# Supervised Learning (COMP0078) - Coursework 1

Team: YES, Student numbers: 21146187 and 21117853

November 15, 2021

## Part 1

**Q1**

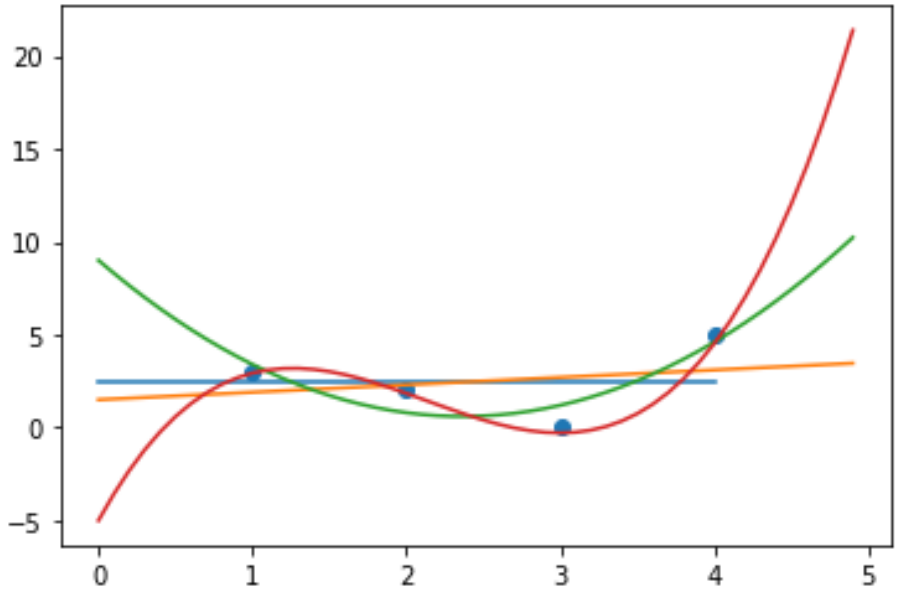
**(a)**

**k=1**

**k=2**

**k=3**

**k=4**




**(b)**

We fit the given data into polynomials $y = \beta_0$, $y = \beta_0 + \beta_1 x, y = \beta_0 + \beta_1 x + \beta_2 x^2$ respectively. Then, by print out its coefficients, we could get:

The equation corresponding to k = 1 is **2.5**

The equation corresponding to k = 2 is **1.5 + 0.4$x$**

The equation corresponding to k is **9 – 7.1$x$ + 1.5$x^2$**

**(c)**

By taking the mean of the squared difference between actual data and our predicted data, we find the MSE as the following:

