Supervised Learning (COMP0078) – Coursework 1

Candidate Number: 18145066, 19019509

November 15, 2022

We implemented the answers for each question as single functions in "CW1.py". To see the result for each question, just simply execute the "run.ipynb" Jupyter notebook. "CW1" is imported in the notebook. We wrote the helper functions separately in "regressionFunction.py", "lossFunction.py" and "knnFunctions.py", which are imported correspondingly in "CW1.py".

1 Part 1

1.1 Linear Regression

Exercise 1: For each of the polynomial bases of dimension k = 1,2,3,4 fit the data set $\{(1,3),(2,2),(3,0),(4,5)\}$.

a) Figure 1 below shows the four different curves corresponding to each fit (dimension $k = 1, 2, 3, 4$) over the four data points.

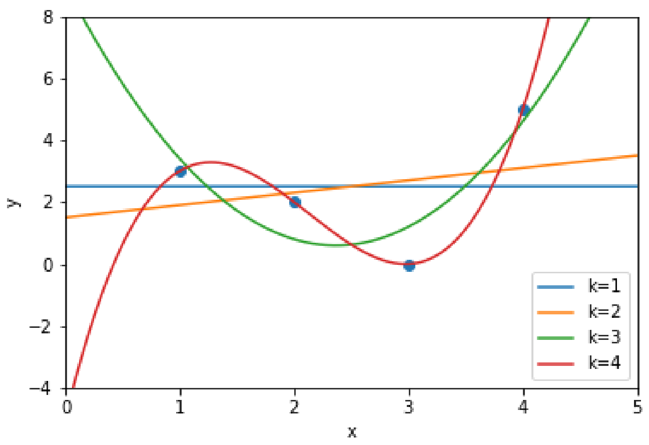

Figure 1: Different fitted curves for k = 1,2,3,4

b) The equations corresponding to the curves fitted for k = 1,2,3,4.

* When $k = 1$, $y = 2.5$
* When $k = 2$, $y = 1.5 + 0.4 \cdot x$
* When $k = 3$, $y = 9 - 7.1 \cdot x + 1.5 \cdot x^2$

* When $k = 4$, $y = -5 + 15.17 \cdot x - 8.5 \cdot x^2 + 1.33 \cdot x^3$

c) For each fitted curve k = 1, 2, 3, 4 the mean square error is:

* When $k = 1$, $mse = 3.25$
* When $k = 2$, $mse = 3.05$
* When $k = 3$, $mse = 0.80$
* When $k = 4$, $mse = 7.12 \cdot 10^{-27} \approx 0$

Exercise 2: In this part we will illustrate the phenomena of *overfitting*.

a) (i) Figure 2 shows the plot of the function $\sin^2(2\pi x)$ superimposed with the data points generated from $\sin^2(2\pi x) + \epsilon$, where $\epsilon \sim \mathcal{N}(0, 0.07)$.

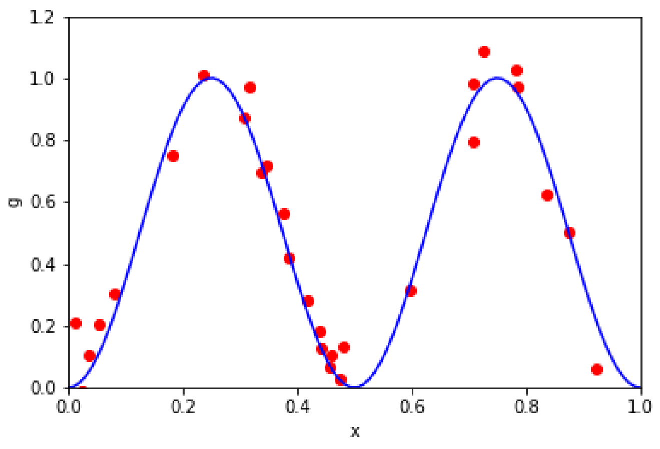

Figure 2: Plot of the curve and the sampled data

(ii) Fit the data set with a polynomial bases of dimension $k = 2, 5, 10, 14, 18$ plot each of these 5 curves superimposed over a plot of data points, shown in figure 3

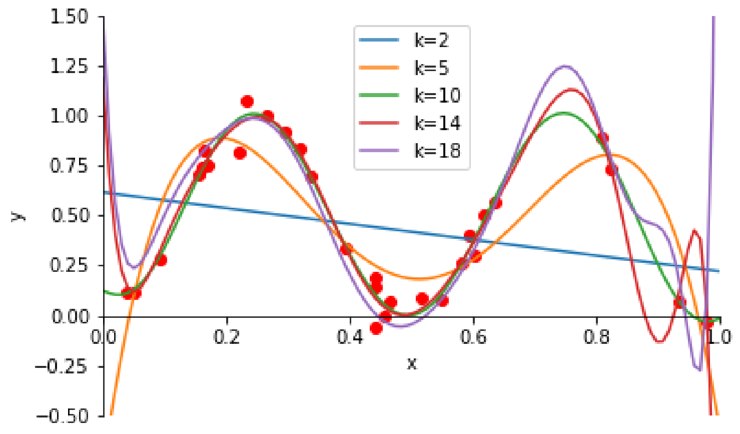

Figure 3: 5 fitted curves for k = 2,5,10,14,18

b) Let the training error $te_k(S)$ denote the MSE of the fitting of the data set S with polynomial basis of dimension k. The natural $log(ln)$ of the training error versus the polynomial dimension $k = 1, ..., 18$ (this should be a decreasing function) is shown in figure 4. Pseudo-inverse is used to make sure the function is decreasing.

