Supervised Learning Coursework 1

Team : 21093512, 18091452

May 10, 2022

1 Part 1

1.1 Linear Regression

A code implementation of this section can be found in the `Part_1_1.ipynb` notebook.

1.1.1 Polynomial fitting

(a.)

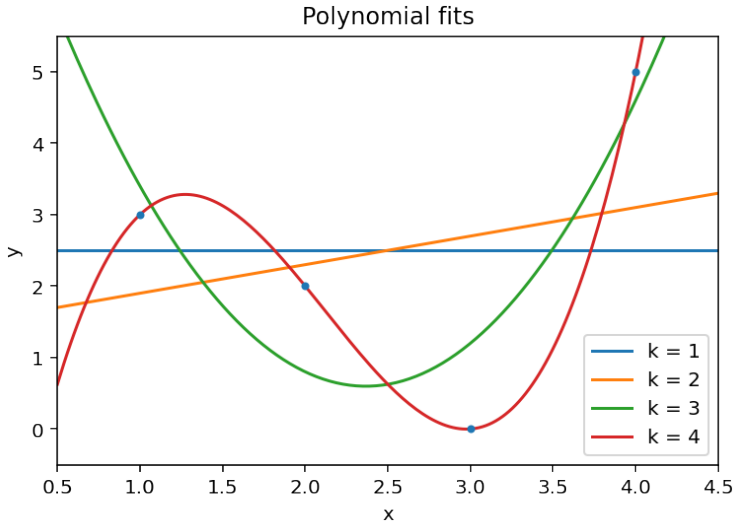

Figure 1: Polynomial fits for the dataset $\{(1,3),(2,2),(3,0),(4,5)\}$

(b.)

Equations of the curves plotted above are:

- for $k=1$: $y = 2.5$
- for $k=2$: $y = 0.5 + 0.4x$
- for $k=3$: $y = 9 + -7.1x + 1.5x^2$
- for $k=4$: $y = -5 + 15.16x - 8.5x^2 + 1.33x^3$

(c.)

The mean squared errors are:

- for $k=1$: $MSE = 3.25$
- for $k=2$: $MSE = 3.05$
- for $k=3$: $MSE = 0.8$
- for $k=4$: $MSE = 0$

1.1.2 Illustrating overfitting

(a.)

Here we plot 30 points generated by the function $g_\sigma := sin^2(2\pi x) + \epsilon$ with $\sigma = 0.07$ and fit some simple polynomials to it.
(i)

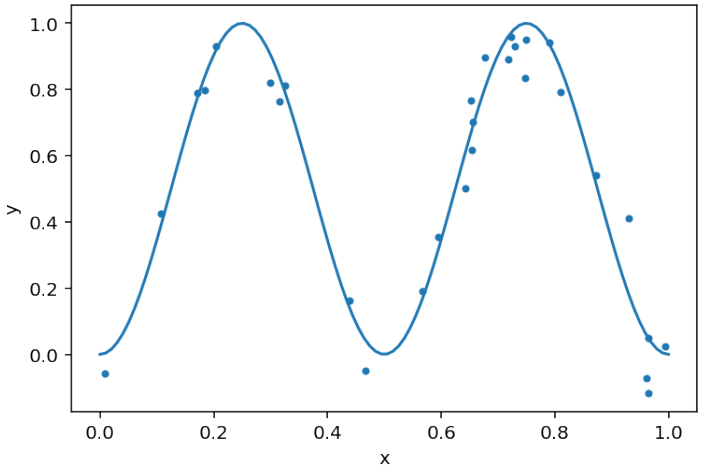

Figure 2: $g_{0.07} := sin^2(2\pi x) + \epsilon$ plotted with 30 points randomly generated from the function

(ii)

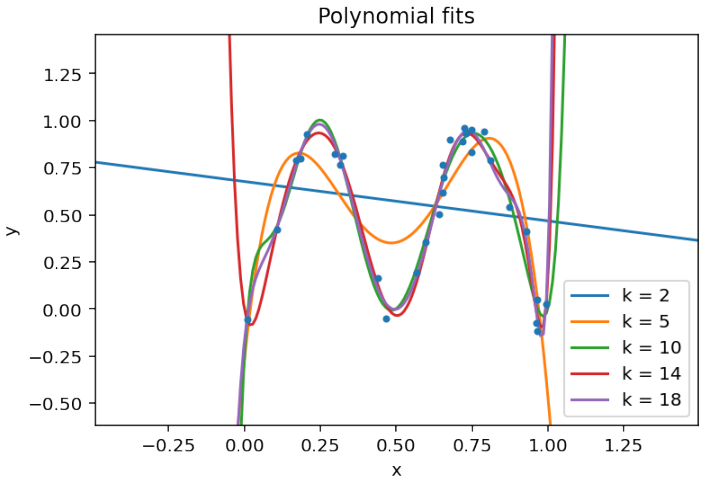

Figure 3: Polynomial fits of a random dataset generated by $g_{0.07} := sin^2(2\pi x) + \epsilon$

(b.)