k=1 >>**MSE=3.25**

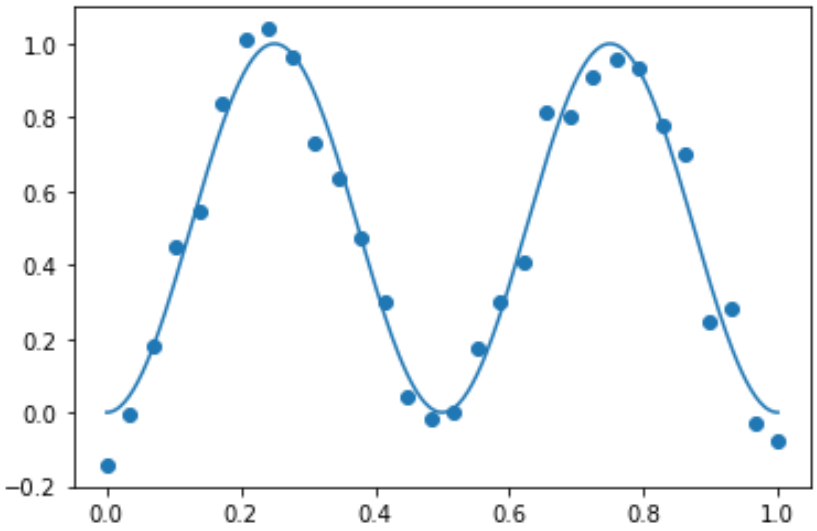
k=2 >>**MSE=3.05**

k=3 >>**MSE=0.8**

k=4 >>**MSE=5.605843e-29**

# Q2

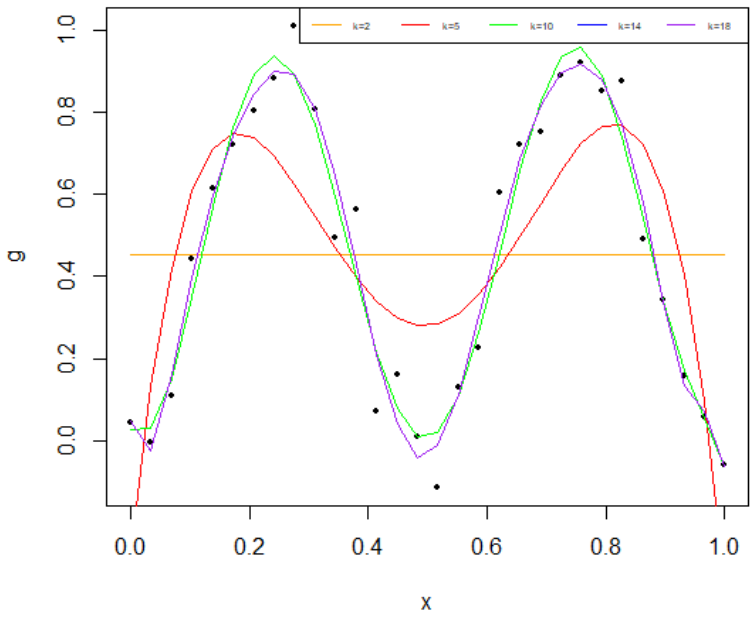
**2a-i**




**2a-ii:**

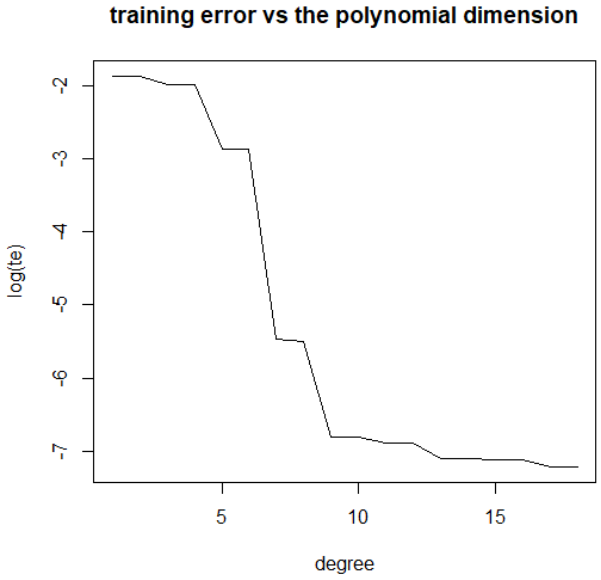
**(**the figure would look smoother if we have change our settings for x-axis)

For this question, we would have to generate the data we are going to use first. Then, do the same process as fitting polynomials in Q1. At this time, we are fitting our data for k= 2, 5, 10, 14, 18, with polynomial degrees equals to 1, 4, 9, 13, 17 respectively.

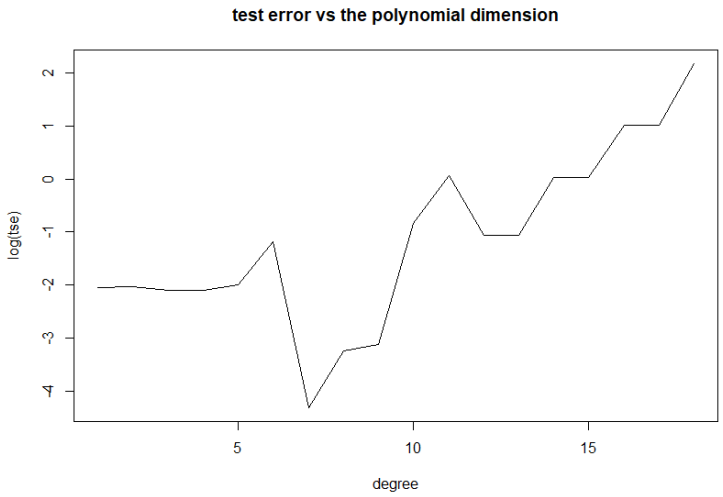


### 2b) log of the training error versus the polynomial dimension

We do similar things as in 2)-ii, fitting with more k values now(k takes value from 1 to 18). Then, combining MSE for each k and take log of these values. Finally, we could plot this figure below.



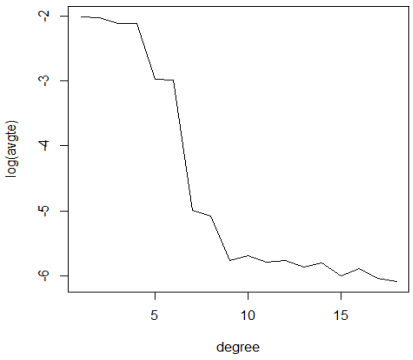

### 2c) log of the training error versus the polynomial dimension



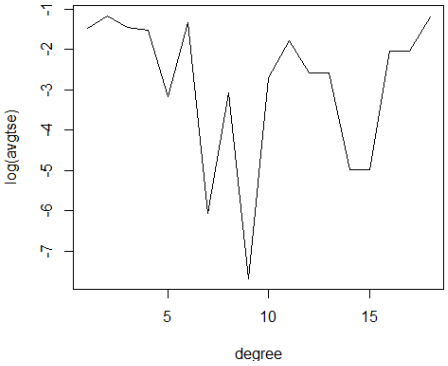
### 2d) log of the training/test error versus the polynomial dimension(repeated)

We repeat our models in 2b and 2c with each degree of k, then collect the MSE and taking means of each MSE value. After taking log of the MSE value, we could plot the figure below.

