Supervised Learning: Coursework 1

Sage Bergerson, Enric Balaguer Rodon

November 10, 2023

Contents

1 Part I

1.1 Linear Regression

1 Exercise 1

1.a

We transform the given data with polynomial bases of dimension $k = \{1, 2, 3, 4\}$ and implement linear regression in order to fit curves to the data for each k.

Functions:

- poly_basis
  - Input: dataset and polynomial dimension
  - Operation: transforms data with polynomial basis such that $\{\phi_1(\mathbf{x}) = 1, \phi_2(\mathbf{x}) = \mathbf{x}, \ldots, \phi_k(\mathbf{x}) = \mathbf{x}^{k-1}\}$
  - Output: transformed dataset
- lstsq
  - Input: dataset and targets
  - Operation: computes a weight vector using the equation $\mathbf{w} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{y}$
  - Output: weight vector
  - *Note: We use the pseudoinverse in order to avoid issues with extremely small values raising singular matrix error*

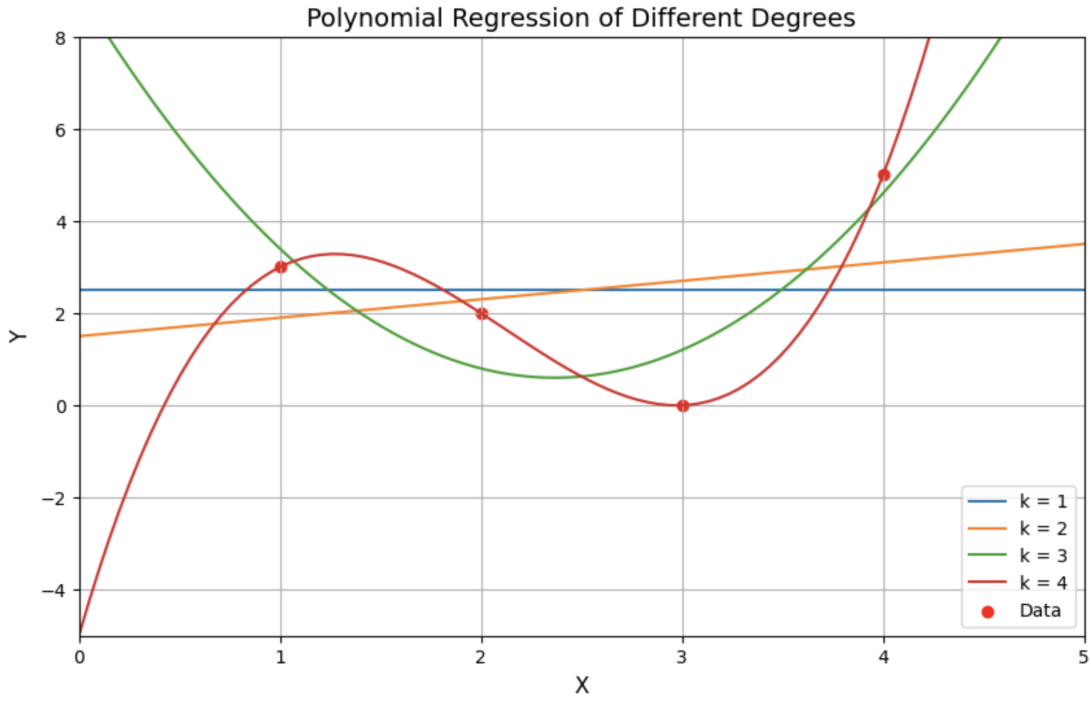

Figure 1: Plot of polynomials of degrees {1, 2, 3, 4} fit to the data.

1.b

We extract the weights computed during linear regression for each dimension in each polynomial $k = \{1, 2, 3, 4\}$ in order to construct the corresponding equations. The resulting equations are:

- $k = 1 : y = 2.5$
- $k = 2 : y = 1.5 + 0.4x$
- $k = 3 : y = 9.0 + -7.1x + 1.5x^2$
- $k = 4 : y = -5.0 + 15.17x + -8.5x^2 + 1.33x^3$

1.c

To compute the mean squared error (MSE) for each polynomial basis of degrees $k = \{1, 2, 3, 4\}$, we first compute the equations given above to find the predicted targets. For each result we compute the MSE as $MSE = \sum_{i=1}^{m}(y_i' - y_i)^2/m$ where y' are the predicted targets, y the ground truth targets, and m the number of data points. The resulting MSEs are:

- $k = 1 : 3.25$
- $k = 2 : 3.05$
- $k = 3 : 0.8$
- $k = 4 : 1.6 \times 10^{-23}$

2 Exercise 2

2.a

2.i

We plot the function $\sin^2(2\pi\mathbf{x})$ with data points sampled uniformly at random from the interval $[0, 1]$ and transformed by the function $g_\sigma(\mathbf{x}) = \sin^2(2\pi\mathbf{x}) + \epsilon$ where ϵ is a normally distributed random variable with mean 0 and variance σ^2 which adds noise to the data.

Functions:

- g_sigma
  - Input: dataset and σ value
  - Operation: sample values of ϵ and transforms data with basis $g_\sigma(\mathbf{x}) = \sin^2(2\pi\mathbf{x}) + \epsilon$
  - Output: transformed dataset

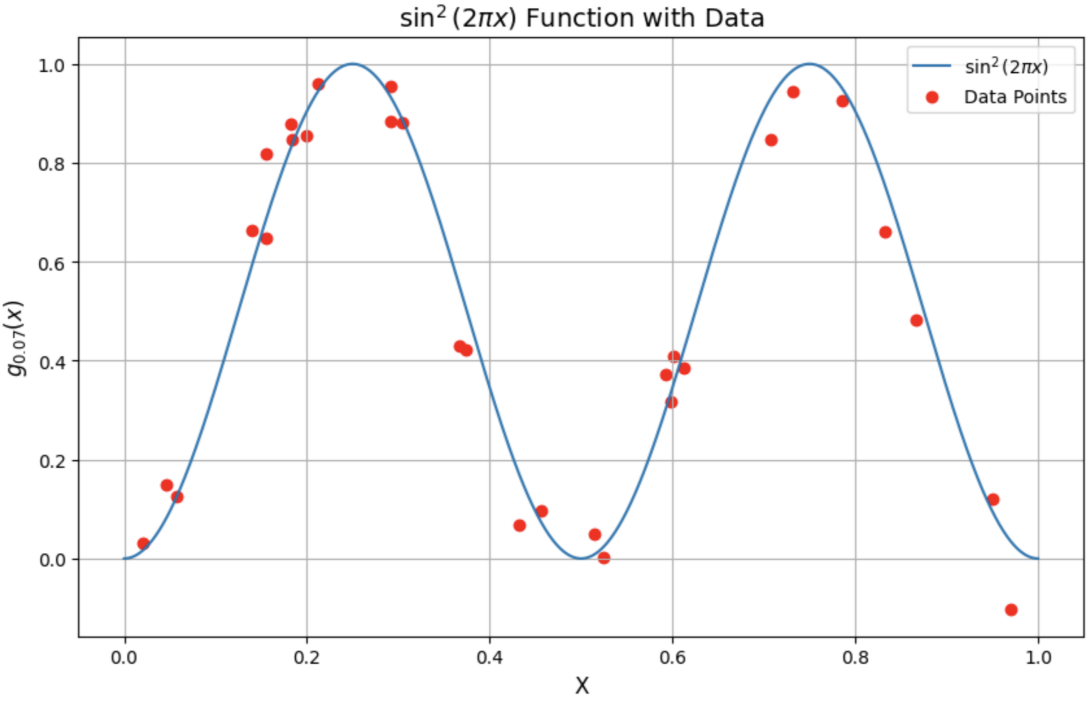

Figure 2: Underlying $\sin^2(2\pi \mathbf{x})$ function with data sampled from $\sin^2(2\pi \mathbf{x}) + \epsilon$

2.ii

We then fit the data with polynomial bases of dimensions $k = \{2, 5, 10, 14, 18\}$. We calculate weight vectors for each k given the data, and then use the resulting coefficients to construct x and y values for each curve.