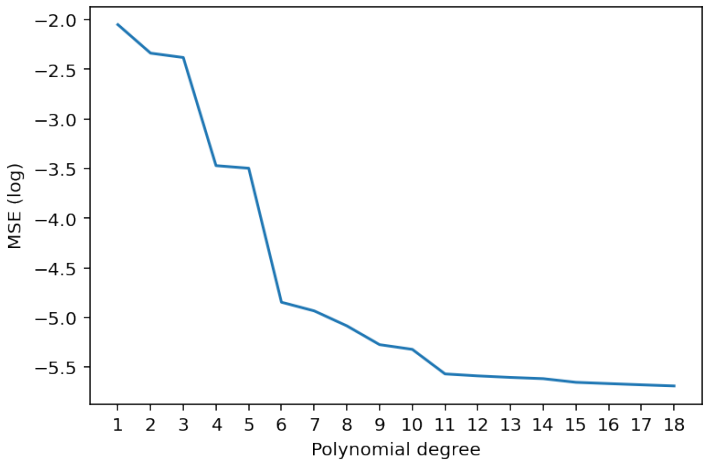

Figure 4: Training error of different polynomial fits (expressed as natural logarithm of the MSE).

(c.)

Note: here and in section 1.1.3, the test set is generated by sampling 1000 new points from $g_{0.07} \coloneqq sin^2(2\pi x) + \epsilon$

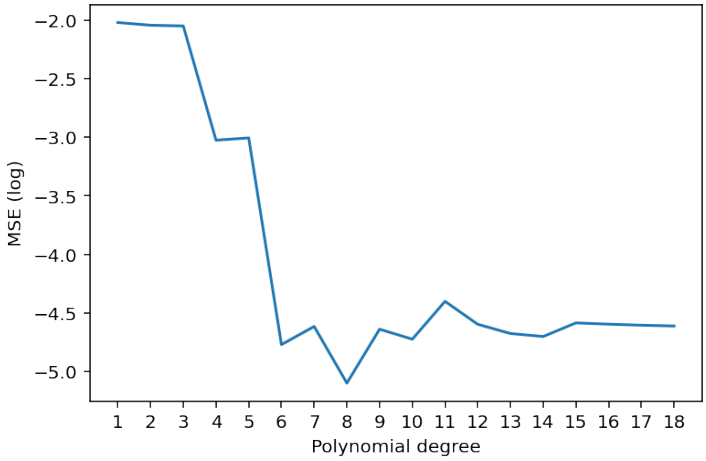

Figure 5: Test error of different polynomial fits (expressed as natural logarithm of the MSE)

(d.)

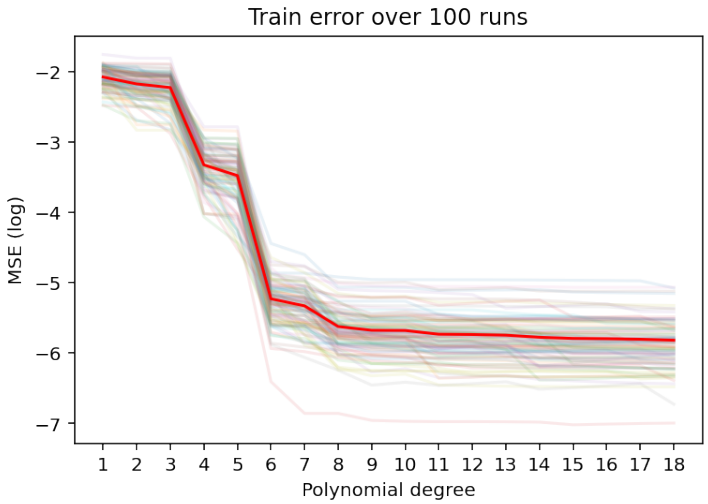

Figure 6: Training error of different polynomial fits (expressed as natural logarithm of the MSE) averaged over 100 runs. Red line is the mean across runs.

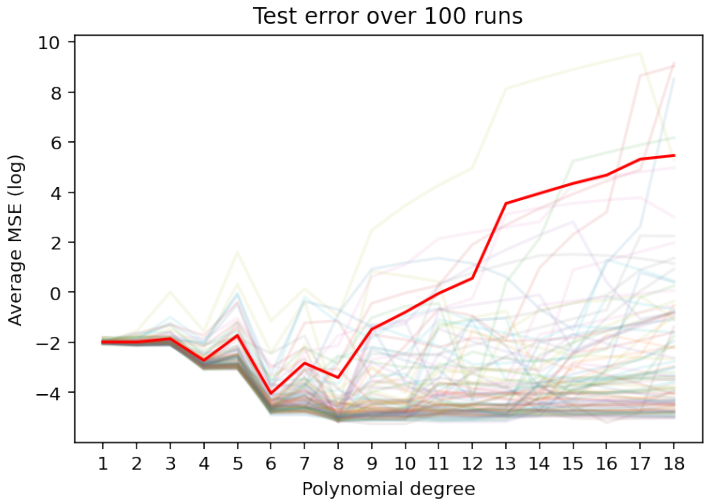

Figure 7: Test error of different polynomial fits (expressed as natural logarithm of the MSE) averaged over 100 runs. Red line is the mean across runs. A prominent feature is the high variance exhibited by higher order fits.

