and test datasets using different methods.

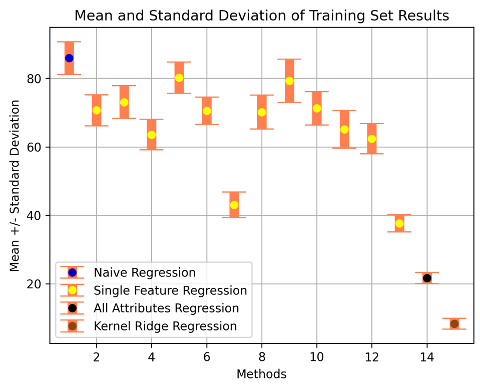


Figure 1.34: Train Set

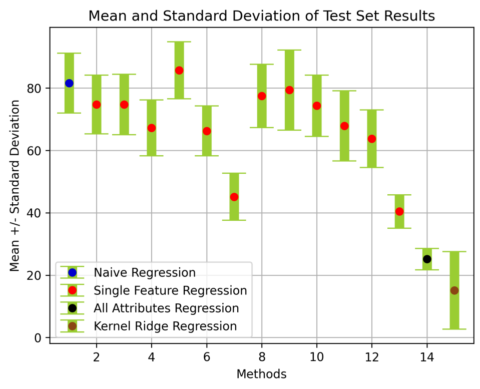


Figure 1.35: Test Set

Figure 1.36: Comparison of the Models Results (Mean and Standard Deviation)

2 | PART II

2.1 k-Nearest Neighbors

2.1.1 Question 6

By sampling 100 centers uniformly at random from $[0,1]^2$ with labels from $\{0,1\}$ (generating from the random process $p_{\mathcal{H}}$), the obtained results are shown in Figures 2.3 below. Figure 2.1 shows the sampled centers, while Figure 2.2 shows the results of the *k-NN* algorithm.

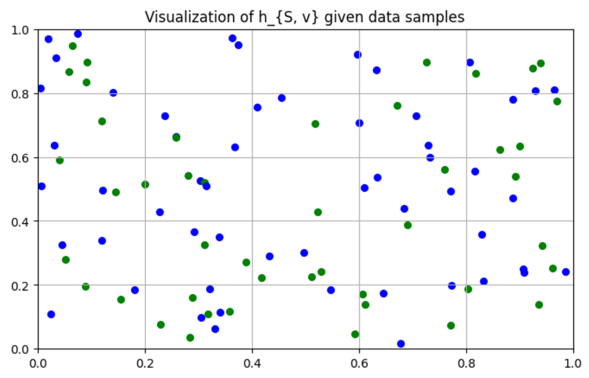


Figure 2.1: Sampled centers from $p_{\mathcal{H}}$

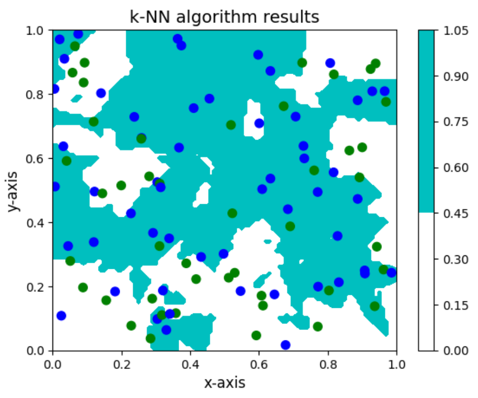


Figure 2.2: *k-NN* results obtained

Figure 2.3: *k-NN* algorithm results

2.1.2 Question 7

For this question, in the given time until the deadline, I could not manage to obtain decent results. Hence, only the random labeling function was made.

2.1.3 Question 8

Not Answered

3 | PART III

3.1 Question 9

We consider the following function:

$$K_c(\boldsymbol{x}, \boldsymbol{z}) = c + \sum_{i=1}^{n} x_i z_i \qquad \boldsymbol{x}, \boldsymbol{z} \in \mathbb{R}^n, c \in \mathbb{R} \tag{3.1}$$

3.1.1 Question 9.(a)

We would like to find the values of c for which the kernel function K_c is positive semidefinite.