2d(b)



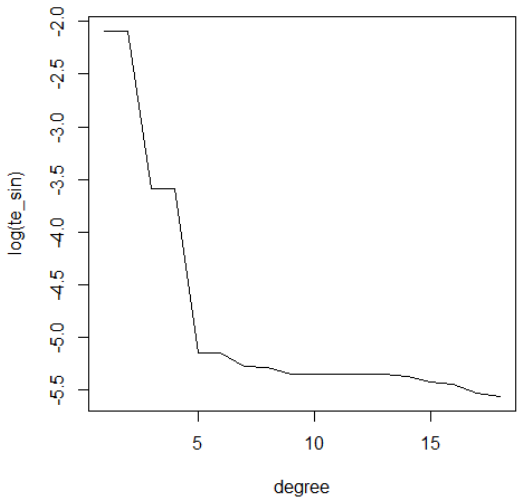

2d(c)



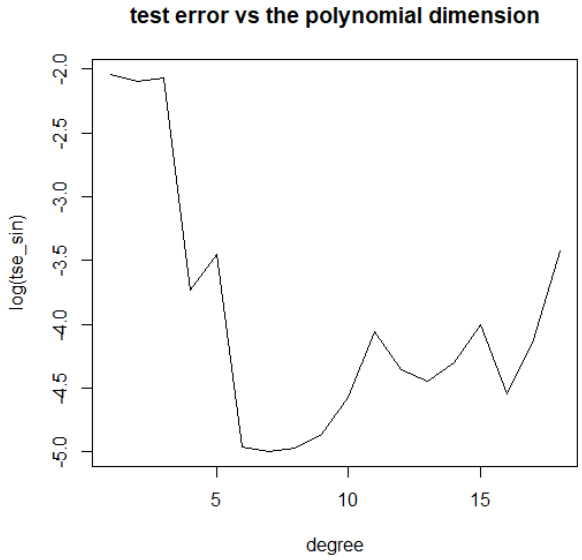
# Q3

Similar as what we did in question 2, but fitting in a different basis now.

**3(b) log of the training error versus the polynomial dimension(sine basis)**

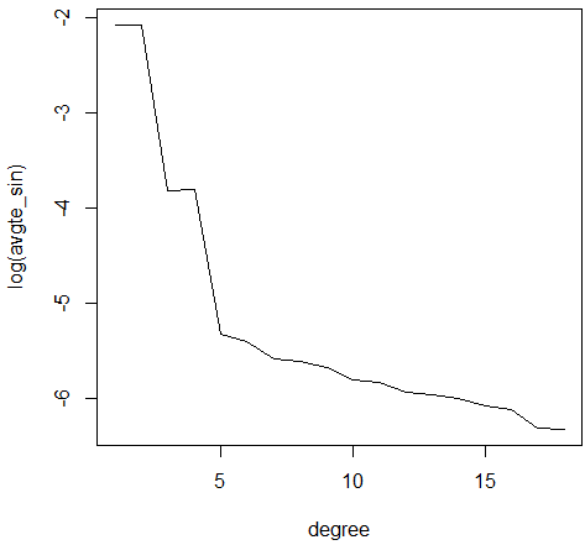



**3(c) log of the test error versus the polynomial dimension(sine basis)**



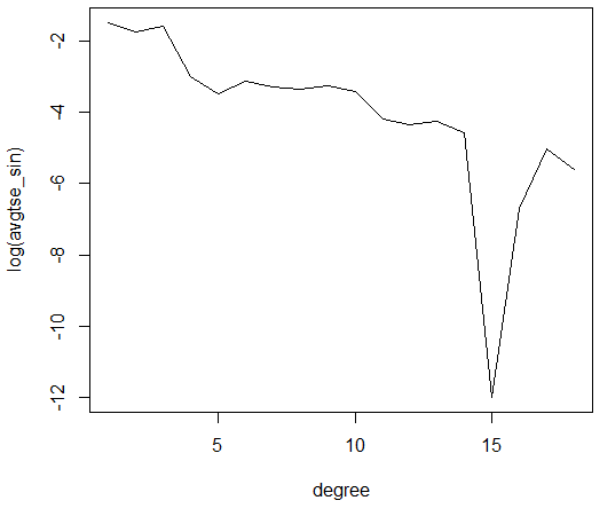
**3(d)-b log of the training error versus the polynomial dimension**

**(sine basis, repeated)**




**3(d)-c log of the training error versus the polynomial dimension**

**(sine basis, repeated)**



# Q4 Boston Housing

**4a Naïve Regression**

The MSE for the **training set** is **85.23079**
The MSE for the **test set** is **83.07443**

We perform 20 runs for fitting models based on different split for our train and test set(each splits like (2/3, 1/3)).
Sinse we have insert a '1', here we called the constant term,
the model we fitted all took form: $y = c$, *c for constant*

For each run, we take MSE as the mean of the (actual data - prediction)^2

At last, taking an average would give us the final answer of MSE for each set, which is 85.23079 for the training set and 83.07443 for the test set.

**4b. interpretation**

The constant function in 4a above would be the mean of the response(MEDV) of the train set in each random split. Though the constants could be different, all of them would equal to the mean of MEDV in the new training set(splitted). After we do the predictions on the test set, the value we would get is still going to be the mean of MEDV as we fit the constant value.

Each value:
22.19970 22.13472 22.52107 22.75045 22.28309 22.40297 22.12938 22.79614 22.98724 22.54184 22.85460 22.30979 22.33442 22.56973 22.50386 22.88042 22.30030 23.00089 23.04451 22.90030
The mean of these values is **22.57227**