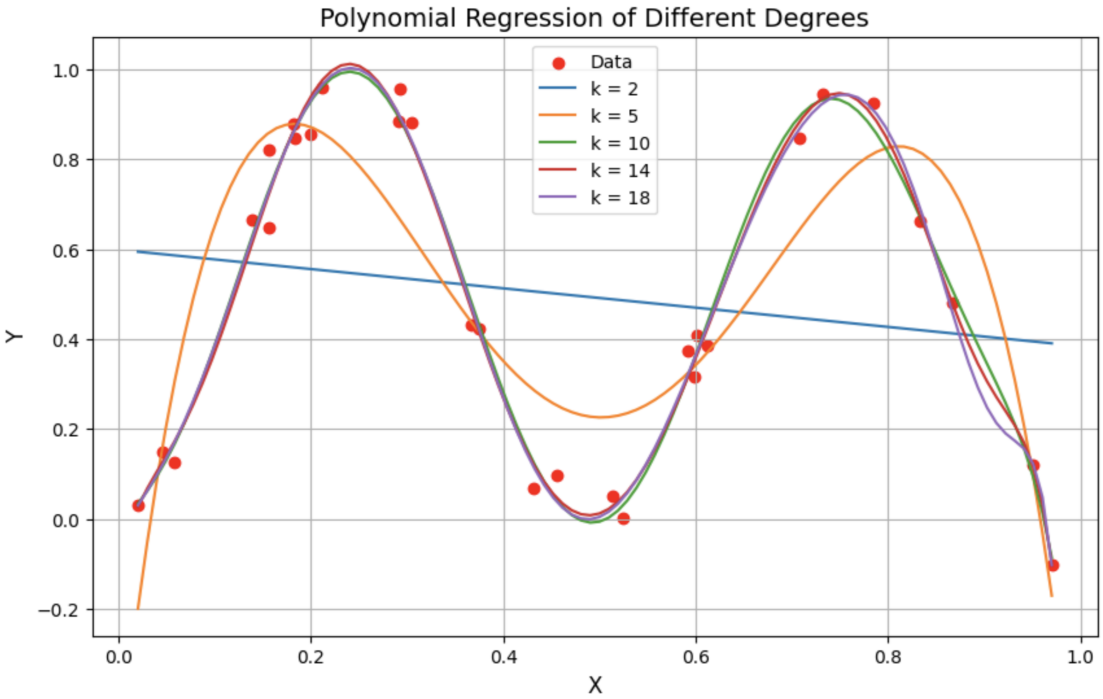

Figure 3: Polynomials of degrees {2, 5, 10, 14, 18} fit to data.

2.b

We compute the natural log (ln) of the training error for polynomial functions of degrees $k = \{1, \ldots, 18\}$ fit to the previously sampled data.

For each degree we transform the data with a polynomial basis, compute the weights based on the ground truth targets, and then predict targets using the data and weights. For each result we compute the MSE as $MSE = \sum_{i=1}^{m}(y_i' - y_i)^2/m$ where y' are the predicted targets, y the ground truth targets, and m the number of data points. We then take the ln of this value.

The resulting plot shows decreasing ln(MSE) with increasing degree k.

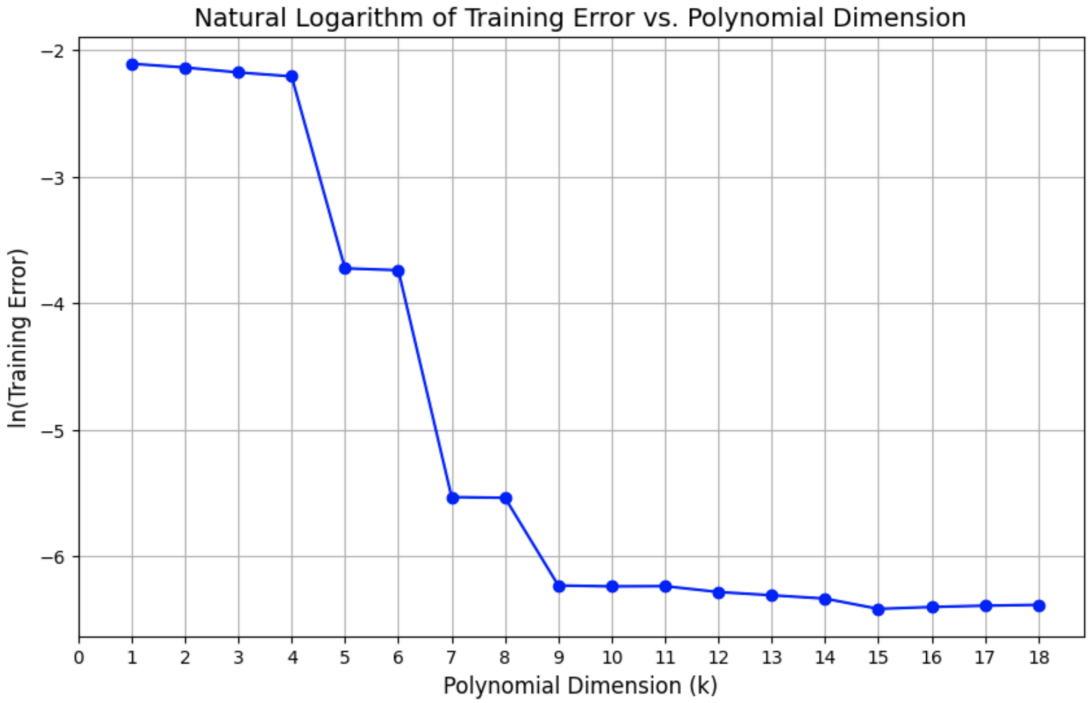

Figure 4: Natural log of training error for polynomials of degrees {1,...,18} fit to data.

2.c

We compute the natural log (ln) of the test error for polynomial functions of degrees $k = \{1, \ldots, 18\}$ fit to the previously sampled data and tested on a new test set sampled from the same distribution.

We draw a new test set of 1000 data points uniformly at random from the interval $[0, 1]$ and transform them using the function $g_\sigma(\mathbf{x}) = \sin^2(2\pi\mathbf{x}) + \epsilon$. We compute weights based on our training data, transform our data with a polynomial basis, and then predict targets using our weights and testing data. For each result we compute the MSE as MSE $= \sum_{i=1}^{m}(y_i' - y_i)^2/m$ where y' are the predicted targets, y the ground truth targets, and m the number of data points. We then take the ln of this value.

The resulting plot shows decreasing ln(MSE) with increasing degree k until $k = 7$, after which point the ln(MSE) again increases.