1.1.3 Sinusoidal basis functions

In this section we fit the same dataset type as in Figure 2 with different sinusoidal basis functions $\{sin(1\pi x), sin(2\pi x), sin(3\pi x), ..., sin(k\pi x)\}$ for $k = 1, 2, ..., 18$.

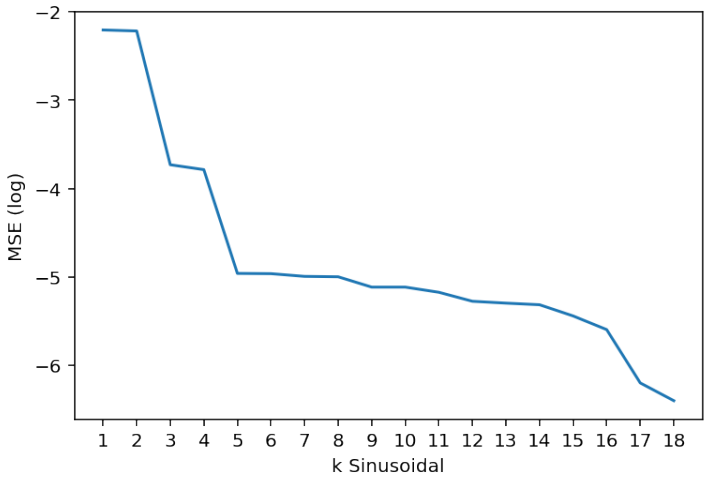

Figure 8: Training error of different sinusoidal fits (expressed as natural logarithm of the MSE).

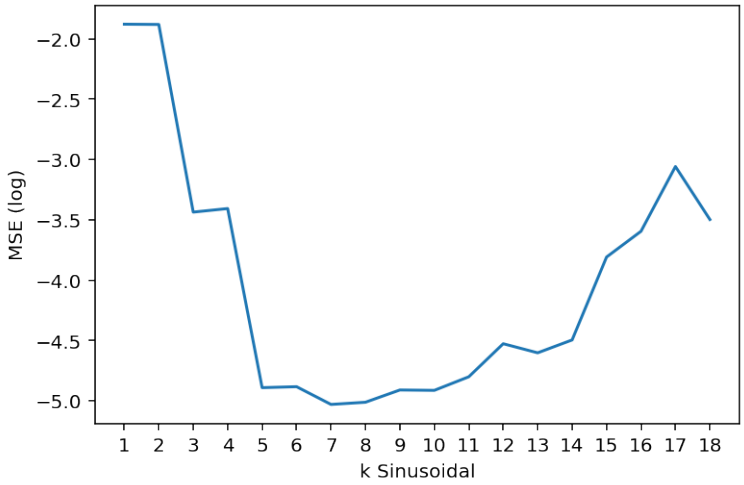

Figure 9: Test error of different sinusoidal fits (expressed as natural logarithm of the MSE)

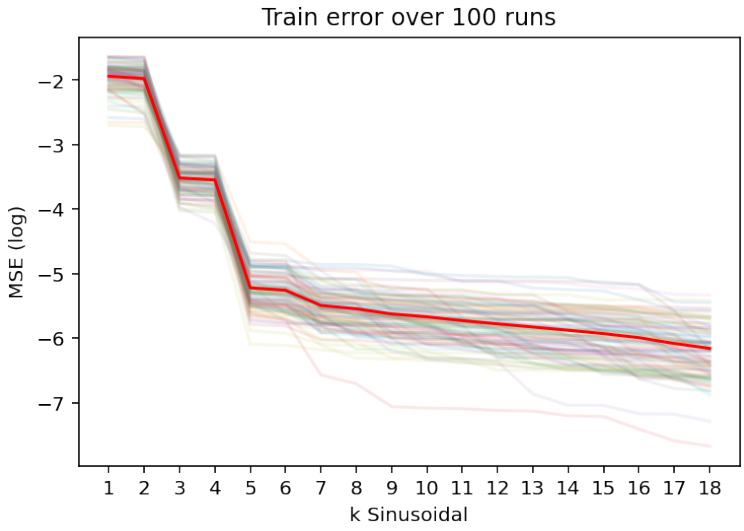

Figure 10: Training error of different sinusoidal fits (expressed as natural logarithm of the MSE) averaged over 100 runs. Red line is the mean across runs.

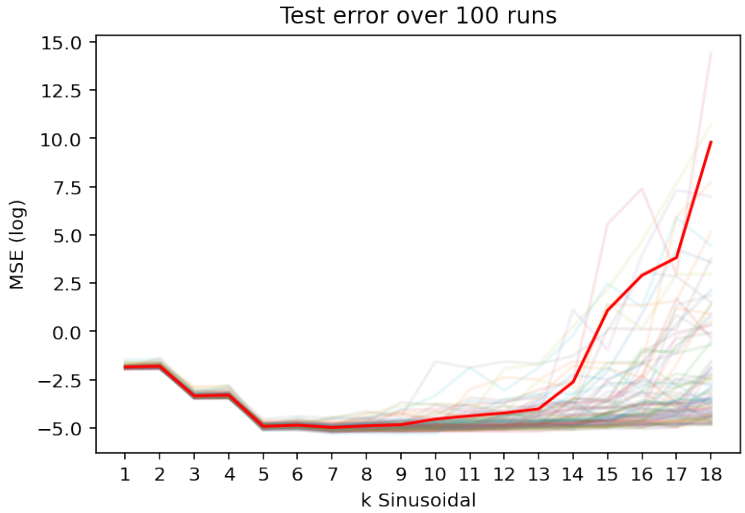

Figure 11: Test error of different sinusoidal fits (expressed as natural logarithm of the MSE) averaged over 100 runs. Red line is the mean across runs. Once again, "higher order" sinusoids increase the variance of the model.

