Supervised Learning (COMP0078) Coursework 1

Student No.: 18010577 18092642

November 15, 2021

1 PART I

1.1 Linear Regression

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

   (a) Produce a plot similar to Figure 1, superimposing the four different curves corresponding to each fit over the four data points.

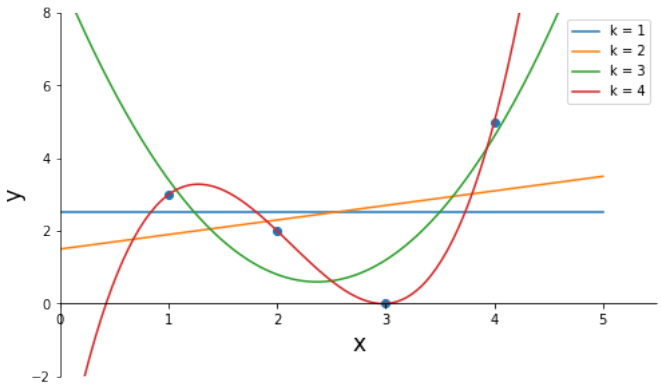

Figure 1: Fitted curves for dimension $k = 1, 2, 3, 4$

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

   - When $k = 1$, $y = 2.5$,
   - When $k = 2$, $y = 1.5 + 0.4x$
   - When $k = 3$, $y = 9 - 7.1x + 1.5x^2$
   - When $k = 4$, $y = -5 + 15.17x - 8.5x^2 + 1.33x^3$

   (c) For each fitted curve $k = 1, 2, 3, 4$ give the mean square error where $MSE = \frac{SSE}{m}$.

   - When $k = 1$, $MSE = 3.25$
   - When $k = 2$, $MSE = 3.05$
   - When $k = 3$, $MSE = 0.80$

- When $k = 4$, $MSE = 3.60746092 \times 10^{-27} \approx 0$

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

   (a) (i) Plot the function $sin^2(2\pi x)$ in the range $0 \leq x \leq 1$ with the points of the above data set superimposed.

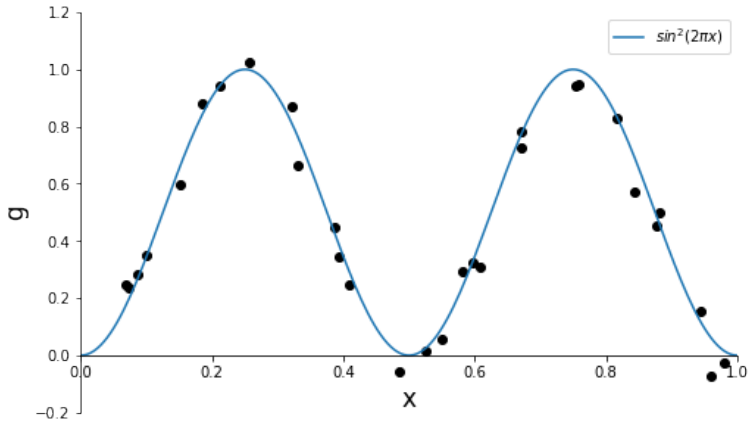

Figure 2: Curves for function $g = sin^2(2\pi x)$

   (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.

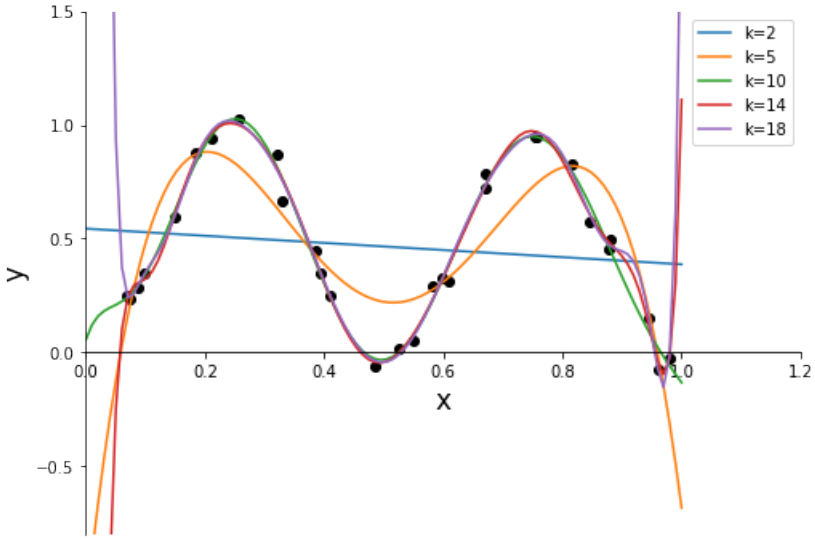

Figure 3: Fitted curves for dimension $k = 2, 5, 10, 14, 18$

   (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. Plot the natural $log(ln)$ of the training error versus the polynomial dimension $k = 1, ..., 18$ (this should be a decreasing function).