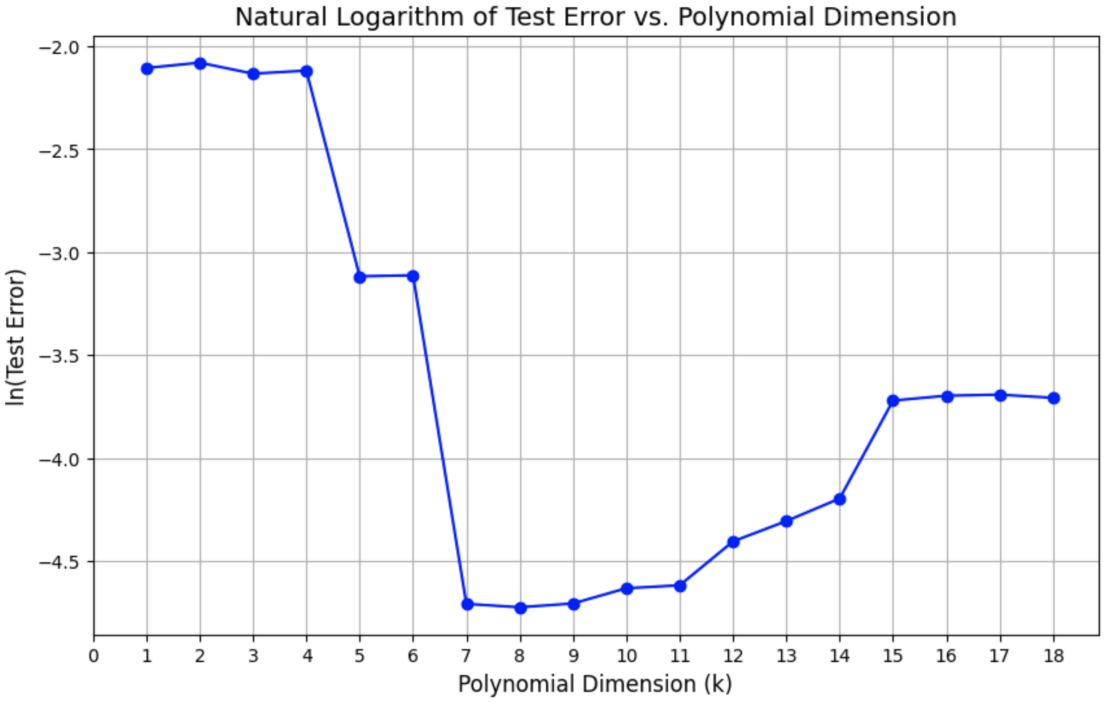

Figure 5: Natural log of test error for polynomials of degrees {1,...,18} fit to data.

2.d

We repeat the sampling and computation procedures in items (b) and (c).

However, for (b) we now sample a new training set for each iteration and repeat the process 100 times, taking the ln or the resulting average training error for each degree. For (c) we generate a new training and test set for each iteration and similarly repeat the process 100 times, taking the ln or the resulting average test error for each degree.

Our resulting plot for training error decreases similarly to the plot in (b), while our plot for test error shows a more dramatic increase in error after $k = 7$ compared to our plot in (c).

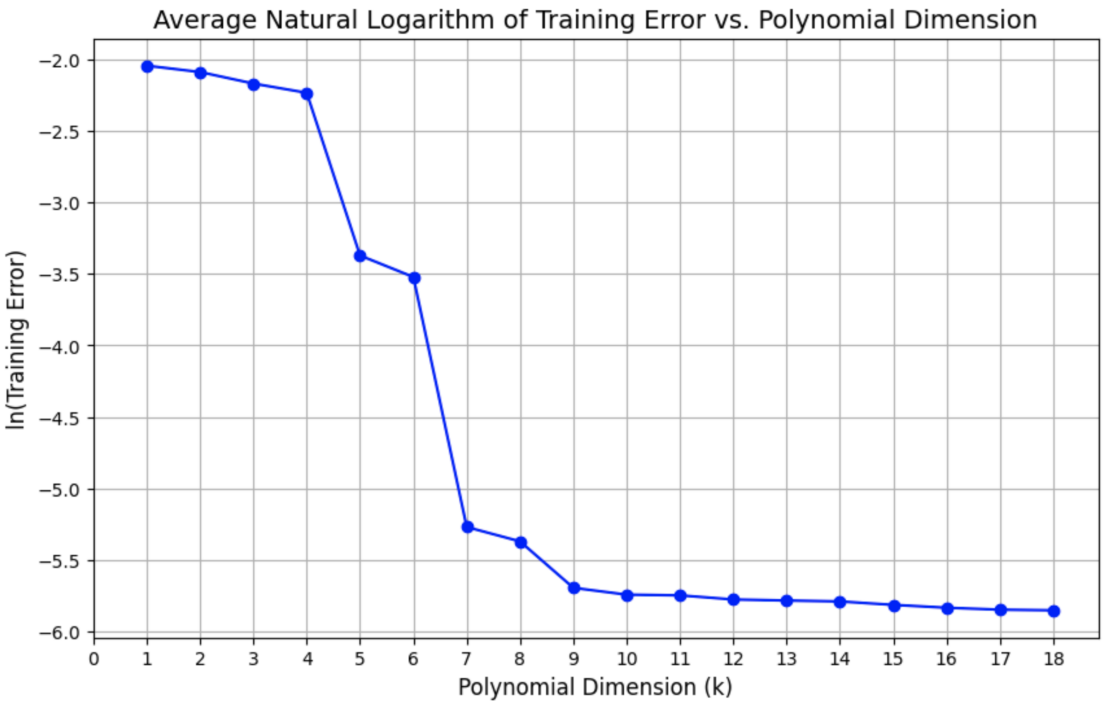

Figure 6: Average natural log of training error for polynomials of degrees {1,...,18} fit to data.

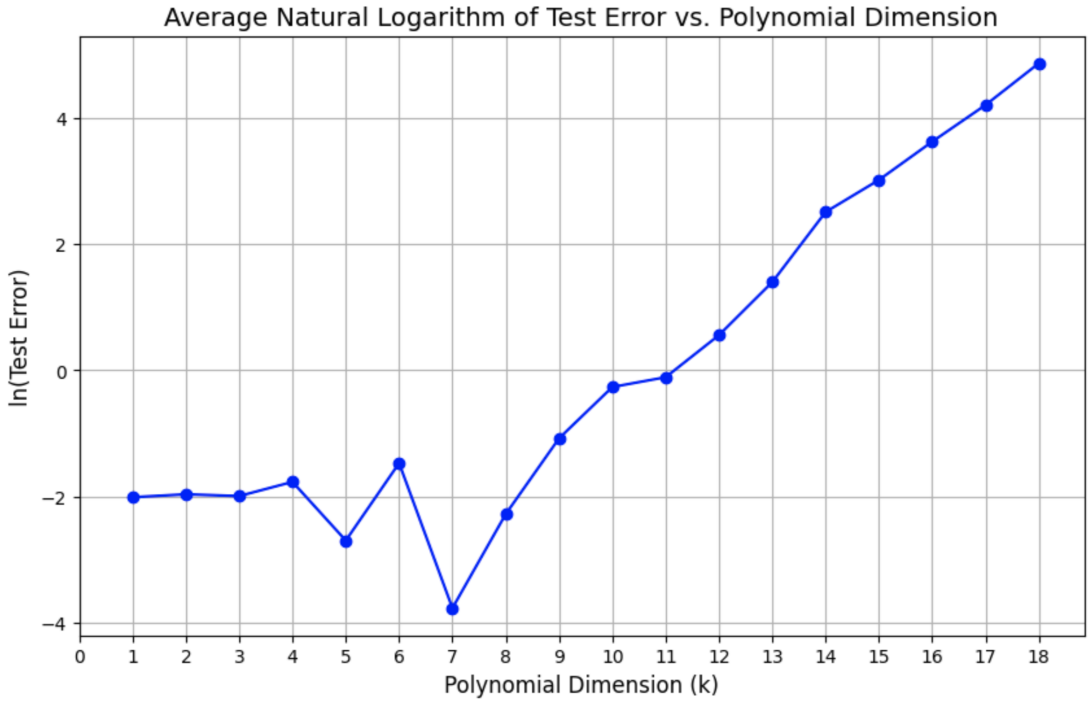

Figure 7: Average natural log of test error for polynomials of degrees {1,...,18} fit to data.