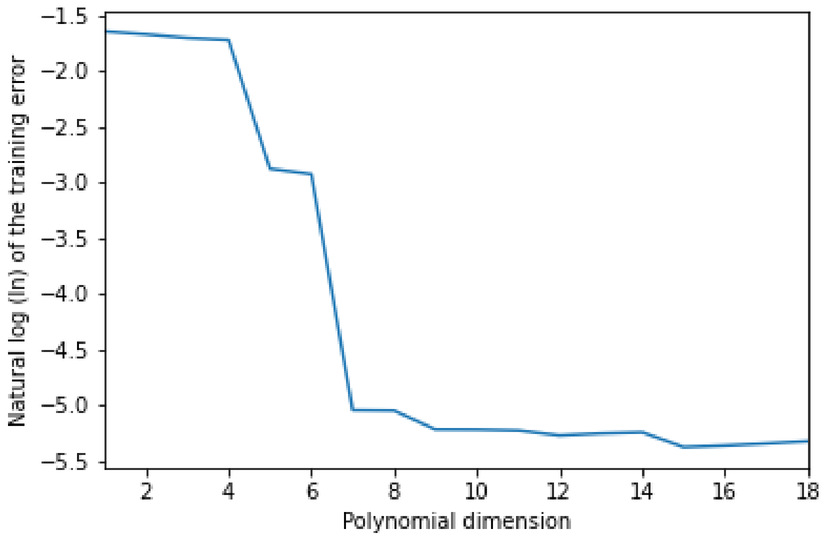

Figure 4: Natural log of the training error against k

c) Generate a test set T of a thousand points, the ln of the test error versus the polynomial dimension $k = 1, ..., 18$ is illustrated in figure 5. Unlike the training error this is not a decreasing function. This is the phenomena of *overfitting*.

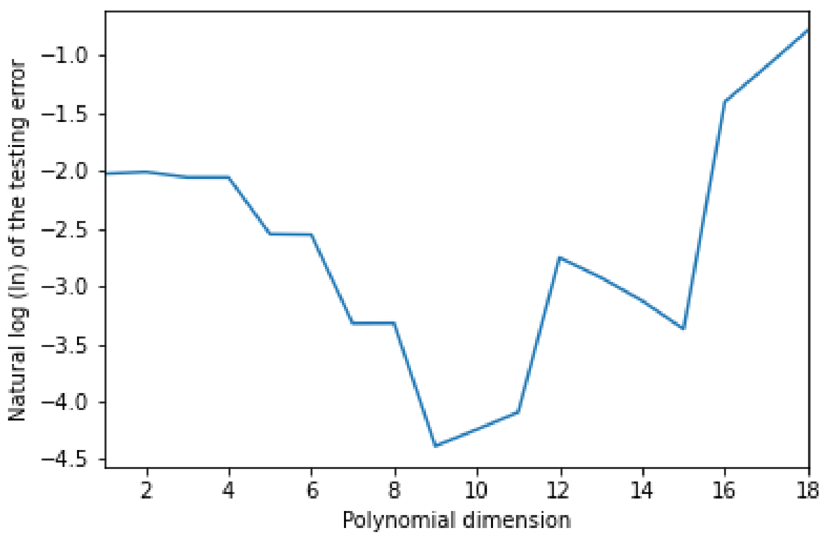

Figure 5: Natural log of the testing error against k

d) For any given set of random numbers we will get slightly different training curves and test curves. It is instructive to see these curves smoothed out. For this part repeat items (*b*) and (*c*) but instead of plotting the results of a single "run", the average results of a 100 runs is plotted as shown in figure 6 .

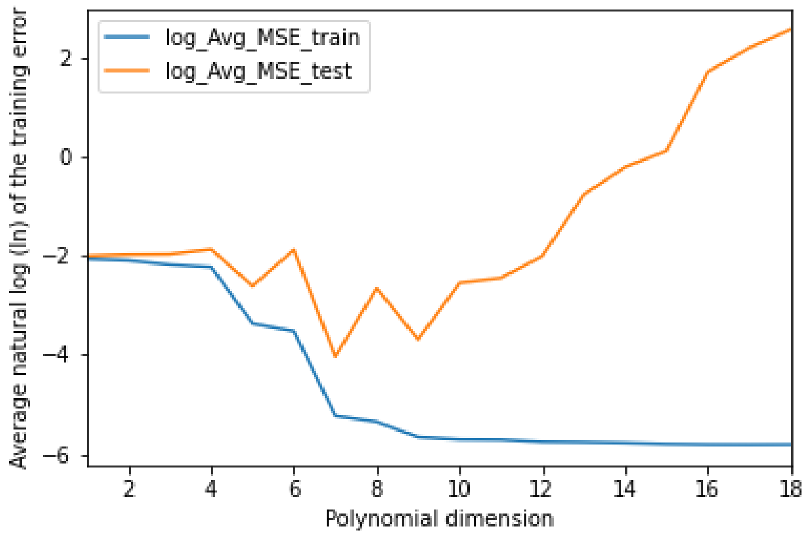

Figure 6: The average natural log of the training and testing error against k, over 100 runs

Exercise 3: Now use basis (for $k = 1, \dots, 18$)

$$\sin(1\pi x), \sin(2\pi x), \sin(3\pi x), \dots, \sin(k\pi x) \tag{1.1}$$

Repeat the experiments in 2 (b-d) with the above basis.