We define the kernel K, such that its feature map ϕ is defined as $\phi : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{Y} \subset \mathcal{H}$ represents subset of the Hilbert Space with inner product $\langle \cdot, \cdot \rangle_{\mathcal{Y}}$. K is a positive semidefinite kernel for its feature map if and only if, for any $x, z \in \mathbb{R}^n$, K is equal to the inner product of their feature maps:

$$K(\boldsymbol{x}, \boldsymbol{z}) = \langle \phi(x), \phi(z) \rangle_{\mathcal{H}} \tag{3.2}$$

By looking at the equation of K_c, we see that for it to be an inner product of the feature map ϕ, we must have:

$$\langle \phi(x), \phi(z) \rangle_{\mathcal{H}} = c + \sum_{i=1}^{n} x_i z_i, \qquad x, z \in \mathbb{R}^n \tag{3.3}$$

Obviously, we can deduce the feature map ϕ's expression:

$$\phi(x) = (\sqrt{c}, x_1, ..., x_n)^T \tag{3.4}$$

And therefore, is defined as $\phi : \mathbb{R}^n \to \mathbb{R}^{n+1}$ since $c \in \mathbb{R}$.

Now, we need to prove the function K_c represents an inner product. For it to be an inner product must be:

1. Symmetric
2. Bilinear
3. Positive definite

Let's prove all these conditions are valid.

Symmetric For the inner product to be symmetric, we must have:

$$\langle \phi(x), \phi(z) \rangle_{\mathbb{R}^n} = \langle \phi(z), \phi(x) \rangle_{\mathbb{R}^n}$$

Since the expression is only made of sums and products of real numbers, the commutativity of the expression of K_c is obvious.

Bilinear For the inner product to be bilinear, we must have:

$$\langle \alpha\phi(x) + \beta\phi(y), \phi(z) \rangle = \alpha\langle \phi(x), \phi(z) \rangle + \beta\langle \phi(y), \phi(z) \rangle \qquad \forall x, y, z \in \mathbb{R}^n$$

We start with:

$$
\begin{aligned}
\langle \alpha\phi(x) + \beta\phi(y), \phi(z)\rangle &= \langle \alpha(\sqrt{c}, x_1, ..., x_n)^T + \beta(\sqrt{c}, y_1, ..., y_n)^T, (\sqrt{c}, z_1, ..., z_n)^T\rangle \\
&= \langle (\alpha\sqrt{c} + \beta\sqrt{c}, \alpha x_1 + \beta y_1, ..., \alpha x_n + \beta y_n)^T, (\sqrt{c}, z_1, ..., z_n)^T\rangle \\
&= (\alpha\sqrt{c} + \beta\sqrt{c})\sqrt{c} + \sum_{i=1}^{n}(\alpha x_i + \beta y_i)z_i \\
&= \alpha c + \beta c + \sum_{i=1}^{n}(\alpha x_i z_i) + \sum_{i=1}^{n}(\beta y_i z_i) \\
&= \alpha\left(c + \sum_{i=1}^{n} x_i z_i\right) + \beta\left(c + \sum_{i=1}^{n} y_i z_i\right) \\
&= \alpha\langle\phi(x), \phi(z)\rangle + \beta\langle\phi(y), \phi(z)\rangle
\end{aligned}
$$

Which proves the bilinearity of the function K_c.

Positive Semi-definiteness For the feature map ϕ to be positive semi-definite, it must respect:

$$\langle\phi(x), \phi(z)\rangle \geq 0 \qquad \forall x, z \in \mathbb{R}^n \tag{3.5}$$

Equivalent to:

$$c + \sum_{i=1}^{n} x_i z_i \geq 0 \qquad \forall x, z \in \mathbb{R}^n \tag{3.6}$$

It leads to two constraints:

1. $\forall x, z \in \mathbb{R}^n$, $c \geq -\sum_{i=1}^{n} x_i z_i$
2. But also, since 0 is an element of the set $\mathbb{R}^n$, then for $x = 0_n$ and/or $z = 0_n$ (where 0_n is the vector of size n containing only 0 as entries, we must have $c + \sum_{i=1}^{n} 0 \geq 0$ leading to $c \geq 0$

Therefore, the most restraining constraint being $c \geq 0$, for K_c to be a positive semidefinite kernel, it must respect the condition $c \in \mathbb{R}_+$.

3.1.2 Question 9.(b)

We consider the case where we use K_c as a kernel function with linear regression. We would like to study how c influences the solution.