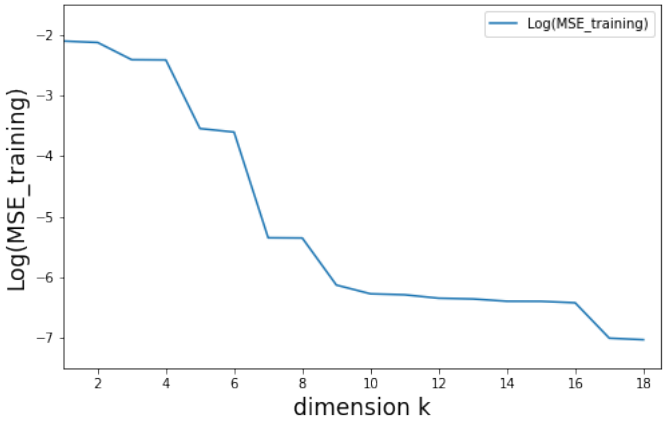

Figure 4: The natural $log(ln)$ of the training error versus the polynomial dimension $k = 1, ..., 18$

(c) Generate a test set T of a thousand points,

$$T_{0.07,1000} = \{(x_1, g_{0.07}(x1)), ..., (x_{1000}, g_{0.07}(x_{1000}))\}$$

Define the test error $tse_k(S, T)$ to be the MSE of the test set T on the polynomial of dimension k fitted from training set S. Plot the ln of the test error versus the polynomial dimension $k = 1, ..., 18$. Unlike the training error this is not a decreasing function. This is the phenomena of overfitting. Although the training error decreases with growing k the test error eventually increases since rather than fitting the function, in a loose sense, we begin to fit to the noise.

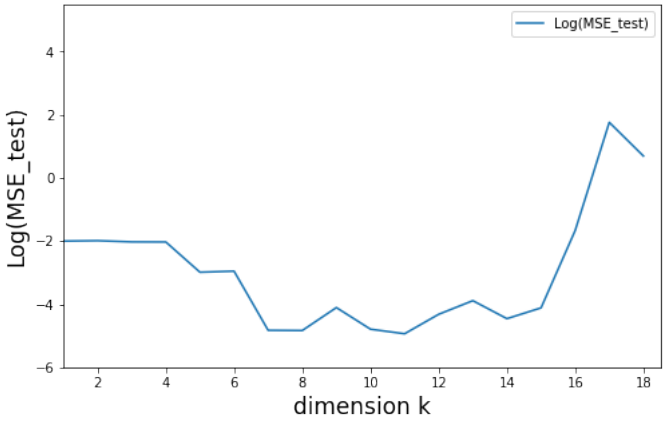

Figure 5: The natural $log(ln)$ of the test error versus the polynomial dimension $k = 1, ..., 18$

(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" plot the average results of a 100 runs (note: plot the $ln(avg)$ rather than the $avg(ln)$).

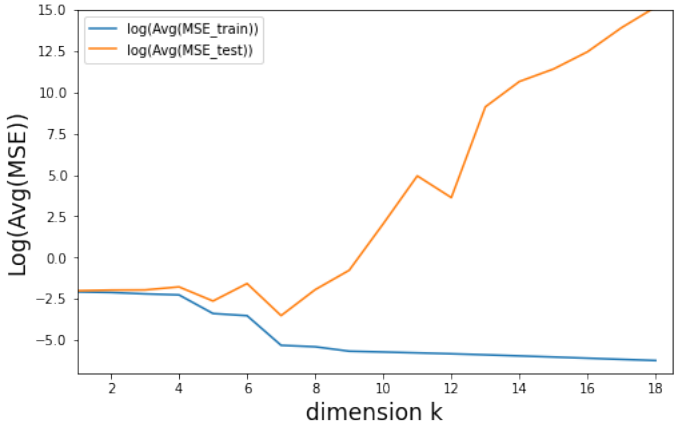

Figure 6: The natural $log(ln)$ of the average training and test error over 100 runs versus the polynomial dimension $k = 1, ..., 18$

3. Now use basis (for $k = 1, ..., 18$)

$$sin(1\pi x), sin(2\pi x), sin(3\pi x), ..., sin(k\pi x)$$

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

- Figure 7 shows the natural $log(ln)$ of the training error versus the new basis with dimension $k = 1, ..., 18$.