- Repeat for question *b* in exercise 2 with the new basis is shown in figure 7 (this should be a decreasing function as well):

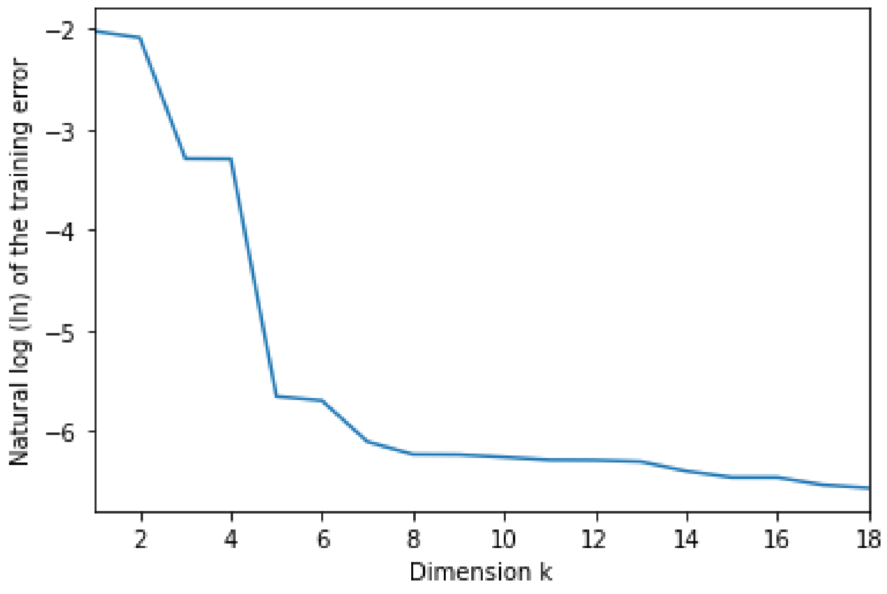

Figure 7: Natural log of the training error against k, for the new basis

- Repeat for question *c* in exercise 2 with the new basis is shown in figure 8. Unlike the training error this is not a decreasing function. This is the phenomena of *overfitting*.

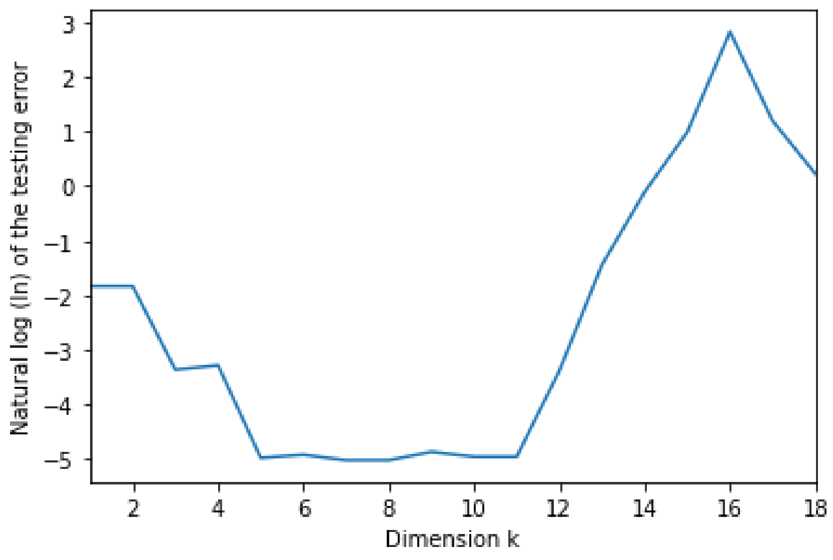

Figure 8: Natural log of the testing error against k, for the new basis

- Repeat for question *c* in exercise 2 with the new basis is shown in figure 9.

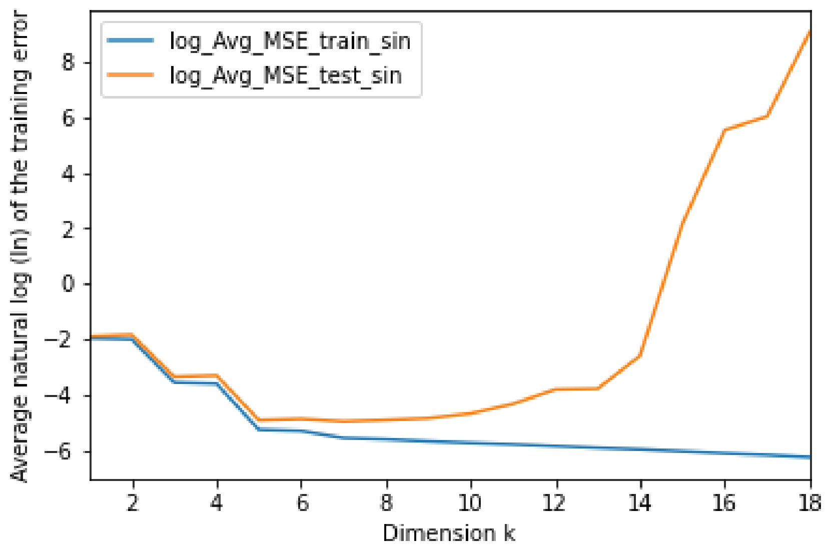

Figure 9: The average natural log of the training and testing error against k, over 100 runs for the new basis

1.2 Filtered Boston housing and kernels

Exercise 4: "Baseline versus full linear regression."