1.2 Filtered Boston housing and kernels

For the following versions of linear regression, the mean square error (MSE) between observed values y_i and predicted values $\hat{y}_i$ is calculated using equation 1. All error values are cited in table 1 below.

$$\textbf{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{1}$$

1.2.1 Naive Linear Regression

The Naive Linear Regression uses a vector of ones that is the same length as the training set and the test set. By using these vectors the data is fitted with a constant function.

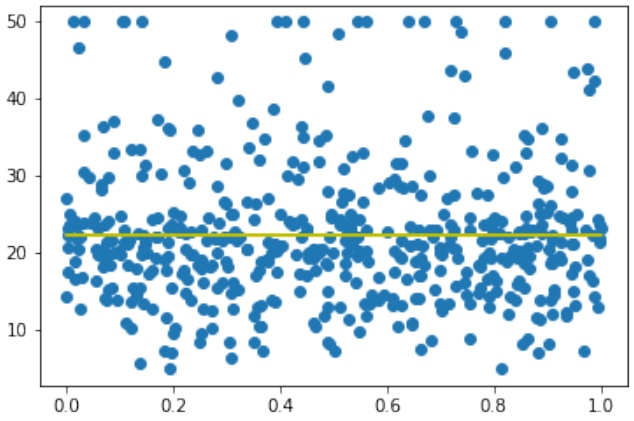

Figure 12: Naive regression fit on median house price

Figure 12 illustrates the Naive Regression. The fitted line is constant at approximately $y = 22.$. This fit seems to be at the average point of all the data points we are trying to fit. Equation 2 proves this intuition.

$$\hat{\theta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} = \left(\begin{bmatrix} 1 & 1 & \dots & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ \dots \\ 1 \end{bmatrix} \right)^{-1} \begin{bmatrix} 1 & 1 & \dots & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_N \end{bmatrix} = \frac{1}{N}\sum_{i=1}^{N} y_i \tag{2}$$

When replacing the training and testing data values to *one* values, the resultant $\hat{\theta}$ will be the mean of the observed values, y_N.

1.2.2 Single-Attribute Linear Regression

Figure 13 illustrates the linear fit on median house price data using single attributes.