**4c. Single attribute >> all takes form** $y = \beta_1 x + \varepsilon$

By looking at the summary for each model, we could learn their coefficients

1. CRIM

coefficient=24.28756 MSE train=72.11124 MSE test=71.36584

2. ZN

coefficient=8.470522 MSE train=73.47042 MSE test=73.91879

3. INDUS

coefficient=6.914193 MSE train=65.01750 MSE test=64.23843

4. CHAS

coefficient=9.53455 MSE train=82.36364 MSE test=81.47620

5. NOX

coefficient=3.066493 MSE train=69.29475 MSE test=68.84793

6. RM

coefficient=-1.872124 MSE train=42.26706 MSE test=46.98539

7. AGE

coefficient=6.244245 MSE train=72.74275 MSE test=72.19663

8. DIS

coefficient=5.590388 MSE train=79.74673 MSE test=78.43921

9. RAD

coefficient=5.478468 MSE train=72.34141 MSE test=72.09610

10. TAX

coefficient=6.793268 MSE train=65.80141 MSE test=66.45528

11. PTRATIO

coefficient=8.234858 MSE train=63.05833 MSE test=62.20557

12. LSTAT

coefficient=8.089312 MSE train=38.43864 MSE test=38.96187

**4d. all the attributes**

**> MSE**

Train: 21.51749

Test: 25.73386

# Q5 KRR

**5a) c)**

Training:

Total>>37.73618

MSE>> 4.225576

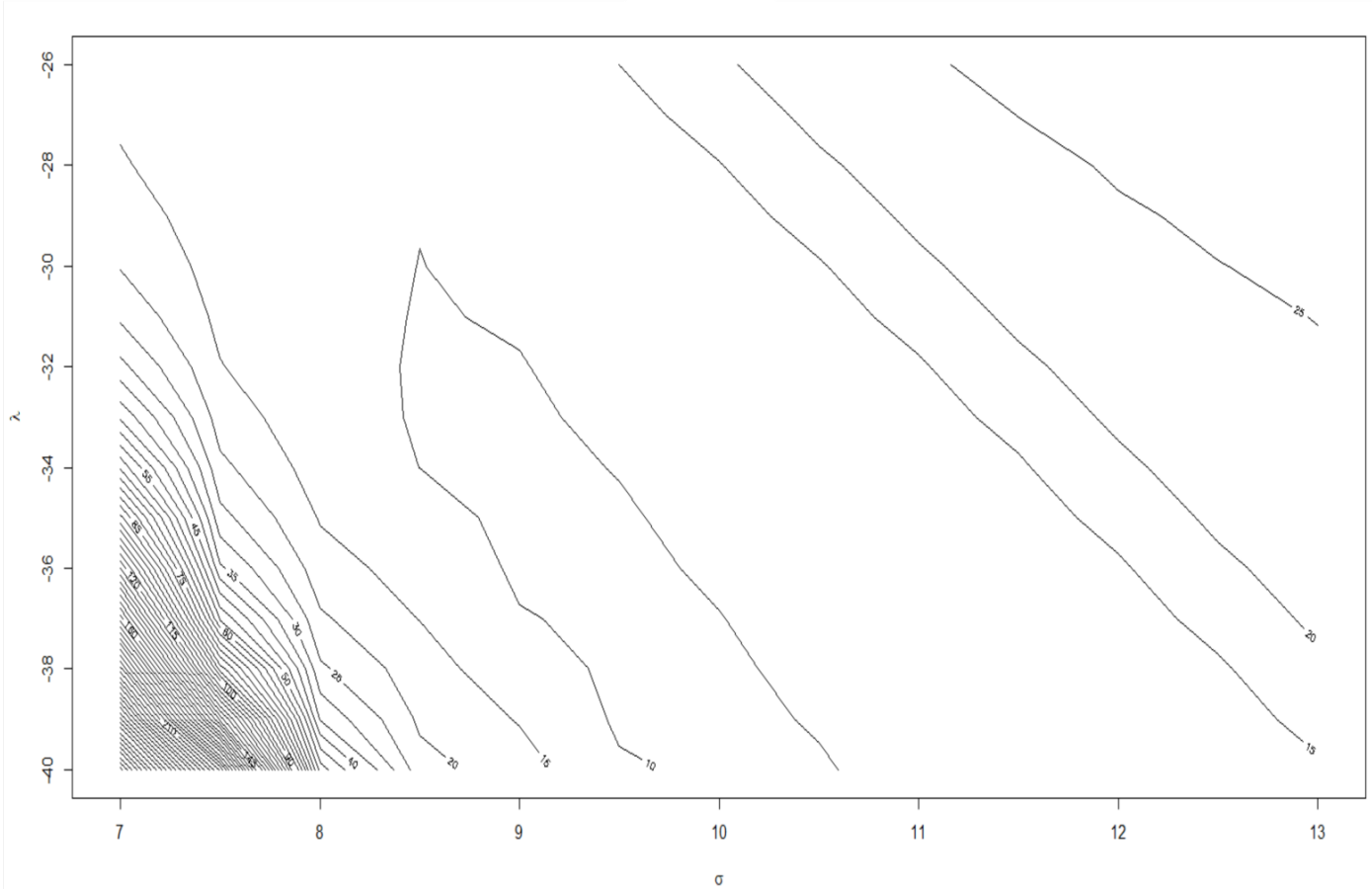
Testing:

Total>>49.0904

MSE>>14.25957

The MSE was calculated by taking mean of the error square. The total error was taken by the sum of the error in each cross validation process. As I have calculated the MSE first, I just calculated the total error by using the relation between MSE and the sum of errors.