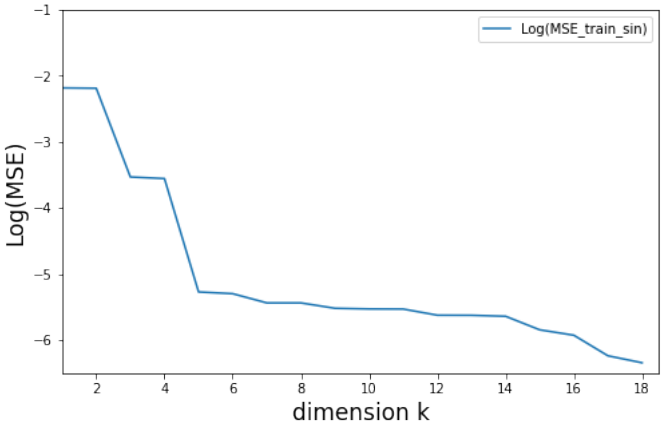

Figure 7: Log of training error with the new basis for dimension $k = 1, ..., 18$

- Figure 8 shows the natural $log(ln)$ of the test error versus the new basis with dimension $k = 1, ..., 18$.

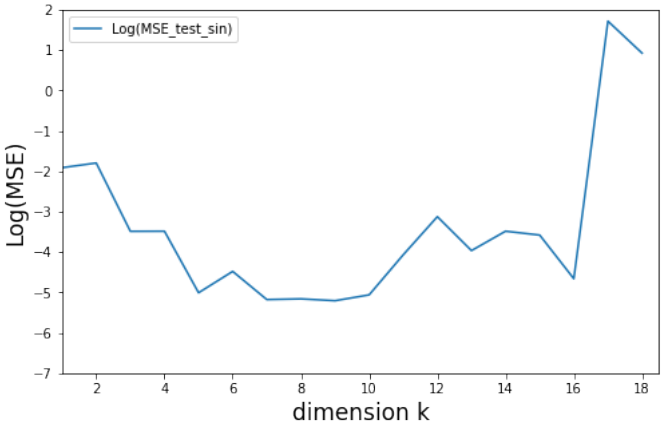

Figure 8: Log of test error with the new basis for dimension $k = 1, ..., 18$

- Figure 9 shows the natural $log(ln)$ of the average training and test error over 100 runs versus the new basis with dimension $k = 1, ..., 18$

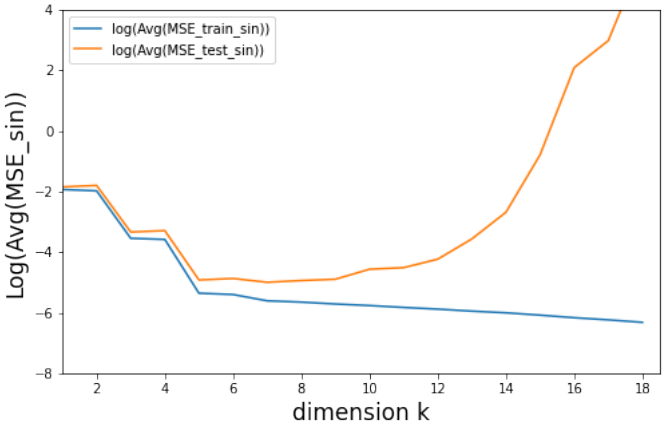

Figure 9: The natural $log(ln)$ of the average training and test error over 100 runs versus the new basis with dimension $k = 1, ..., 18$

1.2 Filtered Boston housing and kernels

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

   a. Naive Regression.

      - Average MSE on the training set is 83.31708666977785.
      - Average MSE on the test set is 87.0455422874325.

   b. Give a simple interpretation of the constant function in 'a.' above.

      - The constant function is equivalent to estimate the mean of y (MEDV column 13) across the training data set.

   c. Linear Regression with single attributes.

Attribute	Train MSE	Test MSE
1	73.9161	68.6356
2	74.4746	71.7526
3	65.1872	64.0389
4	80.7087	84.9232
5	68.6358	70.1635
6	43.8322	43.6567
7	73.3277	71.0563
8	78.8207	80.3705
9	72.7356	71.5001
10	65.8169	66.5866
11	62.6187	63.1907
12	38.1183	39.5822

Table 1: Train MSE and Test MSE for each attribute

   d. Linear Regression using all attributes. Now we would like to perform linear regression using all of the data attributes at once.
   Perform linear regression on the training set using this regressor, and incorporate a bias term as above. Calculate the MSE on the training and test sets and note down the results. You should find that this method outperforms any of the individual regressors.

      - MSE of training set over 20 runs is 21.725443763165256.
      - MSE of test set over 20 runs is 25.233142131166474.