3 Exercise 3

3.a

[Section not repeated]

3.b

We followed the same procedure as described in **2.b** except used a basis of $\{\sin(1\pi\mathbf{x}), \sin(2\pi\mathbf{x}), \ldots, \sin(k\pi\mathbf{x})\}$ as opposed to a polynomial basis.

We still observe decreasing ln(MSE) for training data with increasing degree k.

Functions:

- sin_basis
  - Input: dataset and multiplier
  - Operation: transforms dataset with the basis $\{\sin(1\pi\mathbf{x}), \sin(2\pi\mathbf{x}), \ldots, \sin(k\pi\mathbf{x})\}$
  - Output: transformed dataset

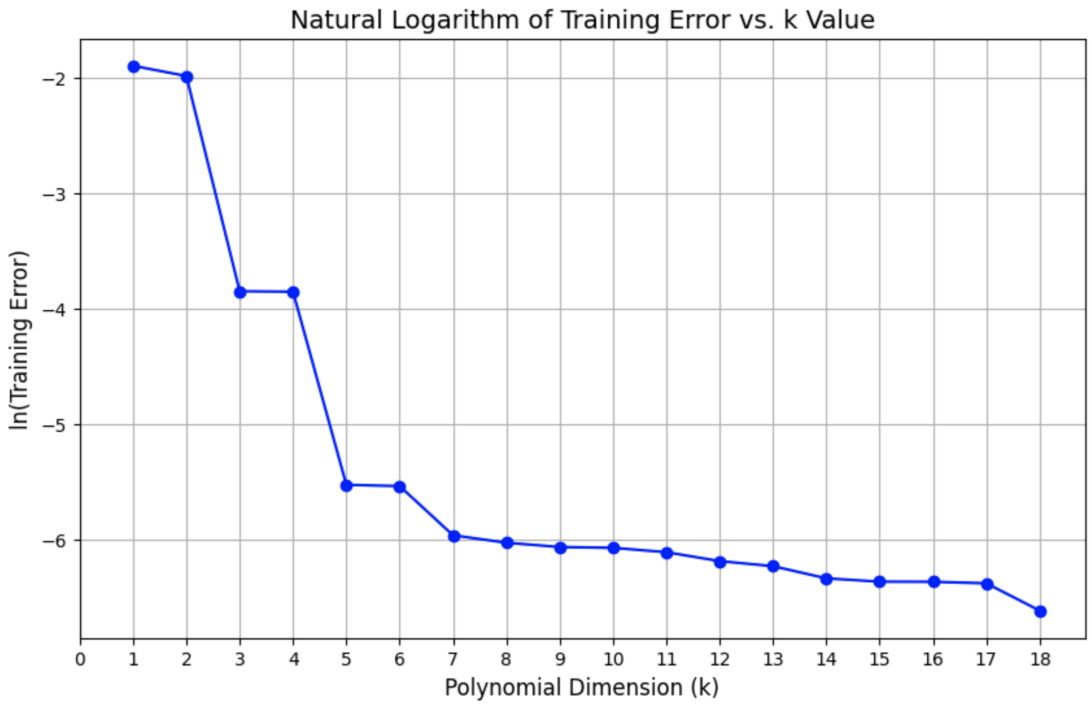

Figure 8: Natural log of training error for polynomials of degrees {1,...,18} fit to data.

3.c

We followed the same procedure as described in **2.c** with the new basis described above.

We now observe decreasing ln(MSE) for test data with increasing degree k until $k = 5$, after which point the ln(MSE) increases slightly, but not as dramatically.

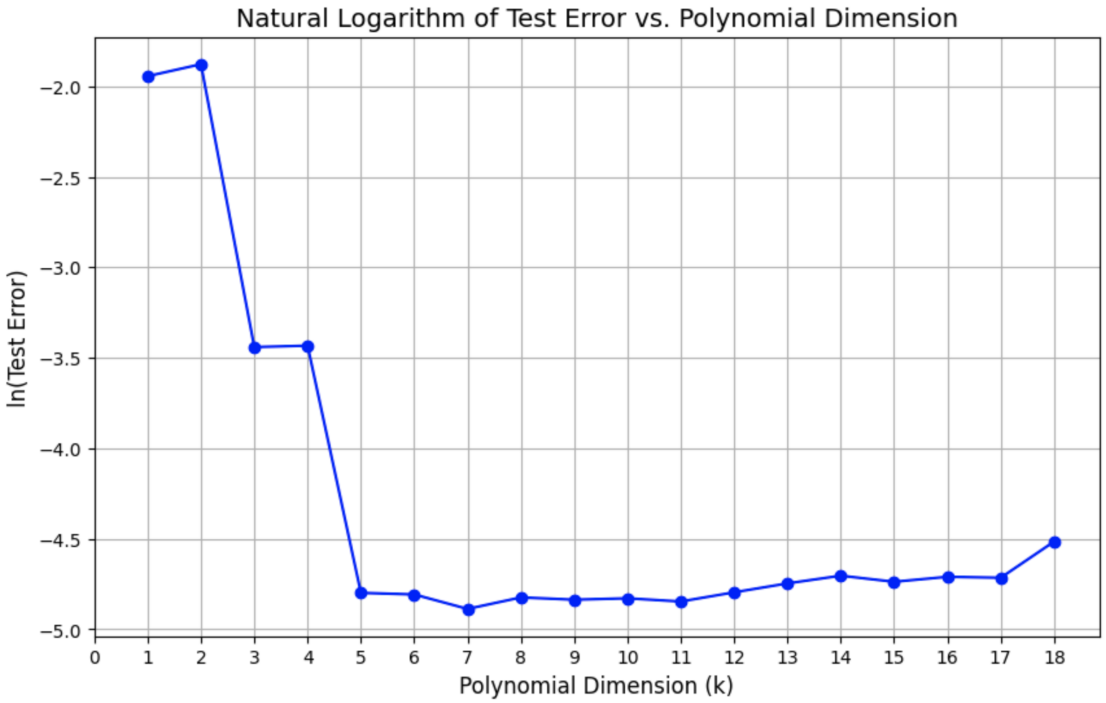

Figure 9: Natural log of test error for polynomials of degrees {1,...,18} fit to data.

3.d

We followed the same procedure as described in **2.d** with the new basis described above.

Our resulting plot for training error decreases similarly to the plot in (b), while our plot for test error shows a more dramatic increase in error after $k = 5$ compared to our plot in (c).