**5b) cross-validation error**



**5d)**

| Method | MSE train | MSE test |
|---|---|---|
| Naïve Regression | 85.23079 ±5.225291 | 83.07443 ±10.37538 |
| Linear Regression 1 | 72.11124 ±4.821854 | 71.36584±9.527645 |
| Linear Regression 2 | 73.47042±5.601844 | 73.91879±11.190405 |
| Linear Regression 3 | 65.01750±5.404015 | 64.23843±10.736959 |
| Linear Regression 4 | 82.36364±5.218720 | 81.47620±10.692614 |
| Linear Regression 5 | 69.29475±5.279176 | 68.84793±10.515201 |
| Linear Regression 6 | 42.26706±4.442841 | 46.98539±9.280079 |
| Linear Regression 7 | 72.74275±5.593867 | 72.19663±11.155257 |
| Linear Regression 8 | 79.74673±5.838468 | 78.43921±11.791739 |
| Linear Regression 9 | 72.34141±5.531267 | 72.09610±11.234499 |
| Linear Regression 10 | 65.80141±5.292812 | 66.45528±10.665948 |
| Linear Regression 11 | 63.05833±4.227105 | 62.20557±8.597540 |
| Linear Regression 12 | 38.43864±2.486173 | 38.96187±5.007643 |
| Linear Regression (ALL) | 21.51749±2.476572 | 25.73386±5.965998 |
| Kernel Ridge Regression | 8.63611±1.484192 | 12.78618±2.621338 |

## Part 2: $k$-nearest neighbours

A voted-centre hypothesis function $h_{S,v}\ [0,1]^2 \to \{0,1,\square\}$ where $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_{|S|}, y_{|S|})\}$ is a set of labelled centres with each $\mathbf{x}_i \in [0,1]^2$ and $y_i \in \{0,1\}$ and $v$ is a positive integer. Where $h_{S,v} = 0$ if the majority of the $v$ nearest $\mathbf{x}_i$'s to $\mathbf{x}$ in $S$ are '0', similarly for $h_{S,v} = 1$ and finally in certain 'corner' cases 'the majority of $v$' will not be well defined, and in that case $h_{S,v}$ is generated by sampling uniformly at random from $\{0,1\}$.

### Question 6

$S$ was generated by sampling uniformly from $[0,1]^2$ to create the $\mathbf{x}$ data with 100 corresponding labels sampled uniformly at random from $\{0,1\}$. The hypothesis $h_{S,3}$ was visualized over the domain $[0,1]^2$ by carrying out the procedure $h_{S,3}(\mathbf{x})$ over $400 \times 400$ uniformly distributed points. The result is shown in 1, where the data $S$ is represented as bold dots, with red representing $y = 0$ and green representing $y = 1$. The domain is partitioned into red and green regions, these represent the areas assigned values of 0 and 1 by the procedure $h_{S,3}$, respectively.

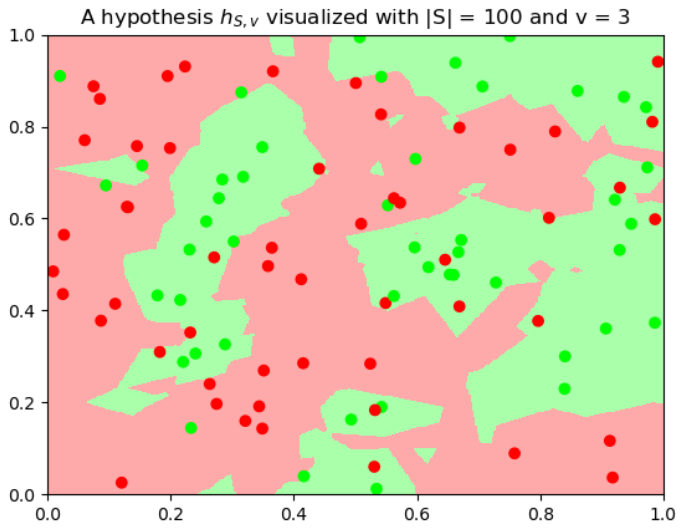



Figure 1: Hypothesis $h_{S,3}$ visualized over $|S| = 100$ and $v = 3$

### Question 7

**(a)** The average generalisation error over 100 runs is plotted in 2. The classifier is trained on 4000 training points and tested on 1000 test points for each run.

**(b)** There is a general decrease in generalisation error from $k = 1$ to $k = 10$. This can be reasoned by a decrease in overfitting, as the smaller the $k$ the more irregular the decision boundary. The

generalisation error tends to increase as $k$ becomes larger than 10. This can be reasoned by training data points physically further away from the test point influencing the classification of said point with larger $k$. When the points are further away, they're less likely of being the same class, and hence not useful for use as a neighbour for classification. We can also see that from $k = 1$ to $k = 8$ the generalisation error for even $k$ is larger than its preceding odd counterpart. The reason for this is because of the 'corner' cases; corner cases occur when there is no clear majority of $k$ nearest neighbours to $\mathbf{x}$, the corresponding predicted $y$ value is sampled uniformly at random from $\{0, 1\}$. This can only occur for even $k$, hence the lower generalisation error (as some of the points may be predicted randomly instead of by $k$-nn. The effect of corner cases is not evident for $k > 9$.