1.3 Kernelised ridge regression

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

$$K(x_i, x_j) = exp(-\frac{\|x_i, x_j\|^2}{2\sigma^2})$$

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

a. Perform kernel ridge regression on the training set using five-fold cross-validation to choose among all pairing of the values of γ and σ. Choose the γ and σ values that perform the best to compute the predictor (by then retraining with those parameters on the training set) that you will use to report the test and training error.

- The best $\gamma = 2^{-35}$
- The best $\sigma = 2^{8.5}$

b. Plot the "cross-validation error" (mean over folds of validation error) as a function of γ and σ.

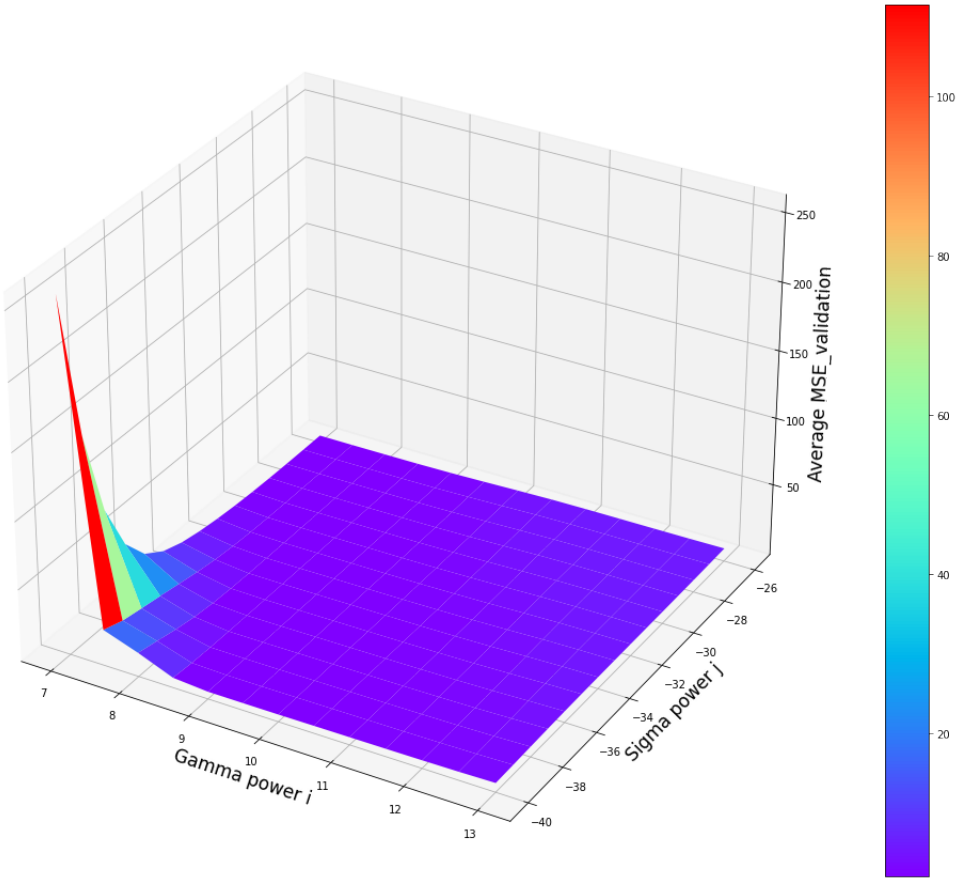

Figure 10: The "cross-validation error" for each pair of γ power i and σ power j

c. Calculate the MSE on the training and test sets for the best γ and σ.

- Train MSE = 3.005378123001379
- Test MSE = 5.992972943499792

d. Repeat "exercise 4a,c,d" and "exercise 5a,c" over 20 random (2/3, 1/3) splits of your data. Record the train/test error and the standard deviations (σ') of the train/test errors and summarise these results in the following type of table.

Method	Train MSE	Test MSE
Naive Regression	84.1816 ± 5.5870	85.0901 ± 11.2246
Linear Regression (attribute 1)	71.1390 ± 4.4375	73.4314 ± 8.7770
Linear Regression (attribute 2)	74.6300 ± 4.8105	71.4079 ± 9.6264
Linear Regression (attribute 3)	64.5203 ± 5.1925	65.5613 ± 10.6685
Linear Regression (attribute 4)	81.6848 ± 4.0633	82.8574 ± 8.2730
Linear Regression (attribute 5)	69.5758 ± 2.7240	68.2147 ± 5.4450
Linear Regression (attribute 6)	43.7696 ± 3.5380	43.7153 ± 7.2320
Linear Regression (attribute 7)	73.6223 ± 4.2488	70.3142 ± 8.4383
Linear Regression (attribute 8)	77.1745 ± 5.3012	83.5155 ± 10.5720
Linear Regression (attribute 9)	73.0261 ± 3.9211	70.6443 ± 7.7452
Linear Regression (attribute 10)	64.8753 ± 3.7726	68.2690 ± 7.4684
Linear Regression (attribute 11)	63.7605 ± 4.6257	60.8157 ± 9.3264
Linear Regression (attribute 12)	38.0112 ± 3.1347	39.7634 ± 6.2118
Linear Regression (all attributes)	21.8140 ± 1.9940	24.9957 ±4.2884
Kernel Ridge Regression	7.6074 ± 2.4089	15.1697 ± 4.8036