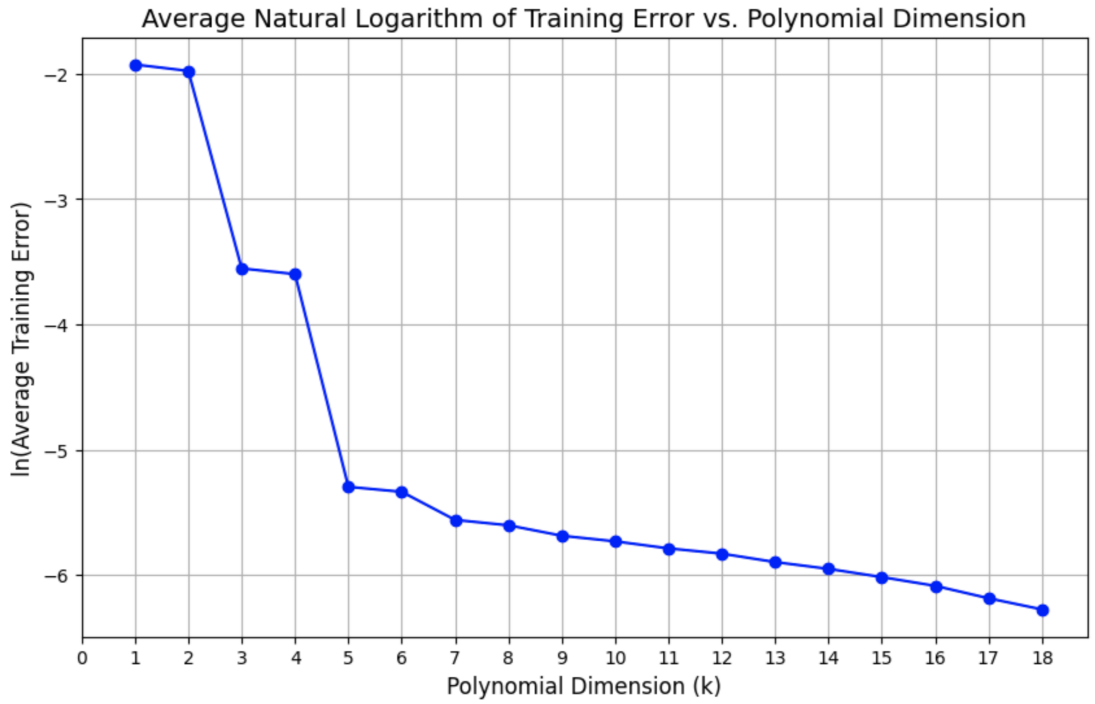

Figure 10: Average natural log of training error for polynomials of degrees {1,...,18} fit to data.

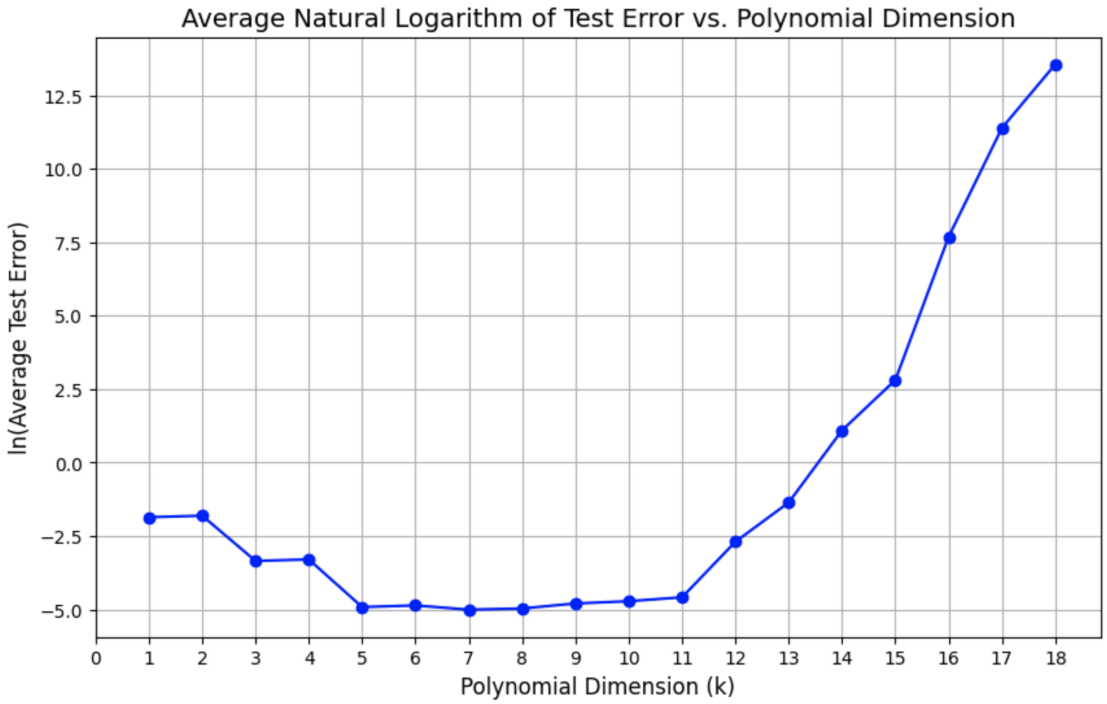

Figure 11: Average natural log of test error for polynomials of degrees {1,...,18} fit to data.

1.2 Filtered Boston housing and kernels

4 Exercise 4

4.a

We fit the data of the target with a constant function of ones as a baseline prediction which doesn't take into account any of the predictor variables. We run the fitting process 20 times and average the resulting MSEs.

After importing our data, we shuffle the examples and select $\frac{2}{3}$ as our training data and $\frac{1}{3}$ as our test data. We replace the training and test sets with vectors of ones of the same lengths. We define our target as the column 'MEDV'. We compute a weight vector based on the training vector of ones and the target.

We then predict our target using our training vector and weights and compute the MSE. We do the same for the test vector. The resulting average MSEs are:

- Training MSE: 84.54
- Test MSE: 84.47

4.b

A constant function provides a constant prediction for the target variable (housing price represented by the 'MEDV' feature). This model will always predicts the same value for the target variable as there is no influence from the predictor variables.

A model with a constant function predicts that the target variable's value does not depend on the input features at all, and its prediction is solely based on this constant value. This allows us to more clearly evaluate the influence of including further features as predictors.

If a model with additional features does not significantly reduce error compared to the constant function, it suggests the model may not be capturing underlying relationships in the data effectively. On the other hand, if model with additional features outperforms the constant function, it indicates that the included features provide predictive power in relation to the target.

4.c

Next, we perform separate linear regressions using each individual feature variable with an added bias term, e.g. $(x_i, 1)$ in order to learn a weight vector $w \in \mathcal{R}^2$. We perform each regression 20 times with a different training and test split and average the MSEs over each run.

After splitting our data we extract a single attribute and augment the training and test feature vectors with a bias term. We define our training and test targets as 'MEDV'.

We compute a weight for each feature using the lstsq function with our training features and target. Then, we predict values for our target using the weights and testing features and compute the MSE between the predicted targets and ground truth targets. We compute the average weight, bias and MSE over 20 runs.

We see an improvement in the MSE for each feature compared to the constant function.

Feature	Weight	Bias	MSE
'CRIM'	-0.42	24.11	71.94
'ZN'	0.14	20.99	73.12
'INDUS'	-0.67	29.96	64.20
'CHAS'	7.02	22.01	82.70
'NOX'	-33.28	41.05	66.16
'RM'	9.10	-34.64	42.12
'AGE'	-0.12	31.05	69.89
'DIS'	1.07	18.44	81.19
'RAD'	-0.40	26.40	70.94
'TAX'	-0.03	33.21	67.57
'PTRATIO'	-2.16	62.40	64.44
'LSTAT'	-0.94	34.34	40.19

Figure 12: Weight, bias and MSE for linear regression with each individual feature.

4.d

Finally, we perform a linear regression using all features as predictors with an incorporated bias term and compute the average training and test MSEs over 20 runs.

We shuffle all the data and split it again into $\frac{2}{3}$ training and $\frac{1}{3}$ test. We create augmented matrices for the predictors with an added bias term of 1. We define our training and test target as 'MEDV' and compute a vector of weights using the training features and target using the lstsq function. We predict the training outcomes using the weights and training targets, and the test outcomes using the weights and test targets. We calculate the training and test MSE for each run, and average the resulting MSEs.