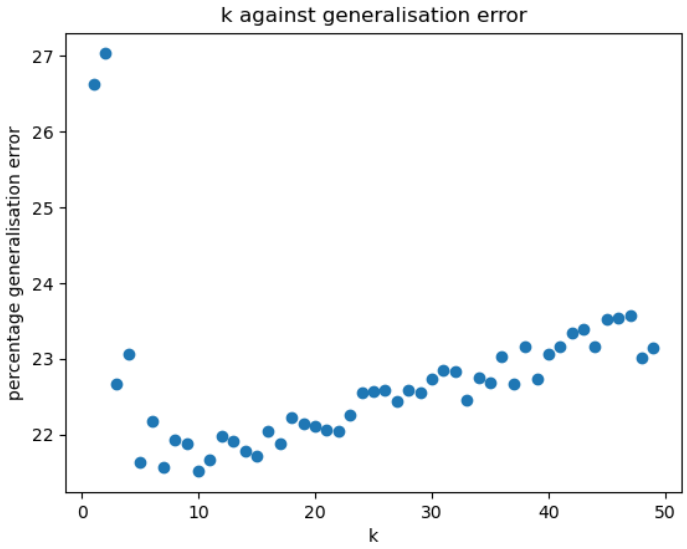



Figure 2: Average generalisation error over 100 runs plotted against $k$

## Question 8

The $k$ that gave the lowest generalisation error (averaged over 100 runs) is plotted against the number of training points in 3. The trend seen is that the value of optimal $k$ increases with the number of training points. This can be explained by the ratio of the number of training points can indicate whether we're in the overfitting or underfitting regimes. Large $\frac{m}{k}$ indicates overfitting (as a small $k$ does not consider enough data points to correctly classify); small $\frac{m}{k}$ indicates underfitting (as a large k with small m considers a larger proportion of 'incorrect' neighbours when classifying a point). To keep $\frac{m}{k}$ away from the overfitting/underfitting regimes, $k$ must increase at a similar rate as $m$ increases.

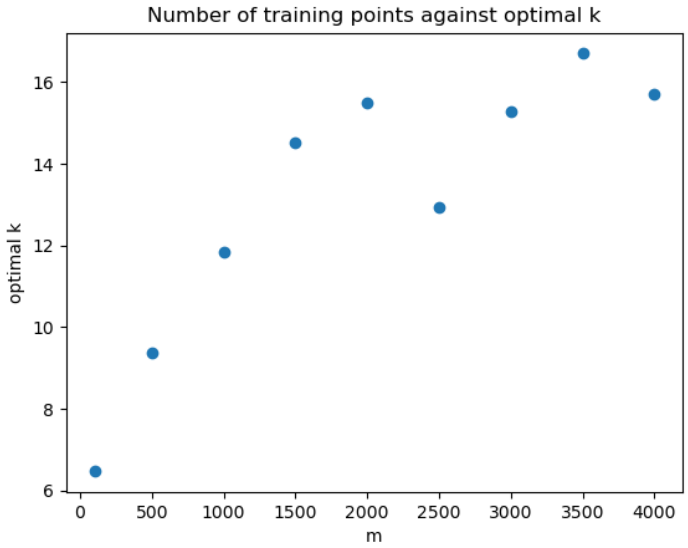



Figure 3: Number of training points plotted against value of $k$ that minimizes generalization error

# Part 3

## Question 9

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

### Part (a)

By definition, $K_c$ is positive semidefinite, if $\sum_{i,j=1}^{N} a_i a_j K_c(\mathbf{x}_i, \mathbf{x}_j) \geq 0$, $\forall N \in \mathbb{Z}^+$, $\forall a_1, \ldots, a_N \in \mathbb{R}$, $\forall \mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathbb{R}^n$. We first prove that the function $K(\mathbf{x}, \mathbf{z}) = \sum_{i=1}^{n} x_i z_i$ is positive semidefinite. We write $\sum_{i=1}^{n} x_i z_i$ as $\mathbf{x}^\top \mathbf{z}$,

$$\begin{aligned}
\sum_{i,j=1}^{N} a_i a_j \mathbf{x}_i^\top \mathbf{x}_j &= \sum_{i=1}^{N} a_i \mathbf{x}_i^\top \sum_{j=1}^{N} a_j \mathbf{x}_j \\
&= \left( \sum_{i=1}^{N} a_i \mathbf{x}_i \right)^\top \left( \sum_{j=1}^{N} a_j \mathbf{x}_j \right) \\
&= \left\| \sum_{i=1}^{N} a_i \mathbf{x}_i \right\|^2 \\
&\geq 0, \forall\ a_i, \mathbf{x}_i
\end{aligned}$$

In order for $K_c$ to be positive semidefinite, we require

$$\sum_{i,j=1}^{N} a_i a_j (c + \mathbf{x}_i^\top \mathbf{x}_j) \geq 0$$

$$c \sum_{i,j=1}^{N} a_i a_j + \sum_{i,j=1}^{N} a_i a_j \mathbf{x}_i^\top \mathbf{x}_j \geq 0$$

$$c \left\| \sum_{i=1}^{N} a_i \right\|^2 + \left\| \sum_{i=1}^{N} a_i \mathbf{x}_i \right\|^2 \geq 0$$

$$c \geq - \frac{\left\| \sum_{i=1}^{N} a_i \mathbf{x}_i \right\|^2}{\left\| \sum_{i=1}^{N} a_i \right\|^2}$$

$$c \geq -\min \left( \frac{\left\| \sum_{i=1}^{N} a_i \mathbf{x}_i \right\|^2}{\left\| \sum_{i=1}^{N} a_i \right\|^2} \right)$$

$$c \geq 0$$

Hence the condition on $c$ is found ($c \geq 0$).