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

2 PART II

2.1 k-Nearest Neighbors

2.1.1 Generating the data

6. Visualising the randomly generated h function and its corresponding decision regions:

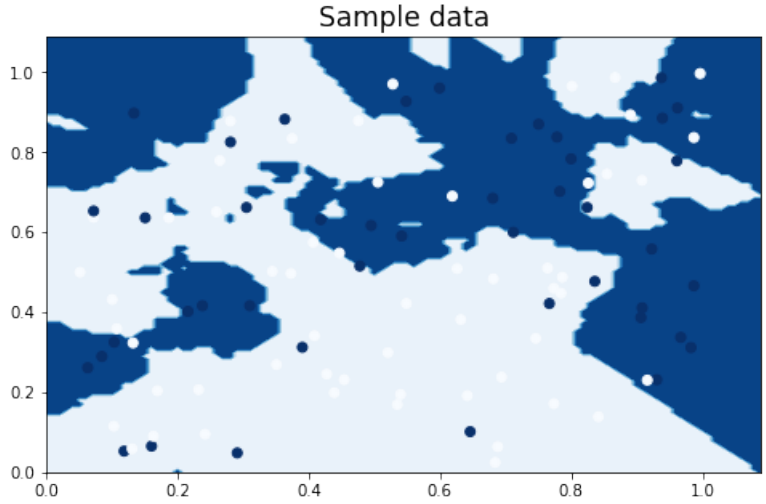

Figure 11: Randomly generated hypothesis h, with $|S| = 100$ and $v = 3$

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

In this section we visualise the performance as a function of k. You will produce a figure where the vertical axis is the estimated generalisation error and the horizontal axis is $k = 1, ..., 49$.

7. (a) Demonstrating the effect of m/k on K-NN performance using Protocol A:

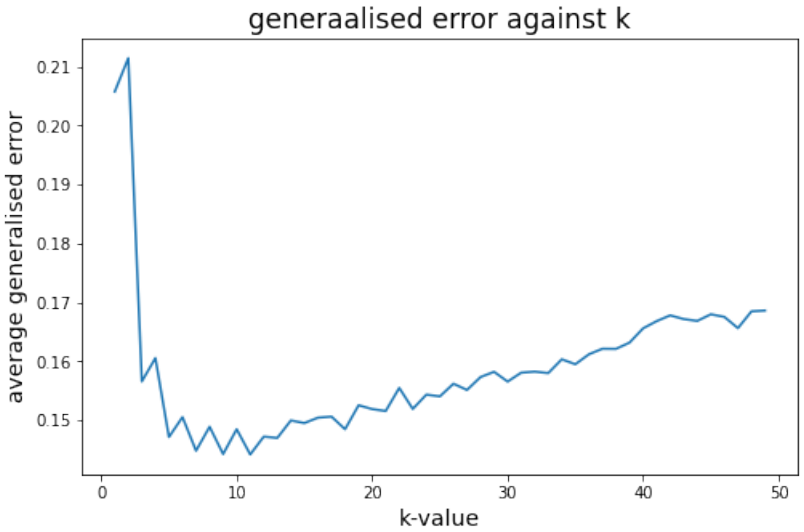

Figure 12: Generalisation error against the number of neighbors k

(b) First we notice that the generalisaton error decreases extremely fast initially, because when taking inadequate amount of neighboring training points, the model would overfit and suffer from high variance thus poorly generalised on unseen data. As k increases, the generalisation error decreases and reaches optimum at around $k = 10$ where we have reached the optimal ratio between training size m and k neighbors. After which the model start to suffer from underfitting, which are therefore biased and cause the error rate to increase.
Another interesting thing is that when k takes even number, its error rate is higher than its adjacent odd-valued k. The reason being when k is even, there is a probability that the result is undefined and y-value is randomly selected, which causes error rate to rise.

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

In this section you will estimate the optimal k as a function of the number of training points. Perform the following experimental protocol.