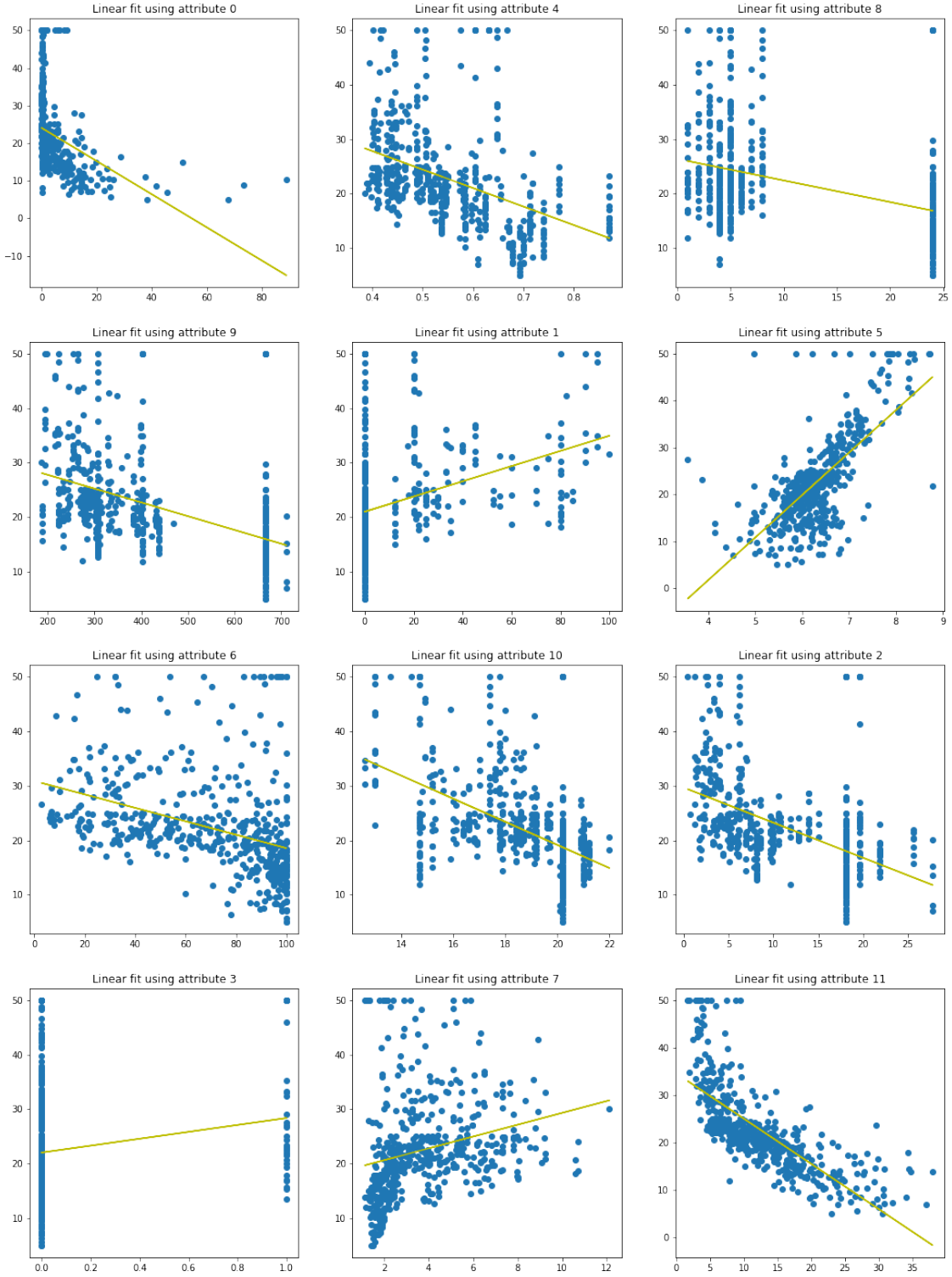

Figure 13: Linear regression fit on median house price using individually 12 attributes

1.2.3 Multi-Attribute Linear Regression

When we perform linear regression using all of the data attributes at once, the prediction outperforms the previous regressors (see table 1 for mean sqaured error)

1.3 Kernelised ridge regression

In this section, we use the Gaussian kernel,

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

to perform Kernel ridge regression on the filtered Boston dataset. Details about the implementation of Gaussian kernel ridge regression can be seen in the `Part_1_3_Kernel.ipynb` notebook.

(a.)

The best values of γ and σ computed with 5-fold cross validation are respectively $\gamma = 2^{-29}$ and $\sigma = 2^{9.5}$.

(b.)

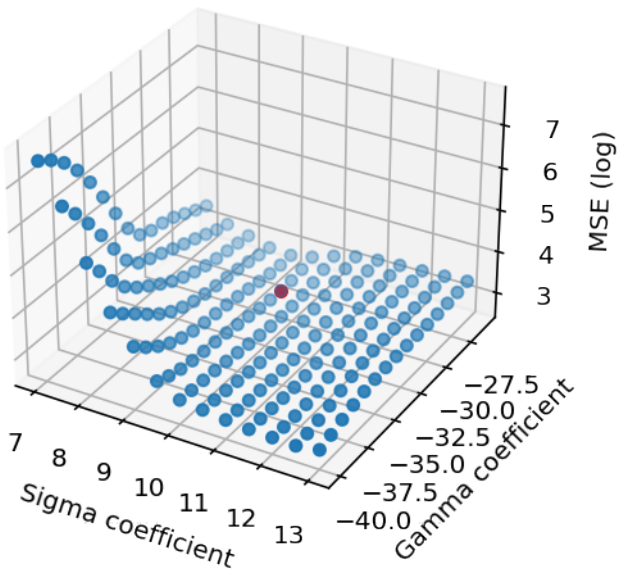

Figure 14: Mean cross-validation error (natural logarithm of MSE on the validation set) as a function of γ and σ. The red dot shows the minimum, which for this particular train/test split is reached at $\gamma = 2^{-29}$ and $\sigma = 2^{9.5}$. An animated version of the plot, graphed with interpolation, can be accessed in the supplementary code.

(c.)

For this particular train/test split and optimal cross-validated hyperparameters, the MSE on the train and test set are respectively $MSE_{train} = 10.35$ and $MSE_{test} = 8.38$. This shows that Gaussian kernel ridge regression performs substantially better than linear regression with all attributes on the filtered Boston dataset.

(d.)

Method	MSE train	MSE test
Naive Regression	87.69 ± 9.36	77.93 ± 8.82
Linear Regression (attribute 1)	71.59 ± 8.46	72.84 ± 8.46
Linear Regression (attribute 2)	73.39 ± 8.56	74.07 ± 8.54
Linear Regression (attribute 3)	64.68 ± 8.04	65.06 ± 8.02
Linear Regression (attribute 4)	81.99 ± 9.05	82.25 ± 9.018
Linear Regression (attribute 5)	67.41 ± 8.21	72.54 ± 8.47
Linear Regression(attribute 6)	43.71 ± 6.61	43.81 ± 6.57
Linear Regression(attribute 7)	73.65 ± 8.58	70.31 ± 8.33
Linear Regression (attribute 8)	79.04 ± 8.89	79.84 ± 8.87
Linear Regression (attribute 9)	72.62 ± 8.52	71.57 ± 8.41
Linear Regression (attribute 10)	66.27 ± 8.13	65.60 ± 8.06
Linear Regression(attribute 11)	62.54 ± 7.91	63.28 ± 7.91
Linear Regression(attribute 12)	37.08 ± 6.09	41.43 ± 6.40
Linear Regression (all attributes)	22.52 ± 4.74	23.60 ± 4.83
Kernel Ridge Regression	8.11 ± 1.10	12.74 ± 1.85

Table 1: Mean Squared Errors on the train set of different model fits to the filtered Boston dataset. Values reported are means across 20 random train/test splits, ± standard deviation.