**Part (b)**

Kernelized least squares regression, with the kernel $K_c$, minimizes the loss function

$$l = \sum_{i=1}^{N} \left( y_i - \sum_{j=1}^{N} \alpha_j K_c(\mathbf{x}_i, \mathbf{x}_j) \right)^2 = \|\mathsf{y} - \mathbf{K}\alpha\|^2$$

where the predictor of $y(\mathbf{x})$ is $f(\mathbf{x}) = \sum_{j=1}^{n} \alpha_j K_c(\mathbf{x}, \mathbf{x}_j)$, $\alpha$ is a vector of weights to be determined, $\mathbf{K}$ is a matrix where $\mathbf{K}_{ij} = K_c(\mathbf{x}_i, \mathbf{x}_j)$ and $N$ is the number of data points. We now minimize $l$ wrt $\alpha$.

$$\vec{\boldsymbol{\nabla}}_\alpha l = \vec{\boldsymbol{\nabla}}_\alpha \|\mathsf{y} - \mathbf{K}\alpha\|^2 = 0$$

$$-2\mathsf{y}^\top \mathbf{K} + 2\alpha^\top \mathbf{K}^2 = 0$$

$$\alpha = \mathbf{K}^{-1}\mathsf{y}$$

$\mathbf{K}$ can explicitly be written $c\mathbb{1}_N + \mathsf{X}\mathsf{X}^\top$, where $\mathbb{1}_N$ is the $N \times N$ identity matrix and $\mathsf{X}$ is the matrix formed by stacking the rows of $\mathbf{x}$ data. If $N > n$ or some of the $\mathbf{x}$ values are not linearly independent, then $\mathsf{X}\mathsf{X}^\top$ is not invertible. If $\mathsf{X}\mathsf{X}^\top$ is 'close' to being invertible then the computation of its inverse can be numerically unstable. The addition of $c$ to the diagonals acts to ensure that $\mathbf{K}$ is invertible and that the computation of its inverse is numerically stable. $c$ acts as a REGULARIZER.

## Question 10

In performing linear regression with the Gaussian kernel $K_\beta(\mathbf{x}, \mathbf{t}) = \exp(-\beta \|\mathbf{x} - \mathbf{t}\|^2)$ on the given dataset, we minimize the loss function

$$l = \sum_{i=1}^{m} \left( y_i - \sum_{j=1}^{m} \alpha_j K_\beta(\mathbf{x}_i, \mathbf{x}_j) \right)^2 = (\mathsf{y} - \mathbf{K}\boldsymbol{\alpha})^\top (\mathsf{y} - \mathbf{K}\boldsymbol{\alpha})$$

with respect to $\boldsymbol{\alpha}$ (a vector of weights to be determined). $\mathbf{K}$ is a matrix where $\mathbf{K}_{ij} = K_\beta(\mathbf{x}_i, \mathbf{x}_j)$. Minimizing $l$ wrt $\boldsymbol{\alpha}$

$$\begin{aligned} \vec{\nabla}_{\boldsymbol{\alpha}} l = \vec{\nabla}_{\boldsymbol{\alpha}} (\mathsf{y} - \mathbf{K}\boldsymbol{\alpha})^\top (\mathsf{y} - \mathbf{K}\boldsymbol{\alpha}) &= 0 \\ -2\mathsf{y}^\top \mathbf{K} + 2\boldsymbol{\alpha}^\top \mathbf{K}^2 &= 0 \\ \boldsymbol{\alpha} &= \mathbf{K}^{-1}\mathsf{y} \end{aligned}$$

The decision function can now be written as

$$f(\mathbf{t}) = \sum_{i,j=1}^{m} \left(\mathbf{K}^{-1}\right)_{ij} y_j K_\beta(\mathbf{t}, \mathbf{x}_i)$$

We now consider the values of the elements of $\mathbf{K}$. The diagonal elements are all 1, and in the limit of $\beta \to \infty$, all the off diagonal elements $\to 0$. Hence in this limit $\mathbf{K}$ is the identity matrix, and it's inverse is also the identity matrix. In this limit, the decision function can now be written

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

If we define the nearest neighbour of $\mathbf{t}$ to be $\mathbf{x}_a$, and the next-nearest neighbour of $\mathbf{t}$ to be $\mathbf{x}_b$, then we can write

$$f(\mathbf{t}) = y_a K_\beta(\mathbf{t}, \mathbf{x}_a) + \sum_{i \neq a}^{m} y_i K_\beta(\mathbf{t}, \mathbf{x}_i) \tag{2}$$

$$\leq y_a K_\beta(\mathbf{t}, \mathbf{x}_a) + K_\beta(\mathbf{t}, \mathbf{x}_b) \sum_{i \neq a}^{m} y_i \tag{3}$$

To approximate the 1-NN classifier, we would like $\text{sign}(f(\mathbf{t})) = \text{sign}(y_a)$. The extreme case of this model to put bounds on beta would be in the case where $y_i = -y_a \; \forall \; i \neq a$. Then 3 can be written

$$f(\mathbf{t}) \leq y_a K_\beta(\mathbf{t}, \mathbf{x}_a) - (m-1) y_a K_\beta(\mathbf{t}, \mathbf{x}_b) \tag{4}$$

$$= y_a \left( K_\beta(\mathbf{t}, \mathbf{x}_a) - (m-1) K_\beta(\mathbf{t}, \mathbf{x}_b) \right) \tag{5}$$