8. (a) demonstrating the effect of m/k on K-NN performance using protocol B:

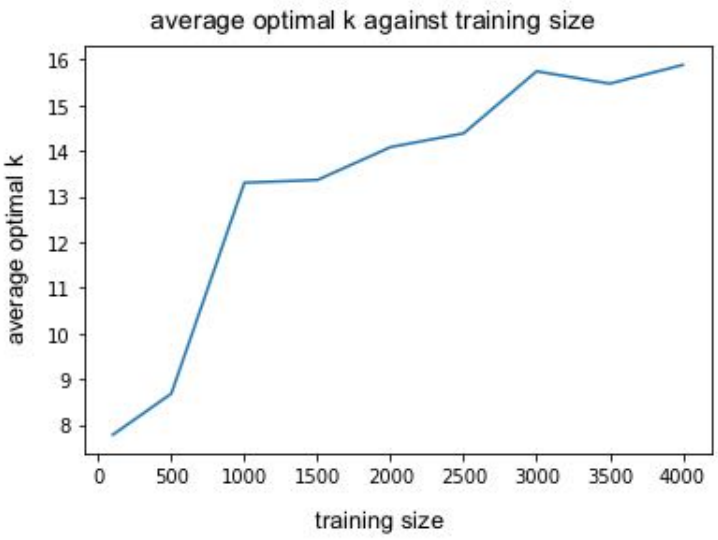

Figure 13: Optimal k-value against the number of neighbors k

(b) The graph resembles a curve which initially increase very rapidly, and then the gradient decreases and starting to flatten out. Although the smallest training set is 100, we can imaging if the training set is 10, the optimal k should be 1 or 2, the shape of the curve will be more obvious.
Initially as training set grows, the need for larger k increases, therefore the steep increase. But when we have reached a sufficiently large k, the training points in this neighbor are enough to represent the local distribution around the point of interest, and a larger k becomes redundant, therefore the curve starts to become flat.

3 PART III

3.1 Questions

9. Kernel modification Consider the function $K_c(\boldsymbol{x}, \boldsymbol{z}) := c + \sum_{i=1}^{n} x_i z_i$ where $\boldsymbol{x}, \boldsymbol{z} \in \Re^n$.

   (a) Claim: the kernel function $K_c(\boldsymbol{x}, \boldsymbol{z})$ is a positive semi-definite kernel if c $\geq$ 0.

   Proof of the claim:
   By the definition of positive semi-definite kernel, we need to prove that the matrix K_c:

   $$\boldsymbol{K}_{\boldsymbol{c}ij} = K_c(\boldsymbol{x}_i, \boldsymbol{x}_j) \quad \text{for } 1 \leq i, j \leq n, \text{ and } \; x_i, x_j \in \mathbb{R}^n$$

   is a positive semi-definite matrix.

   By definition, a matrix M is positive semi-definite iff for any given vector $\mathbf{a} \in \mathbb{R}^n$ we have that $\mathbf{a}^T M \mathbf{a} \geq 0$. Now to prove K_c is P.S.D, we need to show given any vector $\mathbf{a}$, $\mathbf{a}^T K_c \mathbf{a} \geq 0$:

   $$\begin{aligned}
   \mathbf{a}^T K_c \mathbf{a} &= \sum_i^n \sum_j^n a_i a_j K_c(\boldsymbol{x}_i, \boldsymbol{x}_j) \\
   &= \sum_i \sum_j a_i a_j (c + \boldsymbol{x}_i^T \boldsymbol{x}_j) \\
   &= \sum_i \sum_j a_i a_j c + \sum_i \sum_j a_i a_j \boldsymbol{x}_i^T \boldsymbol{x}_j \\
   &= \|\mathbf{a}\|^2 c + \sum_i \sum_j a_i \boldsymbol{x}_i^T \boldsymbol{x}_j a_j \\
   &= \|\mathbf{a}\|^2 c + \| \sum_i a_i \boldsymbol{x}_i \|^2
   \end{aligned}$$

   Since both $\|\mathbf{a}\|^2$ and $\| \sum_i a_i \boldsymbol{x}_i \|^2$ are greater or equal to 0, to guarantee $\mathbf{a}^T K_c \mathbf{a} \geq 0$ we need to take $c \geq 0$. Then, both of the term $c\|\mathbf{a}\|^2$ and $| \sum_i a_i \boldsymbol{x}_i \|^2$ must be greater or equal to 0.

   Thus for $c \geq 0$, the kernel $K_c(\boldsymbol{x}, \boldsymbol{z})$ is a positive semi-definite kernel. □

   (Note that the kernel function $K_c(\boldsymbol{x}, \boldsymbol{z}) := c + \sum_{i=1}^{n} x_i z_i$ is equivalent to $c + \boldsymbol{x}^T \boldsymbol{z}$ which is in the form of a polynomial kernel function $K(\boldsymbol{x}, \mathbf{t}) = (a + \boldsymbol{x}^T \mathbf{t})^r$ if c $\geq$ 0. And since all polynomial kernels are positive semi-definite kernels, K_c is a P.S.D kernel as well, if the value of c is set to be greater or equal to 0.)