We find that resulting MSEs for all features are indeed lower than that of any singular feature:

- Training MSE: 22.28
- Test MSE: 23.98

1.3 Kernelized ridge regression

5 Exercise 5

5.a

We create a vector of γ values $\left[2^{-40}, 2^{-39}, \ldots, 2^{-26}\right]$ and a vector of σ values $\left[2^{7}, 2^{7.5}, \ldots, 2^{12.5}, 2^{13}\right]$. We then divide our data into training and test sets and hold out the test set for final evaluation.

Next, using the training data we create our $k = 5$ subsets for cross validation where each set has a different $\frac{1}{5}$ held out as test data. We initialize a best MSE to a high value.

We then run ridge regression on each combination of σ and γ values for each k set. In this process, we compute the kernel matrix and alpha vector for each sub-training set. We then predict the y values for the test set using the test features, kernel matrix value and alpha vector value. We calculate the MSE using the predicted y values compared to the ground truth y values.

We average MSEs over each of the 5 k-fold tests and if the resulting average MSE for this combination of γ and σ is an improvement to our current best MSE, we update this value.

Functions:

- k_fold_validation
  - Input: k number, training data to k-fold.
  - Operation: shuffles training data, isolates k test sets within data, creates k copies of data with each test held out and the remaining data as the training set.
  - Output: List of length k, with each entry being one of the k folds of data.
- data_splitter
  - Input: dataset
  - Operation: separates feature and outcome variables
  - Output: feature vector, outcome vector
- gaussian_kernel
  - Input: two x values, sigma value
  - Operation: computes one instance of the Gaussian kernel using the inputs using the equation $\mathrm{K}(\mathbf{x}_i, \mathbf{x}_j) = e^{\frac{-\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}}$
  - Output: Gaussian kernel result
- gaussian_kernel_matrix
  - Input: dataset, sigma value
  - Operation: computes the entire Gaussian kernel matrix for the dataset using the gaussian_kernel function
  - Output: Gaussian kernel matrix
- alpha
  - Input: Gaussian kernel matrix, gamma value, length of data, outcome vector
  - Operation: calculates the alpha value using the equation $\boldsymbol{\alpha}^* = (\mathbf{K} + \gamma\ell\mathbf{I}_\ell)^{-1}\mathbf{y}$
  - Output: alpha value

5.b

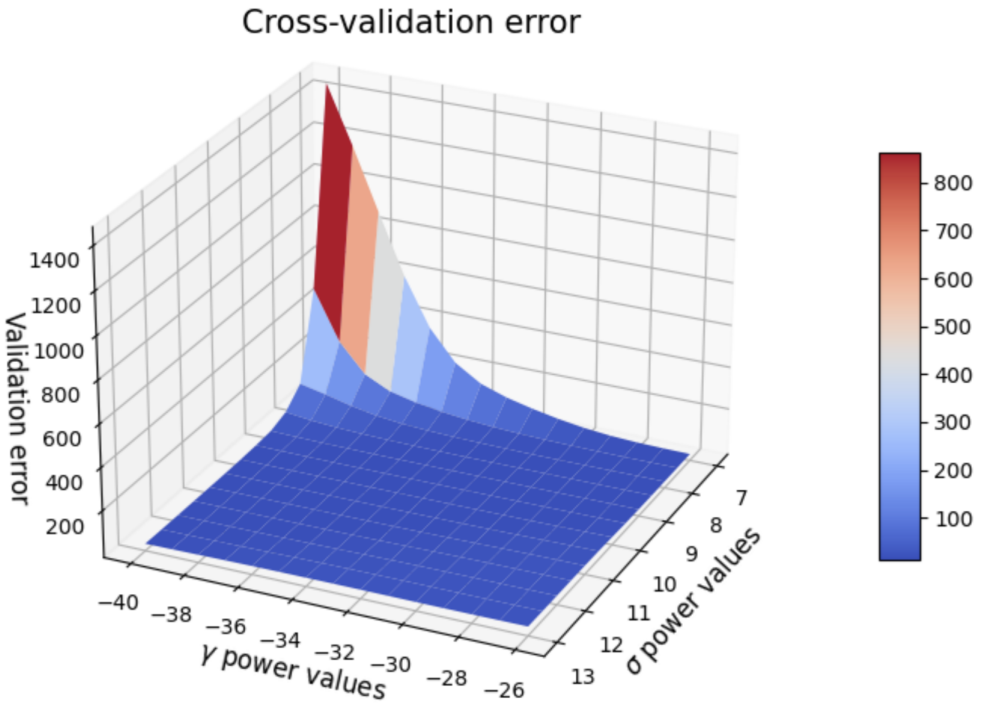

Figure 13: Cross-validation error for each γ-σ pair (shown as power of 2).

5.c

We isolate the best performing γ and σ values from the k-fold test. Using these values we then compute the kernel matrix and alpha vector for our entire training dataset. We then predict our y values for the training set and calculate the MSE. We repeat this process for the testing set.

- Best γ : 1.8189894035458565e-12
- Best σ : 5792.62
- Training MSE : 9.34
- Testing MSE : 18.54

5.d

We acquire a new training test split of our data and repeat the procedures in **5.a** and **5.c** 20 times, averaging the training and test MSEs over all 20 runs and computing the standard deviation of each.

Method	MSE train	MSE test
Naive Regression	84.54 ± 5.39	84.47 ± 10.76
Linear Regression (attribute 1)	72.17 ± 3.96	71.94 ± 3.96
Linear Regression (attribute 2)	73.87 ± 6.02	73.12 ± 6.02
Linear Regression (attribute 3)	65.15 ± 5.31	64.2 ± 5.31
Linear Regression (attribute 4)	81.73 ± 4.91	82.7 ± 4.91
Linear Regression (attribute 5)	70.56 ± 3.77	66.16 ± 3.77
Linear Regression (attribute 6)	44.59 ± 3.23	42.12 ± 3.23
Linear Regression (attribute 7)	73.87 ± 4.15	69.89 ± 4.15
Linear Regression (attribute 8)	78.33 ± 5.54	81.19 ± 5.54
Linear Regression (attribute 9)	72.93 ± 5.56	70.94 ± 5.56
Linear Regression (attribute 10)	65.24 ± 4.37	67.57 ± 4.37
Linear Regression (attribute 11)	61.92 ± 3.49	64.44 ± 3.49
Linear Regression (attribute 12)	37.74 ± 1.9	40.19 ± 1.9
Linear Regression (all attributes)	22.28 ± 1.65	23.98 ± 3.59
Kernel Ridge Regression	6.85 ± 2.24	12.65 ± 2.32

Figure 14: Training and test MSEs with standard deviations for all regression permutations.

2 Part II

2.1 k-Nearest Neighbors

2.1.1 Generating the data

6 Exercise 6

We first generate a visualization of a voted-center hypothesis $h_{S,v} : [0,1]^2 \to \{0,1,\square\}$ where $S = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_{|S|}, y_{|S|})\}$ are labeled centers where $\mathbf{x}_i \in [0,1]^2$, $y_i \in \{0,1\}$, and v as a positive integer representing the number of nearest neighbors considered. We use $v := 3$ and label $h_{S,v}(\mathbf{x}_i) = 0, 1,$ or $\square$ if the v nearest neighbors $\mathbf{x}_i$ are a majority $0, 1$ or tied.

We generate a hypothesis $h_{S,v}$ with $|S| = 100$ and $v = 3$ and determine a classification map for the hypothesis by evaluating the v nearest neighbors for each point on a 100×100 background map where each point $\in [0,1]^2$.