The condition of $\text{sign}(f(\mathbf{t})) = \text{sign}(y_a)$ is then satisfied for

$$K_\beta(\mathbf{t}, \mathbf{x}_a) \geq (m-1) K_\beta(\mathbf{t}, \mathbf{x}_b) \tag{6}$$

$$\exp(-\beta \left\| \mathbf{x}_a - \mathbf{t} \right\|^2) \geq (m-1) \exp(-\beta \left\| \mathbf{x}_b - \mathbf{t} \right\|^2) \tag{7}$$

$$\beta \geq \frac{\ln(m-1)}{\left\| \mathbf{x}_b - \mathbf{t} \right\|^2 - \left\| \mathbf{x}_a - \mathbf{t} \right\|^2} \tag{8}$$

Clearly this does not contradict the limit $\beta \to \infty$, and a $\beta$ exists that can simulate the 1-NN classifier. This is the most extreme case, with the classification of all points (except the nearest-neighbour) being the opposite classification of the nearest-neighbour. So clearly a $\beta$ exists in all other possible configurations of classifications of the data.

## Question 11

The state of an $n \times n$ board can be represented by a $n \times n$ matrix of 1's and 0 's, whose elements represent the holes and are '1' for a mole being up, and '0' for a mole being down.

We can represent the whacking of a mole in the $i$th row and $j$th column by the addition (modulo 2) of an $n \times n$ toggle matrix $\mathsf{T}^{(ij)}$. $\mathsf{T}^{(ij)}$ will be a matrix of all 0's except for the elements representing the holes that will be changed (toggled) by a whack to the hole in position $(i, j)$, whose elements will be 1. i.e whacking a hole is equivalent to adding 1 (modulo 2) to the elements around that hole (including the element for the hole itself). This works because in arithmetic modulo 2, $1 + 1 = 0$, meaning a hole with a mole up can be converted to a hole with a mole down. In total there are $n \times n$ toggle matrices.

We call the initial state of the board $\mathsf{M}$, the objective is then to perform additions of toggle matrices such that the resulting matrix contains all 0's. Addition of the same toggle matrix twice amounts to not changing the state of the board (due to arithmetic mod 2). This means we can formulate the problem as solving

$$\mathsf{M} + \sum_{i,j=1}^{n} c^{(ij)} \mathsf{T}^{(ij)} = \mathsf{0} \text{ (modulo 2)} \tag{9}$$

where $\mathsf{0}$ is a matrix of 0's, and $c^{(ij)} \in \{0, 1\}$ are the coefficients of the toggle matrices (they tell us which holes should be whacked, they are what we want to solve for). From this it is clear the order of whacks don't matter.

All the values we are working with are 0 or 1 with addition and multiplication defined mod 2. GF(2) (the Galois Field of order 2) is the finite field of two elements, it provides a definition of arithmetic mod 2. The elements of GF(2) can be identified with the values 0 and 1. If we say everything in equation 9 is from GF(2) we don't have to explicitly state 'modulo 2' from now on. It is easy to see that $\mathsf{M} + \mathsf{M} = 0$, so if we add $\mathsf{M}$ to both sides, we obtain

$$\sum_{i,j=1}^{n} c^{(ij)} \mathsf{T}^{(ij)} = \mathsf{M} \tag{10}$$

Performing a reshaping via a row stack on the matrices, we convert them to $n \times n$ dimensional vectors, equation 10 can then easily seen to be a system of linear equations in the form of a matrix equation:

$$\mathbf{Tc} = \mathbf{m} \tag{11}$$

where the $n^2 \times n^2$ matrix $\mathbf{T}$ is the matrix whose columns are the vectors (reshaped) $\mathsf{T}^{(ij)}$ in a particular order, $\mathbf{c}$ is a vector of elements $c^{(ij)}$ in the same order, $\mathbf{m}$ is the vector corresponding the reshaped initial board state.

As GF(2) is a field, standard Gaussian eliminaton can be used to solve for $\mathbf{a}$. The result of solving this equation gives a vector whose non-zero elements' position correspond to the the holes that need to be whacked.

**Algorithm for solution**

1. Compute toggle vectors (reshaped toggle matrices) for every button. There are $\mathrm{O}(n^2)$ of these, each taking $\mathrm{O}(n^2)$ to compute (as need to fill in each element of the vector). This gives an $\mathrm{O}(n^4)$ runtime.

2. Gaussian elimination: If the board hole positions are labelled from 1 to $n^2$ started in the top left and going from left to right across a row before starting at the left of the row below, then the corresponding toggle vectors can be ordered in the way that they are labelled to form $\mathbf{T}$. Meaning there is a '1' on every diagonal element, which can be used as the pivot for Gaussian elimination. Start in first column:

   (a) In current ($i$th) column, select the $i$th value to be the pivot.

(b) Clear the rest of values below the pivot to 0 by adding multiples of the pivot row the rows of the values we want to set to 0. (There is a maximum of only 4 non-zero values in the rest of the column) $O(4n^2) = O(n^2)$.

(c) Go to next column, then start at (a) again

Need to do this for each column ($n^2$ times), So Gaussian elimination step in total is $O(n^4)$.

3. Solve by substituting solution of bottom row, into equation of row above consecutively until all values for **c** are found. (Order of this step is less than both the others, so irrelevant). The positions of the non-zero elements of **c** are the solution to the problem.

Algorithm is $O(n^4)$, which is polynomial in $n$