   (b) For this particular polynomial kernel of order 1, the term $\sum_{i=1}^{n} x_i z_i$ is the dot product between the 2 vector. By definition: $\boldsymbol{a} \cdot \boldsymbol{b} = |\boldsymbol{a}||\boldsymbol{b}| cos(\theta)$ where θ is the angle between $\boldsymbol{a}, \boldsymbol{b}$.

Therefore vectors $\mathbf{x}$ that are in similar direction as $\mathbf{z}$ will have higher kernel values, which implies that they are given higher weights in the regression and are more influential to the prediction of interest. conversely a data point completely irrelevant with z will have no impact on prediction since their dot product is 0.

By adding a larger constant we essentially have reduce the influence of similarity and $|\boldsymbol{x}|$ on the weight, and data points will have a more even contribution to the final prediction rather than heavily dependent on those which are more relevant to $\boldsymbol{z}$. In the end we can obtain a smoother prediction.

10. We are performing a linear regression with Gaussian kernel $K_\beta(\mathbf{x}, \mathbf{t}) = exp(-\beta\|\mathbf{x} - \mathbf{t}\|^2)$. Then by representer theorem, the function $f : \mathbb{R}^n \to \mathbb{R}$ can be written in the form $f(\mathbf{t}) = \sum_{i=1}^m \alpha_i K_\beta(\mathbf{x}_i, \mathbf{t})$. And the vector $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_m)^T$ can be given by:

$$\boldsymbol{\alpha} = \boldsymbol{K_\beta}(\mathbf{x}, \mathbf{x})^{-1}\mathbf{y} \tag{1}$$

where $\boldsymbol{K}_\beta(\mathbf{x}, \mathbf{x})$ is the Kernel matrix of the Gaussian kernel:

$$\boldsymbol{K}_\beta(\mathbf{x}, \mathbf{x}) = \begin{bmatrix} 1 & K_\beta(\mathbf{x}_1, \mathbf{x}_2) & \cdots & K_\beta(\mathbf{x}_1, \mathbf{x}_m) \\ K_\beta(\mathbf{x}_2, \mathbf{x}_1) & 1 & \cdots & K_\beta(\mathbf{x}_2, \mathbf{x}_m) \\ \vdots & \vdots & \ddots & \vdots \\ K_\beta(\mathbf{x}_m, \mathbf{x}_1) & K_\beta(\mathbf{x}_m, \mathbf{x}_2) & \cdots & 1 \end{bmatrix}$$

Since this is a symmetric, its inverse is of similar form:

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

where c is some constant to keep the diagonal entries equal to 1, and k_{ij} are constant to be calculated.

Now Consider the Gaussian kernel function $K_\beta(\mathbf{x}, \mathbf{t}) = exp(-\beta\|\mathbf{x} - \mathbf{t}\|^2)$, since we know that $\|\mathbf{x} - \mathbf{t}\|^2 \geq 0$, if we set the value of β to be a large positive number, the value of kernel will tend to 0. Then both $K_\beta(\mathbf{x}_i, \mathbf{x}_j)$ and k_{ij} will tend to 0. But for now we can express $f(\mathbf{t})$ as:

$$f(\mathbf{t}) = f(\mathbf{t}) = \sum_{i=1}^m \alpha_i K_\beta(\mathbf{x}_i, \mathbf{t}) = c \cdot \begin{bmatrix} y_1 + y_2 k_{12} + \cdots + y_m k_{1m} \\ y_1 k_{21} + y_2 + \cdots + y_m k_{2m} \\ \vdots \\ y_1 k_{m1} + y_2 k_{m2} + \cdots + y_m \end{bmatrix} \cdot \begin{bmatrix} K_\beta(\mathbf{x}_1, \mathbf{t}) \\ K_\beta(\mathbf{x}_2, \mathbf{t}) \\ \vdots \\ K_\beta(\mathbf{x}_m, \mathbf{t}) \end{bmatrix}$$

And we can rearrange it into:

$$f(\mathbf{t}) = c \cdot \begin{bmatrix} K_\beta(\mathbf{x}_1, \mathbf{t}) + k_{21} K_\beta(\mathbf{x}_2, \mathbf{t}) + \cdots + k_{m1} K_\beta(\mathbf{x}_m, \mathbf{t}) \\ k_{12} K_\beta(\mathbf{x}_1, \mathbf{t}) + K_\beta(\mathbf{x}_2, \mathbf{t}) + \cdots + k_{m2} K_\beta(\mathbf{x}_m, \mathbf{t}) \\ \vdots \\ k_{1m} K_\beta(\mathbf{x}_1, \mathbf{t}) + k_{2m} K_\beta(\mathbf{x}_2, \mathbf{t}) + \cdots + K_\beta(\mathbf{x}_m, \mathbf{t}) \end{bmatrix} \cdot \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \tag{2}$$