The training set should be $\frac{2}{3}$, and the test set should be $\frac{1}{3}$, of our data in (a)-(c). In the following average your results over 20 runs (each run based on a different ($\frac{2}{3}$, $\frac{1}{3}$) random split).

a) The first trial is Naive Regression. We create a vector of ones that is the same length as the training set and another one for test set. Using these vectors, we will fit a constant function to the data. Apply linear regression to the training data. The average MSE on the training and test sets over 20 runs are shown below:

- Average MSE over 20 runs on the training set: 83.54482741763243
- Average MSE over 20 runs on the test set: 86.59723654404692

b) The constant function we got from the linear regression is equivalent to calculate the mean value of training label y (The thirteen column in our dataset which we used to make prediction).

c) The second trial is to perform linear regression with each of the single attributes. The average MSE for each attribute over 20 runs on both training and test data is shown in Table 1 below:

Attribute	MSE on training	MSE on test
1(CRIM)	71.93729	72.29132
2(ZN)	73.17931	74.45896
3(INDUS)	65.29423	63.84836
4(CHAS)	81.82769	82.50602
5(NOX)	69.22741	68.89388
6(RM)	43.84079	43.57056
7(AGE)	72.69602	72.32192
8(DIS)	79.19020	79.51359
9(RAD)	72.54919	71.63659
10(TAX)	66.57561	64.87006
11(PTRATIO)	62.88252	62.57274
12(LSTAT)	38.76024	38.19612

Table 1: The average MSE for each attribute on both training and test data

d) The third trial would be a linear regression using all of the 12 attributes. The same as what we have done before, average MSE on both training and test data over 20 runs are calculated. Moreover, it is obvious that the MSE is the lowest when all attributes are used to make a prediction, indicating that all-attribute prediction provides the best performance among the models trained previously.

- Average MSE over 20 runs on the training set: 21.55682335614272.
- Average MSE over 20 runs on the test set: 25.471213004487346.

1.3 Kernelised ridge regression

Exercise 5: In this exercise we will perform kernel ridge regression (KRR) on the data set using the Gaussian kernel,

$$\mathbf{K}(x_i, x_j) = \exp\left(-\frac{\|x_i, x_j\|^2}{2\sigma^2}\right) \tag{1.2}$$

For this question, we hold out 2/3 of data for training and report the test results on the remaining 1/3.

a) Firstly, we created a vector of γ values $\left[2^{-40}, 2^{-39}, \dots, 2^{-26}\right]$ and a vector of σ values $\left[2^{7}, 2^{7.5}, \dots, 2^{12.5}, 2^{13}\right]$. Then, we performed kernel ridge regression on the training set using five-fold cross-validation to choose among all pairing of the values of γ and σ. The best pair of γ and σ is shown below in figure 10:

- Best γ is : $\gamma = 2^{-28}$
- Best σ is : $\sigma = 2^{9.0}$

b) Plot of the "cross-validation error" for all the pairs of γ and σ is shown below:

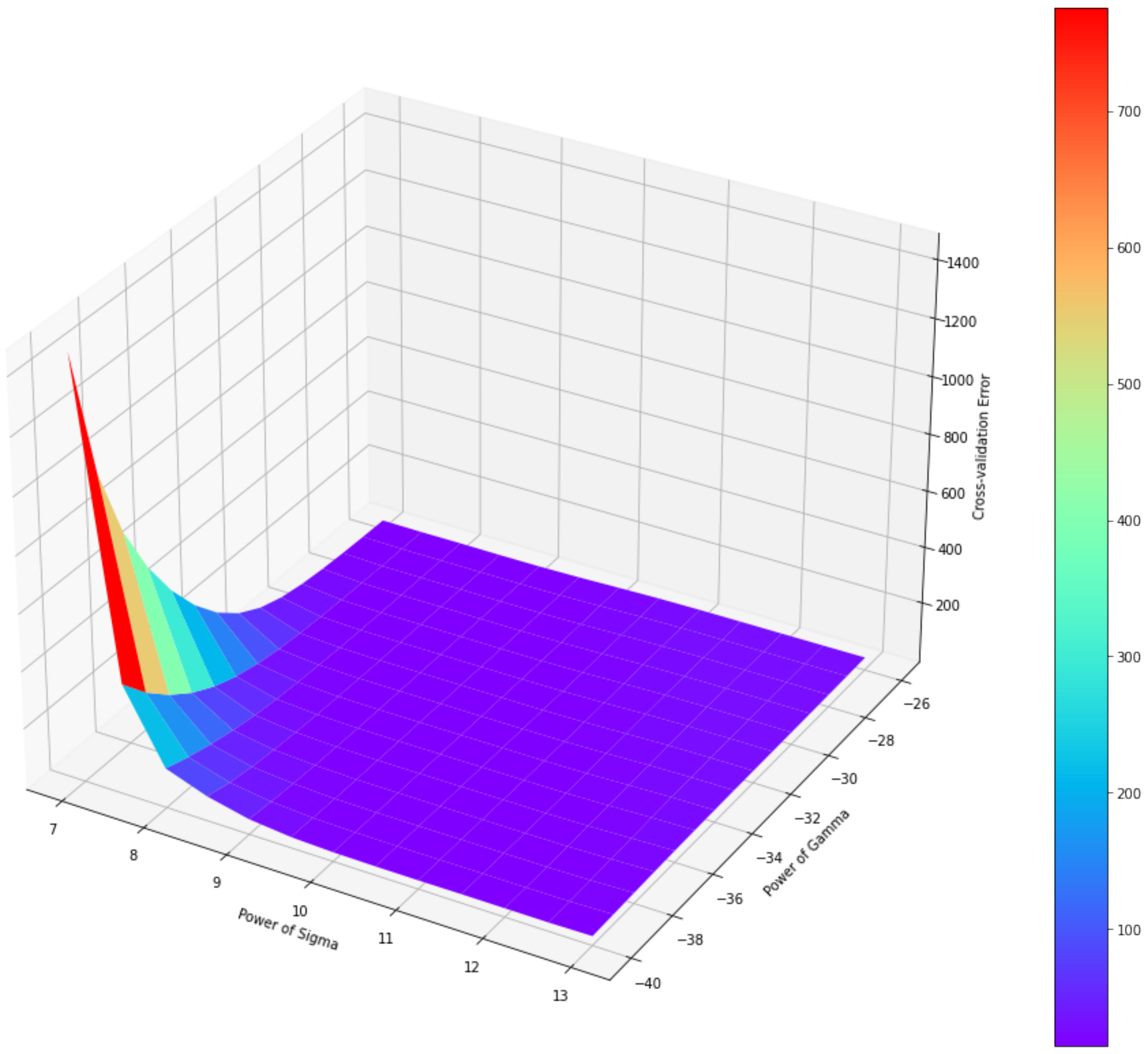

Figure 10: "cross-validation error" for all the pairs of γ and σ

c) Based on the best pair of γ and σ, the MSE on the training and test sets are shown below:

- MSE on the training set: 8.165577
- MSE on the test set: 11.680384

d) Repeat "exercise 4a,c,d" and "exercise 5a,c" over 20 random $(\frac{2}{3}, \frac{1}{3})$ splits of your data. The train/test error and the standard deviations (σ') of the train/test errors are calculated and these results are summarised in the table 2 below.

Method	Train MSE	Test MSE
Naive Regression	84.2061 ± 3.9440	84.9988 ± 7.8935
Linear Regression (attribute 1)	69.7561 ± 4.8745	76.2763 ± 10.1905
Linear Regression (attribute 2)	72.4058 ± 4.6514	76.0150 ± 9.4874
Linear Regression (attribute 3)	63.1650 ± 5.2693	68.1228 ± 10.5815
Linear Regression (attribute 4)	79.8378 ± 4.5486	86.2715 ± 9.1446
Linear Regression (attribute 5)	67.1420 ± 4.7020	73.0608 ± 9.5597
Linear Regression (attribute 6)	43.4027 ± 4.2660	44.4116 ± 8.5197
Linear Regression (attribute 7)	70.4787 ± 5.2598	76.7604 ± 10.7716
Linear Regression (attribute 8)	77.1412 ± 5.1438	83.6288 ± 10.6319
Linear Regression (attribute 9)	70.3114 ± 5.2214	76.1826 ± 10.6471
Linear Regression (attribute 10)	64.0475 ± 5.3005	69.9427 ± 10.7714
Linear Regression (attribute 11)	62.0385 ± 3.6769	64.3711 ± 7.6658
Linear Regression (attribute 12)	37.6038 ± 2.1734	40.5277 ± 4.4442
Linear Regression (all attributes)	22.0660 ± 2.3320	24.3357 ± 4.9849
Kernel Ridge Regression	7.5396 ± 1.4890	12.3232 ± 2.0179

Table 2: Train MSE and Test MSE for each method over 20 runs

2 Part 2

2.1 k-Nearest Neighbors

2.1.1 Generating the data

Exercise 6: A visualisation of an randomly generated $h_{S,v}$ is shown in figure 11. The white area is the mapping to 0 and the green is the mapping to 1, the corresponding centers are blue and red according to our code.

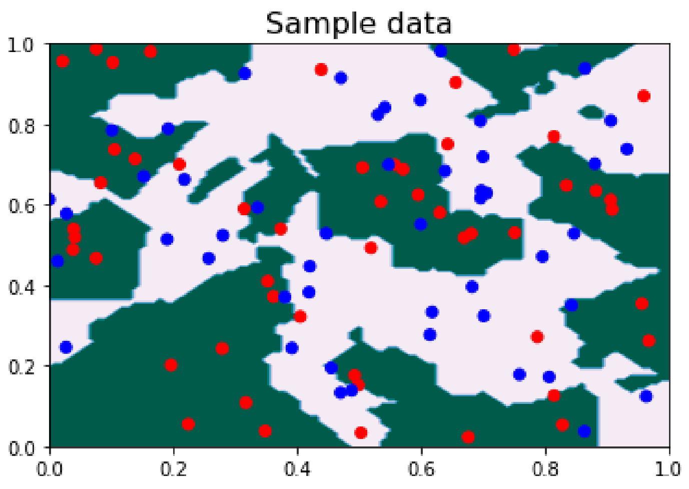

Figure 11: visualization of a randomly generated h, where $|S| = 100$ and $v = 3$

2.1.2 Estimated generalization error of k-NN as a function of k

Exercise 7:

a) Figure 12 shows the relationship between the generalization error and the number of nearest neighbors k from one to fifty.

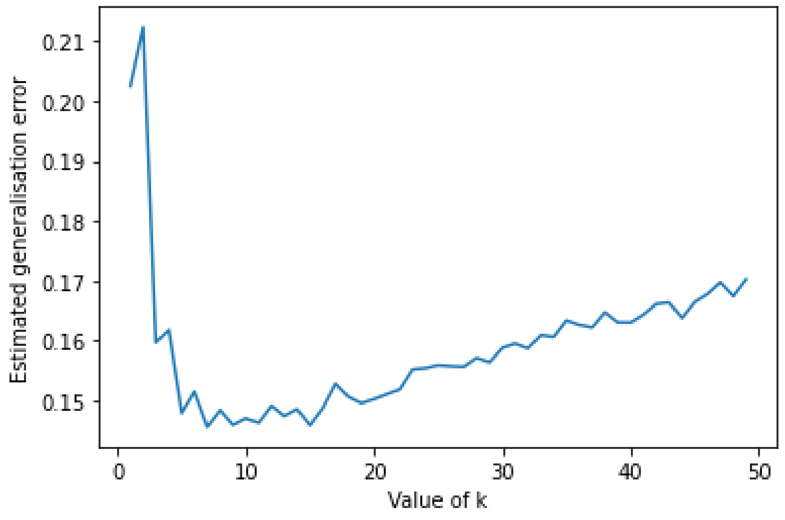

Figure 12: plotted curve for generalization error and k in knn

b) The generalization error experienced a steep downward trend in the beginning. This is because for fixed m training points, when k is small, $\frac{m}{k}$ is large, and the model is likely to overfit and have a high variance. Until around $k = 15$, $\frac{m}{k}$ reaches a optimal value. Then the error starts to gradually increase as $\frac{m}{k}$ becomes smaller and the model starts to underfit.