We first sample 100 centers randomly and generate our background points uniformly to cover the map. We then predict labels for each of our background points using the knn function with $v = 3$. We then plot the map with the original centers included.

Functions:

- vector_euclidean
  - Input: centers and data to predict labels for
  - Operation: checks if the data is a single example or multiple and reshapes accordingly. Computes the Euclidean distances between the data point(s) to all centers using the

equation $dist = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

- Output: an array containing the distance from each data point to each center where the first dimension is the number of data points and the second is the number of centers

- knn
  - Input: centers, center labels, data to predict labels for, k value
  - Operation: calculates the distance from each center to each data point, isolates to k-nn for each data point, counts the number of 0 and 1 labels for the k-nn of each data point, generates a list of labels corresponding to each data point based on the majority of k-nn labels
  - Output: list of labels (0 or 1) for each data point based on the k-nn labels for centers

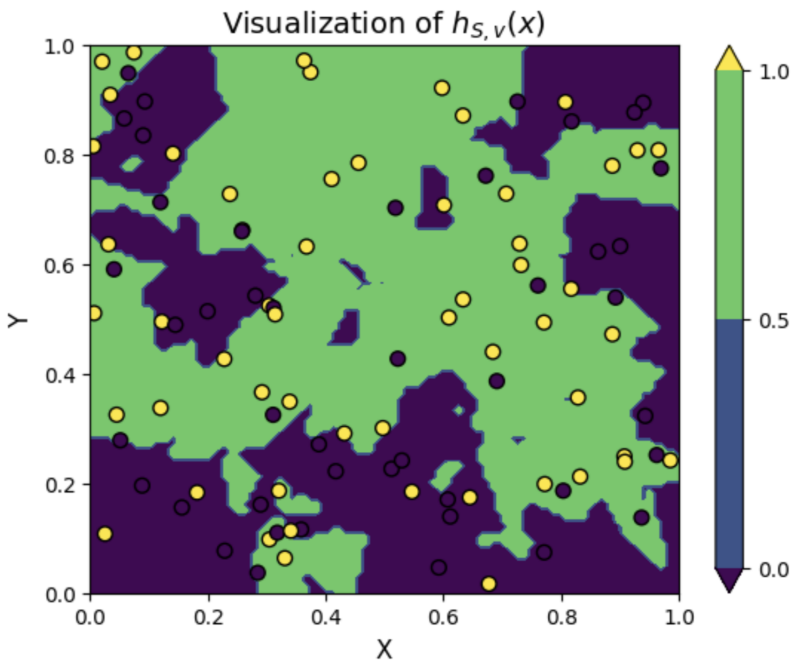

Figure 15: Visualization of $h_{S,v}(x)$ with original centers and background map.

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

7 Exercise 7

7.a

We follow Experimental Protocol A. For each $k \in \{1, \ldots, 49\}$ we complete 100 runs of the following.

We sample a new h from $p_{\mathcal{H}}$ with $|S| = 100$. We next generate 4000 training points by sampling

randomly from $[0, 1]$ in two dimensions and generating the corresponding labels for a randomly selected 80% based on k-nn of the centers with $k = 3$, while the remaining 20% receive a randomly generated label. We then sample 1000 testing points following the same method.

We next predict labels for the test points using our knn function with k set to the current value of k for that iteration. We compare the predicted labels to the ground truth test labels to determine the generalization error for each run and average the result over 100 runs and plot the results.

Functions:

- decision
  - Input: a probability in the range $[0, 1]$ of the data being randomly generated
  - Operation: returns True with the probability specified as input
  - Output: True or False boolean
- generate_data
  - Input: number of data points, randomly generated data points, centers, center labels
  - Operation: determines labels for randomly generated data points based either on k-nn of center labels (with $k = 3$) for 80% of points or a random label from $[0, 1]$ for 20% of points.
  - Output: a vector of all labels for the randomly generated data points

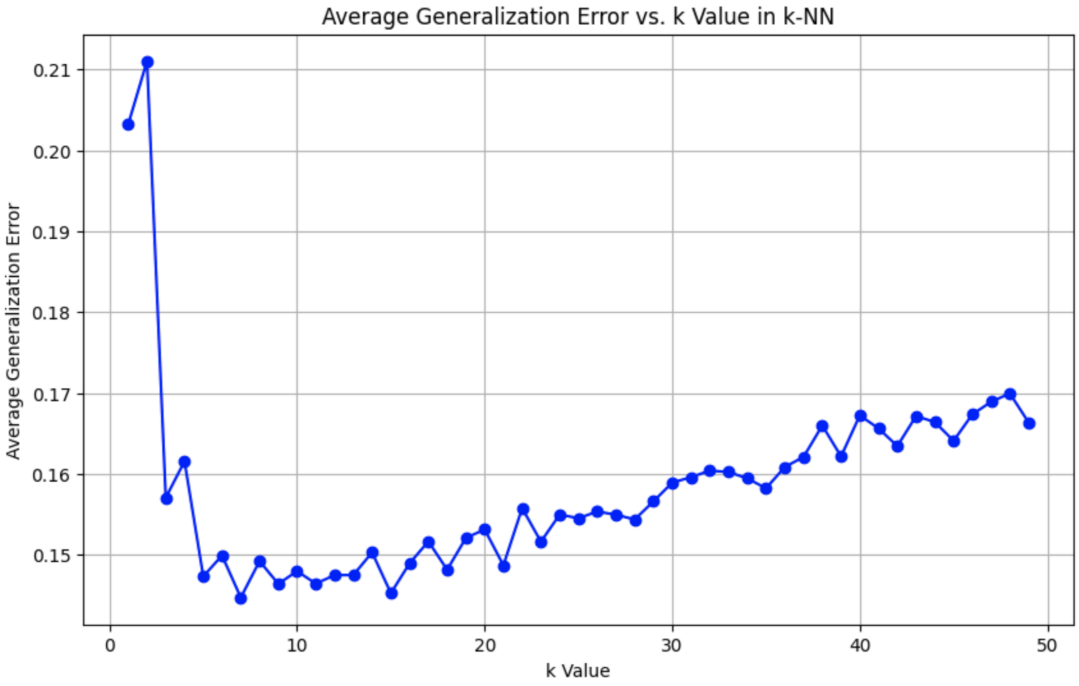

Figure 16: Average generalization error for each k value in {1,...,49}.

7.b

Explanation of figure:

The resulting figure has the lowest average generalization error between k values of 5 and 15, roughly centered around 10. When k is very low (≤ 4) we see an initially high but rapidly decreasing error, due to the fact that with very few neighbors taken into consideration, our labels are not necessarily based on local patterns and could be capturing only the noise generated in the training data (we see an initial error of roughly 0.2, close to the proportion of noise in the training data).

However, as the k value continues to increase (for $5 \leq k \leq 15$) we see the error drop to its lowest points (≤ 0.15), as considering this number of neighbors allows us to capture local patterns sufficiently and hits the correct balance for the bias-variance tradeoff. However, beyond this point, as our k value increases, we see a steady rise in error ($0.15 \leq \text{error} \leq 0.17$) suggesting that our k value is too large to capture local variation and is capturing information over too large an area in the training examples.

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

8 Exercise 8

8.a

We follow Experimental Protocol B. For each $m \in \{100, 500, 1000, 1500, \ldots, 4000\}$ we complete 100 runs of the following.

We iterate over each $k \in \{1, \ldots, 49\}$. We sample a new h from $p_{\mathcal{H}}$ with $|S| = 100$. We next generate m training points by sampling randomly from $[0, 1]$ in two dimensions and generating the corresponding labels for a randomly selected 80% based on k-nn of the centers with $k = 3$, while the remaining 20% receive a randomly generated label. We then sample 1000 testing points following the same method.