Now denote the data point which is the closest to $\boldsymbol{t}$ as $\boldsymbol{x}_{nn}$, we must have:

$$\text{for } i \neq nn \qquad exp(-\beta\|\mathbf{x}_{nn} - \mathbf{t}\|^2) > exp(-\beta\|\mathbf{x}_i - \mathbf{t}\|^2) \qquad \text{if } \beta > 0$$

and the larger β is, the larger $K_\beta(\mathbf{x}_{nn}, \mathbf{t})$ is comparing to other $K_\beta(\mathbf{x}_i, \mathbf{t})$. Therefore we need β to be large and positive for y_{nn} to be more influential to our final prediction. Recall that when β is large, both kernel matrix and its inverse will resemble identity matrix and $k_{ji}K_\beta(\mathbf{x}_i, \mathbf{t})$ will converge to 0 much faster than $K_\beta(\mathbf{x}_i, \mathbf{t})$.

Thus equation (2) becomes:

$$f(\mathbf{t}) \to \sum_{i=1}^{m} y_i K_\beta(\mathbf{x}_i, \mathbf{t}) \tag{3}$$

Where now every y-value contributes to the prediction, and the amount of influence it has depends on the Gaussian Kernel value. In order to simulate 1-NEAREST NEIGHBOR CLASSIFIER using this Gaussian kernel regression, we need the kernel of x_{nn} to be large enough to dominate.

Consider the worst case where every other points are classified differently from x_{nn}: in order for $\mathbf{t}$ to be classified the same as x_{nn}, we need:

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

then a function $\beta = \hat{\beta}(\boldsymbol{x}_1, \cdots, \boldsymbol{x}_m, \boldsymbol{t})$ should exist to find a lower bound for β large enough to satisfy (3), and for any value greater than that, $f((t))$ will completely and only depend on y_{nn} and we have a 1-NN algorithm.

11. We first define a matrix X to represent the current state of the board, whose entries take value from $\{0, 1\}$:

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nn} \end{bmatrix} \tag{4}$$

The value of x_{ij} represent the state of the hole at a given time. When there is a mole appearing in that hole $x_{ij} = 1$, and x_{ij} takes 0 when mole in that hole is underground.

Now we define the action of whacking a hole whose coordinate is ij, call this action A_{ij}.

A_{ij} is also a binary valued matrix whose (i, j) and its adjacent entries take value 1 and everywhere else take 0. e.g. For a 4 by 4 whack a mole board:

$$A_{11} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{5}$$

Every time an action is executed, a matrix addition is performed by adding the action matrix A_{ij} to M. Since when a hole is whacked twice, the outcome is the same as the initial state i.e. $1 + 1 = 0$. Therefore we need to perform the addition in binary number system, the we can represent the game as a summation of action matrices and the initial state of the board. Now define c_{ij} to denote the number of times action A_{ij} is executed before the board is cleared. In the end we can represent the who process as an equation as follow:

$$\boldsymbol{X}_{start} + \sum_{ij} A_{ij} c_{ij} = \mathbf{0} \tag{6}$$

Note that since we are in a binary system, $c_{ij} \in \{0, 1\}$ as well. If we flatten the action matrices into vectors and construct a larger matrix using all these vectors, (3) can be represented as follow:

$$\begin{bmatrix} \cdots & vec(A_{11})^T \cdots \\ \cdots & vec(A_{12})^T \cdots \\ & \vdots \\ \cdots & vec(A_{nn})^T \cdots \end{bmatrix} \begin{bmatrix} c_{11} \\ c_{12} \\ \vdots \\ c_{nn} \end{bmatrix} = -vec(\mathbf{X}_{start}) = vec(\mathbf{X}_{start}) \tag{7}$$

For example for a 2 by 2 board the objective equation would be:

$$\begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} c_{11} \\ c_{12} \\ c_{21} \\ c_{22} \end{bmatrix} = vec(\mathbf{X}_{start}) \tag{8}$$

Therefore the game can be represented as a linear equations system. Then we can obtain

a solution to the game given a starting board configuration by performing row echelon reduction (note that the procedure also need to be performed in binary). Once we have reached a solution for the linear system, c_{ij} with solution 1 are the holes that we need to hit, and in what order do we whack the holes does not matter.

Since the linear system is constructed using vectorised action matrices, the dimension of the linear system is n^2 by n^2, thus the Gaussian elimination process has time complexity of $\boldsymbol{O}(n^6)$, which is polynomial in n. In addition, since the matrix of the linear transformation has non-zero determinant, it has full rank thus must be solvable with a unique solution, thus the approach is viable.