In addition, the graph looks unsmooth, this is likely because the k numbers that are even may introduce a random selection of the outcome, which results every even k having an error higher than its neighboring k values.

2.1.3 Determine the optimal k as a function of the number of training points (m)

Exercise 8:

a) Figure 13 shows the relationship between the average optimal k in 100 iterations against the number of training point used, m.

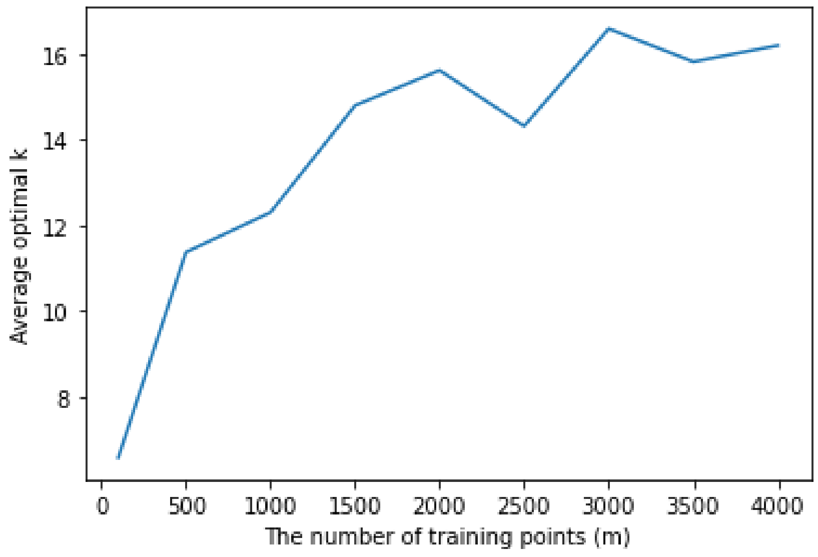

Figure 13: Relationship between the average optimal k and m

b) The graph roughly shows that the optimal k value increases as the number of training point increases. However, the rate of increase decays as m gets larger. This is because, initially, as m grows, k must also increase to maintain an optimal ratio. As k is already sufficiently large, the classification region does not need to expand any more because it adequately represents the local distribution for each area, thus the need for growth is less extreme for k so the curve start to flatten.

3 Part 3

Exercise 9:

a) When $c \geq 0$, the function $K_c(\boldsymbol{x}, \mathbf{z}) := c + \sum_{i=1}^{n} x_i z_i$ is a positive semidefinite kernel.

Proof:
To prove a kernel function is positive semidefinite, the following statement must be hold for $\forall n \geq 1, \forall(a_1, \dots a_n) \in \mathbb{R}^n, \forall(x_1, \dots, x_n) \in \mathcal{X}^n$ (The kernel matrix is a positive semidefinite matrix):

$$\sum_{i=1}^{n} \sum_{j=1}^{n} a_i a_j k\left(x_i, x_j\right) \geq 0 \tag{3.1}$$

Each entry in the kernel matrix of $K_c(\boldsymbol{x}, \mathbf{z})$ could be written as for $1 \leq i, j \leq n$, and $x_i, x_j \in \mathbb{R}^n$:

$$K_{cij}\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) = c + \boldsymbol{x}_i^T \boldsymbol{x}_j \tag{3.2}$$

Then, we could substitute (3.2) into the left hand side of formula (3.1):

$$\begin{aligned} \sum_{i=1}^{n} \sum_{j=1}^{n} a_i a_j K_{cij}\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) &= \sum_{i=1}^{n} \sum_{j=1}^{n} a_i a_j \left(c + \boldsymbol{x}_i^T \boldsymbol{x}_j\right) \\ &= \sum_{i=1}^{n} \sum_{j=1}^{n} a_i a_j c + \sum_{i=1}^{n} \sum_{j=1}^{n} a_i a_j \boldsymbol{x}_i^T \boldsymbol{x}_j \\ &= \|\mathbf{a}\|^2 c + \left\| \sum_{i=1}^{n} a_i \boldsymbol{x}_i \right\|^2 \end{aligned} \tag{3.3}$$

It is obvious that $\|\mathbf{a}\|^2$ and $\left\|\sum_{i=1}^{n} a_i \boldsymbol{x}_i\right\|^2$ are greater than or equal to 0. Therefore, to guarantee $\sum_{i=1}^{n} \sum_{j=1}^{n} a_i a_j K_c\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) = \|\mathbf{a}\|^2 c + \left\|\sum_{i=1}^{n} a_i \boldsymbol{x}_i\right\|^2 \geq 0$, $c \geq 0$ must be satisfied to make sure $\|\mathbf{a}\|^2 c \geq 0$.
On the other hand, if $c < 0$, then $\|\mathbf{a}\|^2 c$ would be smaller than 0, which would result a negative number for formula (3.3). Then, it would not satisfy the condition specified in the formula (3.1) for a positive semidefinite kernel.
Therefore, when $c \geq 0$, $K_c(\boldsymbol{x}, \mathbf{z})$ is a positive semidefinite kernel.

b) Based on the proof from last question, we have $c \geq 0$ to make the $K_c(\boldsymbol{x}, \mathbf{z})$ a positive semidefinite kernel. Then we consider the influence of c under two scenarios, $c > 0$ and $c = 0$, respectively.
When $c > 0$, it adds a bias to our regression function and the value of c would not influence the final solution. Specifically, given a test point t, the regression function to make prediction of it could be written as the formula (3.4) and it is obvious that it adds a bias term to the function.

$$\begin{aligned} f(\mathbf{t}) &= \sum_{i=1}^{n} \alpha_i K_c(\boldsymbol{x}_i, \mathbf{t}) = \sum_{i=1}^{n} \alpha_i (c + \boldsymbol{x}_i^T t) \\ &= \sum_{i=1}^{n} \alpha_i c + \sum_{i=1}^{n} \alpha_i \boldsymbol{x}_i^T t \end{aligned} \tag{3.4}$$

When $c = 0$, it backs to the standard linear regression solution because the formula (3.4) becomes $\sum_{i=1}^{n} \alpha_i \boldsymbol{x}_i^T t$.