2 Part 2

2.1 k-Nearest Neighbors

Details about the implementation of the following k-nearest neighbors protocols can be seen in the `Part_2_KNN.ipynb` notebook.

2.1.1 Data Generation

Figure 15 shows the hypothesis $h_{s,v}$ visualised with $|S| = 100$ and $v = 3$.

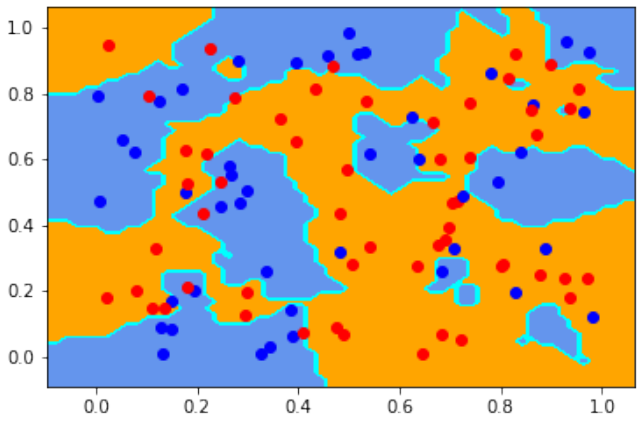

Figure 15: A hypothesis $h_{s,v}$ visualised with $|S| = 100$ and $v = 3$

The hypothesis h can produced "undefined" results in certain corner cases where neighboring data points have balanced classes. In this case the class prediction is determined uniformly at random (*see code for implementation deatils*).

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

Figure 16 provides a visualisation of Protocol A, a plot of the estimated generalization error versus increasing k values. The estimated generalization error is calculated using the mean squared error on the test set. The shape of the figure illustrates the bias-variance decomposition of the generalization error. Indeed for small values of k,

the error is high. This is due to underfitting. This is because bias is high and variance is low. When increasing k, the error drops to a minimum value, the optimal k (for $k \approx 10$). When k values get larger, the error increases slightly. This is due to overfitting. This is because bias is low and variance is high.

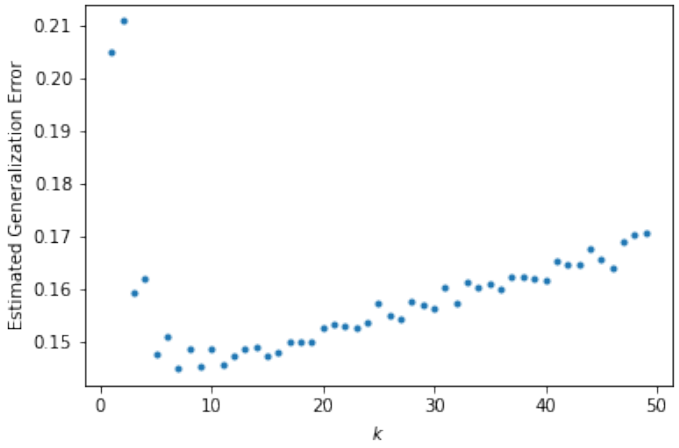

Figure 16: Plot of the estimated generalization error versus k values from 1 to 49

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

Figure 17 provides a visualisation of Protocol B, a plot of m (*training set size*) versus optimal k value.

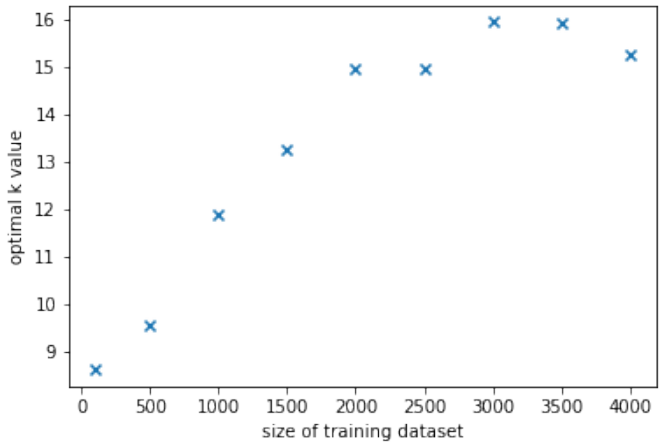

Figure 17: Plot of m (*training set size*) versus optimal k value

3 Part 3

3.1 Kernel modification

3.1.1 Context

Consider the function,

$$\mathbf{K}_c(\mathbf{x}, \mathbf{t}) = c + \sum_{i=1}^{n} x_i z_i,$$

where $\mathbf{x}, \mathbf{t} \in \mathbb{R}^n$.