Using the expression from question 5, we start from the dual optimization for alpha:

$$\alpha^* = \arg\min_{\alpha\in\mathbb{R}^\ell} \frac{1}{\ell}\sum_{i=1}^{\ell}\left(\sum_{j=1}^{\ell}\alpha_j K_{c,i,j} - y_i\right)^2 + \gamma\alpha^\top\mathbf{K_c}\alpha$$

Setting γ to 0, we get:

$$\alpha^* = \arg\min_{\alpha\in\mathbb{R}^\ell} \frac{1}{\ell}\sum_{i=1}^{\ell}\left(\sum_{j=1}^{\ell}\alpha_j K_{c,i,j} - y_i\right)^2 \tag{3.7}$$

And replacing with our expression for K_c:

$$\alpha^* = \arg\min_{\alpha\in\mathbb{R}^\ell} \frac{1}{\ell}\sum_{i=1}^{\ell}\left(\sum_{j=1}^{\ell}\alpha_j\left(c+\sum_{j=1}^{\ell}x_kx_j\right)-y_i\right)^2 \qquad x_i, x_j \in \mathbb{R}^n$$

$$= \arg\min_{\alpha\in\mathbb{R}^\ell} \frac{1}{\ell}\sum_{k=1}^{\ell}\left(\sum_{j=1}^{\ell}\alpha_j c+\sum_{j=1}^{\ell}\sum_{j=1}^{\ell}\alpha_j x_k x_j - y_k\right)^2$$

$$= \arg\min_{\alpha\in\mathbb{R}^\ell} \frac{1}{\ell}\sum_{k=1}^{\ell}\left(\left(\sum_{j=1}^{\ell}\alpha_j c\right)^2+2\left(\sum_{j=1}^{\ell}\alpha_j c\sum_{j=1}^{\ell}\sum_{j=1}^{\ell}(\alpha_j x_k x_j - y_k)\right)+\left(\sum_{j=1}^{\ell}\sum_{j=1}^{\ell}\alpha_j x_k x_j - y_k\right)^2\right)$$

Identifying the Gram matrix expression:

$$= \arg\min_{\alpha\in\mathbb{R}^\ell} \frac{1}{\ell}\sum_{k=1}^{\ell}\left(\left(\sum_{j=1}^{\ell}\alpha_j c\right)^2+2\left(\sum_{j=1}^{\ell}\alpha_j c\sum_{j=1}^{\ell}(\alpha_j K_{i,j} - y_k)\right)+\left(\sum_{j=1}^{\ell}\alpha_j K_{i,j} - y_k\right)^2\right)$$

And developing slightly:

$$= \arg\min_{\alpha\in\mathbb{R}^\ell}\left(\frac{1}{\ell}\ell c^2\left(\sum_{j=1}^{\ell}\alpha_j\right)^2+2\frac{1}{\ell}\sum_{k=1}^{\ell}\left(\sum_{j=1}^{\ell}\alpha_j c\sum_{j=1}^{\ell}(\alpha_j K_{i,j} - y_k)\right)+\frac{1}{\ell}\sum_{k=1}^{\ell}\left(\sum_{j=1}^{\ell}\alpha_j K_{i,j} - y_k\right)^2\right)$$

Leading to the final expression:

$$\alpha^* = \arg\min_{\alpha\in\mathbb{R}^\ell}\left(c^2\left(\sum_{j=1}^{\ell}\alpha_j\right)^2+2c\sum_{j=1}^{\ell}\alpha_j\left(\frac{1}{\ell}\sum_{k=1}^{\ell}\left(\sum_{j=1}^{\ell}(\alpha_j K_{i,j} - y_k)\right)\right)+\frac{1}{\ell}\sum_{k=1}^{\ell}\left(\sum_{j=1}^{\ell}\alpha_j K_{i,j} - y_k\right)^2\right) \tag{3.8}$$

In this expression, we can clearly identify the original expression of the dual optimization of alpha through kernel $K_{i,j}$. The other terms are the impact of c on the solution.

We can divide them into two cases:

1. $c = 0$
2. c is positive

For the first case, $c = 0$, there is no impact on the solution.