Exercise 10:
Given the Gaussian kernel $K_\beta(\boldsymbol{x}, \mathbf{t}) = \exp\left(-\beta\|\boldsymbol{x} - \mathbf{t}\|^2\right)$ and the function $f(\mathbf{t}) = \sum_{i=1}^{m} \alpha_i K_\beta(\boldsymbol{x}_i, \mathbf{t})$. The solution to solve the $\boldsymbol{\alpha}$ could be written as:

$$\boldsymbol{\alpha} = \boldsymbol{K}_{\boldsymbol{\beta}}(\mathrm{x}, \mathrm{x})^{-1}\mathbf{y} \tag{3.5}$$

The kernel matrix could be expressed as below:

$$\boldsymbol{K}_{\boldsymbol{\beta}}(\mathrm{x}, \mathrm{x}) = \begin{pmatrix} 1 & K_\beta(\boldsymbol{x}_1, \boldsymbol{x}_2) & \cdots & K_\beta(\boldsymbol{x}_1, \boldsymbol{x}_m) \\ K_\beta(\boldsymbol{x}_2, \boldsymbol{x}_1) & 1 & \cdots & K_\beta(\boldsymbol{x}_2, \boldsymbol{x}_m) \\ K_\beta(\boldsymbol{x}_3, \boldsymbol{x}_1) & \ldots & 1 & K_\beta(\boldsymbol{x}_3, \boldsymbol{x}_m) \\ \vdots & \ldots & \ldots & \vdots \\ K_\beta(\boldsymbol{x}_m, \boldsymbol{x}_1) & K_\beta(\boldsymbol{x}_m, \boldsymbol{x}_2) & \ldots & 1 \end{pmatrix} \tag{3.6}$$

The inverse of a symmetric matrix is still a symmetric matrix. Since inverse form of kernel matrix is hard to express explicitly, we would use k_{ij} to represent each entry of matrix. Then, the inverse of the kernel matrix could be written as:

$$\boldsymbol{K}_\beta(\mathbf{x}, \mathbf{x})^{-1} = \begin{bmatrix} k_{11} & k_{12} & \cdots & k_{1m} \\ k_{21} & k_{22} & \cdots & k_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ k_{m1} & k_{m2} & \cdots & k_{mm} \end{bmatrix} \tag{3.7}$$

Therefore, substitute (3.7) in (3.5), the α could be expressed as:

$$\boldsymbol{\alpha} = \boldsymbol{K}_{\boldsymbol{\beta}}(\mathrm{x}, \mathrm{x})^{-1}\mathbf{y} = \begin{bmatrix} y_1k_{11} + y_2k_{12} + \cdots + y_mk_{1m} \\ y_1k_{21} + y_2k_{22} + \cdots + y_mk_{2m} \\ \vdots \\ y_1k_{m1} + y_2k_{m2} + \cdots + y_mk_{mm} \end{bmatrix} \tag{3.8}$$

Then, we could write the function $f(\mathbf{t})$ as:

$$\begin{aligned} f(\mathbf{t}) = \sum_{i=1}^{m} \alpha_i K_\beta(\mathbf{x}_i, \mathbf{t}) &= \begin{bmatrix} y_1k_{11} + y_2k_{12} + \cdots + y_mk_{1m} \\ y_1k_{21} + y_2k_{22} + \cdots + y_mk_{2m} \\ \vdots \\ y_1k_{m1} + y_2k_{m2} + \cdots + y_mk_{mm} \end{bmatrix} \cdot \begin{bmatrix} K_\beta(\mathbf{x}_1, \mathbf{t}) \\ K_\beta(\mathbf{x}_2, \mathbf{t}) \\ \vdots \\ K_\beta(\mathbf{x}_m, \mathbf{t}) \end{bmatrix} \\ &= \begin{bmatrix} k_{11}K_\beta(\mathbf{x}_1, \mathbf{t}) + k_{21}K_\beta(\mathbf{x}_2, \mathbf{t}) + \cdots + k_{m1}K_\beta(\mathbf{x}_m, \mathbf{t}) \\ k_{12}K_\beta(\mathbf{x}_1, \mathbf{t}) + k_{22}K_\beta(\mathbf{x}_2, \mathbf{t}) + \cdots + k_{m2}K_\beta(\mathbf{x}_m, \mathbf{t}) \\ \vdots \\ k_{1m}K_\beta(\mathbf{x}_1, \mathbf{t}) + k_{2m}K_\beta(\mathbf{x}_2, \mathbf{t}) + \cdots + k_{mm}K_\beta(\mathbf{x}_m, \mathbf{t}) \end{bmatrix} \cdot \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \end{aligned} \tag{3.9}$$

Then, for Gaussian kernel $K_\beta(\boldsymbol{x}, \mathbf{t}) = \exp\left(-\beta\|\boldsymbol{x} - \mathbf{t}\|^2\right)$, when the β tends to a positive large enough number, the kernel matrix (3.6) would tends to a diagonal matrix (diagonal is 1 otherwise is 0 in this case). Then the inverse of kernel matrix would tend to the same as its kernel matrix.