3.1.2 Semidefinite kernel

Question: For what values of $c \in$ *is* $\mathbf{K}_c$ *a positive semi-definite kernel?*
Let K_1 and K_2 be functions such that $\mathbf{K}_c = K_1 + K_2$, ie.

$$K_1 = c \tag{3}$$

$$K_2 = \sum_{i=1}^{n} x_i z_i \tag{4}$$

Lemma 3.1 *A real-valued function $\mathbf{K}(x, y)$ satisfies Mercer's condition if $\sum_{i=1}^{n}\sum_{j=1}^{n}\mathbf{K}(x, y)a_i a_j \geq 0$ for all square integrable functions, a. If $\mathbf{K}(x, y)$ satisfies the condition, the existence of an underlying mapping ϕ is guaranteed, confirming that $\mathbf{K}(x, y)$ is a valid positive semi-definite kernel.*

Given that $K_1 = c$, the Mercer condition to satisfy would be of the form,

$$\sum_{i=1}^{n}\sum_{j=1}^{n} K_1 a_i a_j = \sum_{i=1}^{n}\sum_{j=1}^{n} c a_i a_j = c\sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j \geq 0$$

This condition holds if and only if, $c \geq 0$. Therefore for values of c greater than or equal to zero, K_1 is a kernel.

Lemma 3.2 *A function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, is a kernel if there exists a "feature map" $\phi : \mathcal{X} \to \mathcal{H}$ such that*

$$k(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle,$$

where $\mathcal{H}$ is a vector space and $\langle \_, \_ \rangle$ is an inner product in $\mathcal{H}$.

K_2 can be written as,

$$K_2 = \sum_{i=1}^{n} x_i z_i = \langle \mathbf{x}, \mathbf{z} \rangle, \tag{5}$$

where $\langle \mathbf{x}, \mathbf{z} \rangle$ is the dot product between $\mathbf{x}$ and $\mathbf{z}$. Given the lemma mentioned above, K_2 is therefore a valid dot product kernel $\forall x_i, z_i \in \mathbb{R}$. Precisely, K_2 is referred to as the linear kernel, and its feature map is simply the identity.

Lemma 3.3 *Given valid kernels $\mathbf{K}_a$ and $\mathbf{K}_b$, the kernel $\mathbf{K}_c$, given by, $\mathbf{K}_c = \mathbf{K}_a + \mathbf{K}_b$, is itself a valid kernel.*

K_1 and K_2 are both kernels, as shown above. Hence, we can conclude that $\mathbf{K}_c(\mathbf{x}, \mathbf{t}) = c + \sum_{i=1}^{n} x_i z_i$ is a valid kernel $\forall\ c \geq 0$.

3.1.3 $\mathbf{K}_c$ with linear regression

When using K_c in kernelised linear regression, c influences the solution by allowing to learn a bias term, similar to augmenting the design matrix with a vector of ones in standard linear regression. In other words, adding a non-negative, non-zero c to each entry of the kernel matrix amounts to adding a fixed feature to each sample. This allows the regressor to predict an accurate y-intercept. An example implementation of K_c can be accessed in the notebook `linear_kernel.ipynb`.

3.2 Gaussian Kernel

3.2.1 Context

In this question a classifier is generated by a linear regression using a Gaussian Kernel,

$$\mathbf{K}_\beta(\mathbf{x}, \mathbf{t}) = \exp^{-\beta||\mathbf{x}-\mathbf{t}||^2} \tag{6}$$

where $\mathbf{x}$ is the training data $\mathbf{t}$ is the testing data. Given the role of the kernel on the regression, this classifier depends on what parameter β is selected.

Question: In what scenario will chosen β enable the trained linear classifier to simulate a 1-Nearest Neighbor classifier trained on the same dataset?

3.2.2 Proof

For a given test point, $\mathbf{t}$, a $1NN$ classifier selects the single closest training point x_i and selects its label y_i as a predicted label. In an ideal situation, a $1NN$ classifier would perform best if the closest neighbor was considerably closer than all other neighbors.

Let $x^\star$ be the nearest neighbor of training point, $\mathbf{t}$. When decomposing the Gaussian kernel, the distance between points can be compute in the following fashion:

$$||\mathbf{x} - \mathbf{t}||^2$$

For $x^\star$, the distance will be small. For further neighbors of $\mathbf{t}$, the distance will be larger. When adding the exponential term, we get:

$$\exp^{-||\mathbf{x}-\mathbf{t}||^2} \tag{7}$$

For $x^\star$, this value will be largest. For further neighbors of $\mathbf{t}$, this value will be smaller.

As mentioned above, a $1NN$ classifier would perform best if the closest neighbor, $x^\star$, was considerably closer than all other neighbors. β should be chosen to create a larger gap in between the kernel of the closest neighbor and all the other neighboring "kernels". A single scalar β value would no suffice, as it will need to take into account the ratio of distance from other points for all training points, $\mathbf{x_i}$. In order to do so β must be a function of all the training points,

$$\beta = \hat{\beta}\big(\mathbf{x_{1:m}}, \mathbf{t}\big),$$

such that $\hat{\beta}(x^\star, t)$ is large and that $\hat{\beta}(\mathbf{x}_{1:m\setminus\{x^\star\}}, t)$ is small. Therefore, β acts as a scale parameter, tending to zero at indices of the training points that are not the nearest neighbor of the test point.