For the second case, when c is positive (and possibly large), the term $c^2\left(\sum_{j=1}^{\ell}\alpha_j\right)^2$ has more importance than the rest of the expression (since c is squared and not divided by $1/\ell$). The $\alpha \in \mathbb{R}^l$ minimizing the term is $\alpha = 0_l$, the vector containing only zeros. Therefore, the impact of c on the solution when it is positive is that it forces the solution to decrease to 0 because $\alpha = 0_l \Rightarrow \tilde{y} \simeq 0$.

3.2 Question 10

We suppose we perform a linear regression with a Gaussian kernel $K_\beta(\boldsymbol{x}, \boldsymbol{t}) = \exp\Big\{-\beta\|\boldsymbol{x} - \boldsymbol{t}\|^2\Big\}$ to train a classifier on a dataset $(\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_m, y_m) \in \mathbb{R}^n \times \{-1, 1\}$. We obtain a function $f : \mathbb{R}^n \to \mathbb{R}$ which is of the form $f(\boldsymbol{t}) = \sum_{i=1}^m \alpha_i K_\beta(\boldsymbol{x}_i, \boldsymbol{t})$.

The corresponding classifier is then $\text{sign}(f(\boldsymbol{t}))$. This classifier depends on the parameter β selected for the kernel. We would like to know in what scenario the chosen β enables the trained linear classifier to simulate a 1-Nearest Neighbor Classifier trained on the same dataset.

Let's consider the kernelized least-squares regression $\boldsymbol{\alpha = K^{-1}y}$. In this expression, the entries for each i and j of the matrix $\boldsymbol{K}$ are the evaluation of the Gaussian kernel $K_\beta(\boldsymbol{x}_i, \boldsymbol{x}_j)$ for the entry points. As β becomes very large ($\beta \gg 0$), the exponential values will decrease to 0 faster when x_i and x_j are far away and grow to one when they are close:

$$x_i \neq x'_j \Rightarrow K_\beta(x_i, x'_j) \approx 0 \tag{3.9}$$

And

$$x_i \approx x'_j \Rightarrow K_\beta(x_i, x'_j) = e^{-\beta * 0} \approx 1 \tag{3.10}$$

Hence, when $\beta \gg 0$, $\boldsymbol{K} \approx \boldsymbol{I}$, where $\boldsymbol{I}$ is the identity matrix. But since the inverse of the identity matrix is itself, then we can assume:

$$\boldsymbol{K} \approx \boldsymbol{K}^{-1} \tag{3.11}$$

And therefore:

$$\boldsymbol{\alpha} = \boldsymbol{K}^{-1}\boldsymbol{y} \approx \boldsymbol{I}\boldsymbol{y} = \boldsymbol{y} \tag{3.12}$$

Therefore, the expression for the function $f(\boldsymbol{t})$, since it is given by:

$$f(\boldsymbol{t}) = \sum_{i=1}^{m} \alpha_i K_\beta(\boldsymbol{x}, \boldsymbol{t}) \tag{3.13}$$

Becomes:

$$f(\boldsymbol{t}) = \sum_{i=1}^{m} y_i K_\beta(\boldsymbol{x}, \boldsymbol{t}) \tag{3.14}$$

The classifier $\text{sign}(f(\boldsymbol{t}))$ for each element, since they take values $y \in \{-1, 1\}$, becomes:

1. For $y_i = 1$:
$$\text{sign}(\alpha_i K_\beta(\boldsymbol{x}, \boldsymbol{t})) \approx \text{sign}(y_i K_\beta(\boldsymbol{x}, \boldsymbol{t})) = 1 \tag{3.15}$$

2. For $y_i = -1$:
$$\text{sign}(\alpha_i K_\beta(\boldsymbol{x}, \boldsymbol{t})) \approx \text{sign}(y_i K_\beta(\boldsymbol{x}, \boldsymbol{t})) = -1 \tag{3.16}$$

This classifier hence returns the labels of the dataset. Therefore, when t approaches a value x_i in the data set, the value of $\alpha_i K_\beta(x_i, t)$ will have the biggest weight due to the Gaussian Kernel impact, and therefore it will approximate a 1-Nearest Neighbor Classifier.

Hence, naming the dataset $x_{data} = \boldsymbol{x} \backslash x_i$, we have:

$$|\alpha_i K_\beta(x_i, t)| > \left| \sum_{x_{data}} \alpha_{data} K_\beta(x_{data}, t) \right| \tag{3.17}$$

And therefore, our classifier is an approximation of a 1-Nearest Neighbor Classifier.