Therefore, the formula (3.9) would tend to:

$$
\begin{aligned}
f(\mathbf{t}) &\to \begin{bmatrix} K_\beta(\mathbf{x}_1,\mathbf{t}) + 0 + \cdots + 0 \\ 0 + K_\beta(\mathbf{x}_2,\mathbf{t}) + \cdots + 0 \\ \vdots \\ 0 + 0 + \cdots + K_\beta(\mathbf{x}_m,\mathbf{t}) \end{bmatrix} \cdot \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \\
&= \begin{bmatrix} K_\beta(\mathbf{x}_1,\mathbf{t}) \\ K_\beta(\mathbf{x}_2,\mathbf{t}) \\ \vdots \\ K_\beta(\mathbf{x}_m,\mathbf{t}) \end{bmatrix} \cdot \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \\
&= \sum_{i=1}^{m} K_\beta(\mathbf{x}_i,\mathbf{t}) y_i
\end{aligned}
\tag{3.10}
$$

According to the question, the classifier would be sign($f(\mathbf{t})$). Therefore, the final prediction is sign($\sum_{i=1}^{m} K_\beta(\mathbf{x}_i,\mathbf{t})y_i$), which means the weight($K_\beta(\mathbf{x}_i,\mathbf{t})$) of each label would influence the result. Given the nearest neighbour point of test data is $\boldsymbol{x}_{nn}$, we want the weight for nearest neighbour point to be large enough to simulate a 1-Nearest Neighbor classifier. Therefore, the weight for the nearest neighbour point should be $K_\beta(\boldsymbol{x}_{nn},\boldsymbol{t}) = \max\left\{K_\beta(\boldsymbol{x}_1,\boldsymbol{t}),\cdots,K_\beta(\boldsymbol{x}_m,\boldsymbol{t})\right\}$. This condition could be also written as:

$$
\exp\left(-\beta\|\boldsymbol{x}_{nn} - \mathbf{t}\|^2\right) > \exp\left(-\beta\|\boldsymbol{x}_i - \mathbf{t}\|^2\right) \text{ for } i \neq nn \tag{3.11}
$$

To satisfy (3.11), $\beta > 0$ must be held since $\|\boldsymbol{x}_{nn} - \boldsymbol{t}\|^2 < \|\boldsymbol{x}_i - \boldsymbol{t}\|^2$.
For the worst case, which means all the other points have a different label from $\boldsymbol{x}_{nn}$. Then the weight for $\boldsymbol{x}_{nn}$ should be larger than the sum of the other points, which can be expressed as:

$$
K_\beta(\boldsymbol{x}_{nn},\mathbf{t}) > \sum_{i \neq nn} K_\beta(\boldsymbol{x}_i,\mathbf{t}) \tag{3.12}
$$

In conclusion, a lower bound of β (positive large enough) could be found that achieve the trend in (3.10) and constrain described in (3.12) that enable the trained linear classifier to simulate a 1-Nearest Neighbor classifier trained on the same dataset.

Exercise 11:
To represent the game in a mathematical way, we write the board state at time step t as a matrix B_t, with 0 representing a mole underground and 1 representing a mole outside the hole at location $B_{i,j}$, e.g.:

$$
\boldsymbol{B}_1 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix} \tag{3.13}
$$

We can also represent a change to a hole by adding one to that hole position. We can then write the changes cause by an action $A_{i,j}$ as a matrix too, e.g. smacking hole 2,2:

$$
\boldsymbol{A}_{2,2} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.14}
$$

Note that any number of changes that is a multiple of 2 will not change the state at all. Thus changing X times to a hole is equivalent to changing X *mod* 2 times to a hole. Thus, by denoting the number of actions taken in location i,j ($A_{i,j}$) as $X_{i,j}$:

$$\boldsymbol{B}_{start} + \sum_{i,j}^{n^2} A_{i,j} X_{i,j} \mod 2 = \boldsymbol{0} \tag{3.15}$$

We can further flatten the matrices to vectors, e.g.:

$$\boldsymbol{B}_1 = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix} \tag{3.16}$$

$$\boldsymbol{A}_{2,2} = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.17}$$

Thus equation 3.15 becomes:

$$\begin{bmatrix} A_{1,1} \\ A_{1,2} \\ \cdots \\ A_{n,n-1} \\ A_{n,n} \end{bmatrix} \cdot \begin{bmatrix} X_{1,1} \\ X_{1,2} \\ \cdots \\ X_{n,n-1} \\ X_{n,n} \end{bmatrix} \mod 2 = -\boldsymbol{B}_{start}^T \mod 2 = \boldsymbol{B}_{start}^T \tag{3.18}$$

So, the solving the game means solving for the system of linear equations above. We can therefore use Gaussian elimination to find out the corresponding $X_{i,j}$. Note that since we are working in modulo 2, each action also only need to be played at most once, thus after the Gaussian elimination, the set of locations (i,j)s where $X_{i,j} \mod 2 = 1$ are where we want to smack.

Intuition: Performing Gaussian elimination with the actions can be thought as looking for possible combination of actions with all the other for each action, in the same time while taking the action on the board. Until we constructed a combination of actions that looks the same as the board state, the Gaussian elimination is done, since two same vectors are eliminated.

Complexity: The flattened vector of $A_{i,j}$ is of length n^2. Thus the matrix composed by all the actions in equation 3.18 is size $n^2 \times n^2$. When performing Gaussian elimination, for each $A_{i,j}$ we need to loop through each element (n^2) of the matrix below it . Thus making $\frac{(n^2+1)(n^2)}{2} \times n^2$ calculations. So the complexity is $\boldsymbol{O}(n^6)$ which is a polynomial of n.