3.3 Whack-A-Mole!

3.3.1 Model Assumptions

In order to solve this problem, we have considered the $n \cdot n$ grid of holes (ie. the original configuration) as a matrix of zeros and ones. When a mole is out, the matrix value is a 1. Similarly when a mole is in, the matrix value is a 0. The original matrix will be represented as S.

Whenever a mole were to get hit the surrounding moles would either hide, if they were out, or come out, if they were hidden. In our model this scenario would be modelled by replacing the surrounding values (of the matrix) with 0 or 1 depending on their previous state. This is equivalent to adding 1 $\mod 2$.

Given that hitting a mole is equivalent to adding 1 to surrounding matrix locations (leaving the other values unchanged), we can assume that every possible location represents a matrix of what values to add to the grid (ie. a whacking matrix). Hence, the simulation of 'hitting a mole' given a $n \cdot n$ grid configuration can be represented as the addition of the grid matrix itself and a chosen whacking matrix, $W_{i,j}$. Equation 8, describes a situation where one hit the center ($T_{2,2}$) of a $3 \cdot 3$ grid configuration.

$$W_{2,2} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix}, S = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \tag{8}$$

The resultant matrix corresponds to the original grid after being hit at location $(2, 2)$.

The following model assumptions do factor in situations where an empty hole is hit as it will, as aforementioned, add 1 modulus 2 regardless of the presence of a mole. These assumptions and design decisions were influenced by the Lights Out Puzzle, a similar problem referenced in [1].

3.3.2 Model

Notation:

- Original configuration matrix : S
- whacking matrix, the shape of a whack at location (i, j) in S : $W_{i,j}$
- action coefficient, whether to whack or not at location (i, j) in S : $\alpha_{i,j}$

The action matrix, $\alpha_{i,j}$, corresponds to matrix that displays whether a hole is hit (showing a 1) or not (showing a 0)

In that case, we are trying to find a set of coefficients $\alpha_{i,j}$ such that,

$$S_{i,j} + \sum_{i,j} \alpha_{i,j} W_{i,j} = 0 \mod 2 \tag{9}$$

In order to compute this objective we consider that our data is generated using truth tables for boolean **XOR** and **AND**, shown below in figure .

A	B	A **XOR** B	A **AND** B
0	0	0	0
0	1	1	0
1	1	0	1
1	0	1	0

Figure 18: Table for boolean XOR and AND

The **XOR** table will be used for addition and the **AND** table for multiplication. This will allow the model to process values modulo 2 and inverse calculations (arithmetic inverse = multiplicative inverse). Hence, assuming data in drawn from these truth tables, we can write our objective as the following.

$$\begin{aligned} S_{i,j} + \sum_{i,j} \alpha_{i,j} W_{i,j} = 0 \mod 2 &\equiv S_{i,j} + \sum_{i,j} \alpha_{i,j} W_{i,j} = 0 \\ &\equiv \sum_{i,j} \alpha_{i,j} W_{i,j} = -S_{i,j} \\ &\equiv \sum_{i,j} \alpha_{i,j} W_{i,j} = S_{i,j},\ \forall x \in \textbf{AND} \end{aligned} \tag{10}$$

In order to compute this we need to flatten both α and W matrices in row-major ordering. The resultant flattened matrices will be:

- $V_{i,j}$: matrix of flattened $W_{i,j}$; same shape as $W_{i,j}$, with flattened whacking configurations instead of matrices.
- a: matrix of flattened $\alpha_{i,j}$

Using this notation, we can represent the objective as the following system of linear equations, shown in figure 11. The system can be solved for a using Gaussian elimination.

$$Va = S \tag{11}$$

3.3.3 Unsolvable patterns

To solve this problem we must solve equation 11. To do so, both sides of the equation must be multiplied by the inverse of V, V^{-1}. In order to compute its inverse, V must have full rank.

If this is not the case, the initial grid will not be solvable. For example the following $3 \cdot 2$ grid does not have a solution.

$$V = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

In addition, there are some conditions where a is not winnable, meaning that there exists no set of whacking configurations that allow to remove all moles from the grid.

3.3.4 Computational Complexity

The input of size $n \cdot m$ is of $\mathcal{O}(nm)$. It will cost $\mathcal{O}(n^3m^3)$ to run the described model on a this grid. The following points break down explain the steps of the algorithm and each of the computational costs.

- Find location of the whacking matrices ($\mathcal{O}(nm)$) and compute ($\mathcal{O}(nm)$) them : $\mathcal{O}(n^2m^2)$
- Perform Gaussian elimination on linear system. At each step, we need to scan the matrix for a column containing a mole (a *1* value) to use as a pivot. This requires $\mathcal{O}(nm)$ per column for $\mathcal{O}(nm)$ total columns: $\mathcal{O}(n^2m^2)$.
- When column is found, we clear all other columns. We are actively looking at all elements of the of this matrix ($\mathcal{O}(n^2m^2)$), column by column ($\mathcal{O}(nm)$). This will cost $\mathcal{O}(n^3m^3)$ and will be the most computationally costly steps in the algorithm. Hence, the algorithm an overall computational complexity of $\mathcal{O}(n^3m^3)$, making it polynomial in n.

References

[1] Lights Out Puzzle. https://mathworld.wolfram.com/lightsoutpuzzle.html.