We next predict labels for the test points using our knn function with k set to the current value of k for that iteration. We compare the predicted labels to the ground truth test labels to determine the generalization error for each run. We save the k value with the lowest error for each run and average the result over 100 runs.

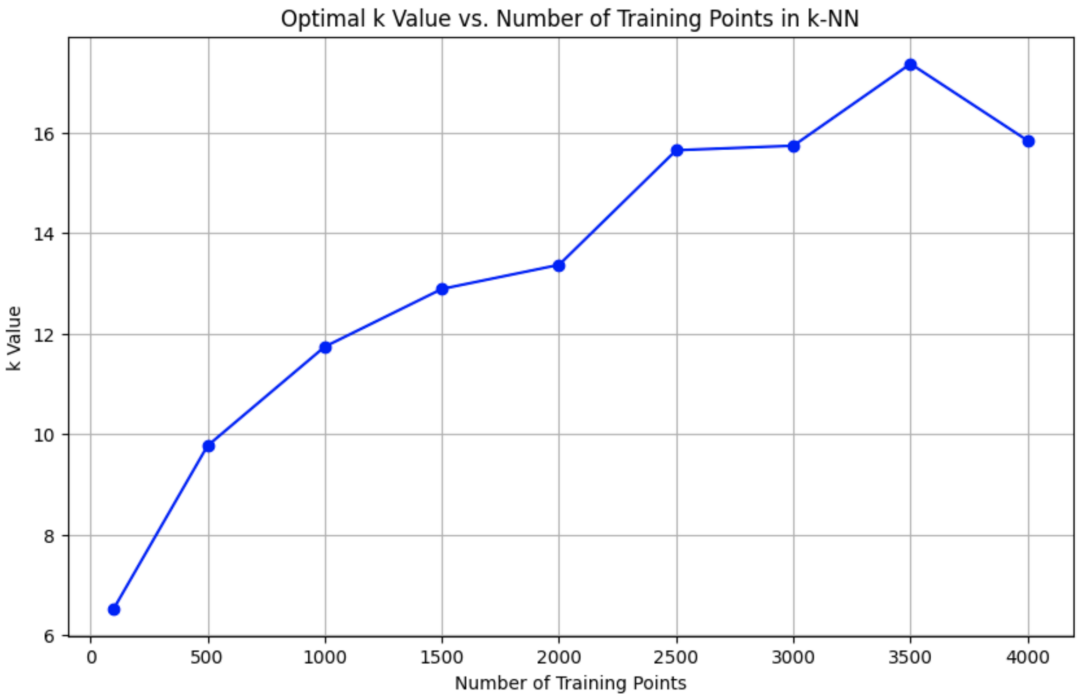

Figure 17: Optimal k value in the range {1,...,49} for number of training points.

8.b

In general, as the number of training points increases the optimal k value also increases, a trend which is more dramatic when the number of training points is lower (≤ 1000) and begins to flatten out afterwards.

The general trend of the optimal k increasing as the number of training points increases indicates that with fewer training points, less neighbors are needed to overcome noise in the data and capture spatial patterns, whereas with more training points (and more points representing noise) the value of the optimal k increases. Additionally with a higher k value with more training points, we do not run into the issue of capturing too large of a space in our consideration as the training points are more concentrated.

The fact that the optimal k value increases more dramatically for a lower number of data points is likely due to the fact that as the training data becomes increasingly concentrated, the value of adding more neighbors decreases.

3 Part III

3.1 Questions

9 Exercise 9

9.a

We show that for the kernel

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

where $\mathbf{x}, \mathbf{z} \in \mathbb{R}^n$, the kernel K_c is positive semidefinite when $c \geq 0$.

Proof:

A kernel K is positive semidefinite for a feature map $\phi(\mathbf{x})\mathbb{R}^n \rightarrow W$, where $W \in \mathcal{H}$ if and only if

$$K(\mathbf{x}, \mathbf{t}) = \langle \phi(\mathbf{x}),\ \phi(\mathbf{t}) \rangle\ . \tag{2}$$

Therefore, we must find a feature map $\phi_c : \mathbb{R}^n \rightarrow \mathbb{R}^{n+1}$, where $\mathbb{R}^{n+1} \in \mathcal{H}$. We show that the following feature map works to satisfy this constraint:

$$\phi_c(\mathbf{x}) = (\sqrt{c}, x_1, x_2, ..., x_n). \tag{3}$$

To do this, we show that using the feature map $\phi_c\,(\mathbf{x})$, the inner product of our kernel $K_c\,(\mathbf{x}, \mathbf{z})$ satisfies the properties of conjugate symmetry, linearity of first argument and positive semidefiniteness.

1. Conjugate symmetry implies that for $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{R}$ we must have

   $$\langle \phi_c(\mathbf{x}),\ \phi_c(\mathbf{y}) \rangle\ = \langle \overline{\phi_c(\mathbf{y}),\ \phi_c(\mathbf{x})} \rangle\ . \tag{4}$$

   We know our kernel is symmetric as its dot product is symmetric. Since we also have $\mathbf{x}, \mathbf{z} \in \mathbb{R}^n$, $c \in \mathbb{R}$ and $\phi_c : \mathbb{R}^n \rightarrow \mathbb{R}^{n+1}$, this becomes trivial.

2. Linearity in first argument implies that for some scalars a, b

   $$\langle a\phi_c(\mathbf{x}) + b\phi_c(\mathbf{y}),\ \phi_c(\mathbf{z}) \rangle\ = a\langle \phi_c(\mathbf{x}),\ \phi_c(\mathbf{y}) \rangle\ + b\langle \phi_c(\mathbf{x}),\ \phi_c(\mathbf{z}) \rangle\ . \tag{5}$$

   Using our feature map we can show this equality:

   $$\begin{aligned}
   \langle a\phi_c(\mathbf{x}) + b\phi_c(\mathbf{y}),\ \phi_c(\mathbf{z}) \rangle\ &= \langle a(\sqrt{c}, x_1, ..., x_n) + b(\sqrt{c}, y_1, ..., y_n),\ (\sqrt{c}, z_1, ..., z_n) \rangle & (6)\\
   &= \langle ((a+b)\sqrt{c}, ax_1 + by_1, ..., ax_n + by_n),\ (\sqrt{c}, z_1, ..., z_n) \rangle & (7)\\
   &= (ac + ax_1z_1 + ...ax_nz_n + bc + by_1z_1 + ... + by_nz_n) & (8)\\
   &= a\langle \phi_c(\mathbf{x}),\ \phi_c(\mathbf{y}) \rangle\ + b\langle \phi_c(\mathbf{x}), \phi_c(\mathbf{z}) \rangle & (9)
   \end{aligned}$$

3. Positive semi-definiteness implies that for our kernel we must have

   $$\langle \phi_c(\mathbf{x})\ ,\ \phi_c(\mathbf{x}) \rangle\ \geq 0. \tag{10}$$

   In our case, this means we must have

   $$\langle \phi_c(\mathbf{x})\ ,\ \phi_c(\mathbf{x}) \rangle\ = \langle (\sqrt{c}, x_1, ..., x_n)\ ,\ (\sqrt{c}, x_1, ..., x_n) \rangle\ = c + \sum_{i=1}^{n} x_i x_i \geq 0 \tag{11}$$

   For the above to hold, we must have $c \geq -(\sum_{i=1}^{n} x_i x_i)$.

We further constrain our results by checking for the positive semidefiniteness when $\mathbf{x} = \mathbf{z} = \vec{0}$. This gives us

$$K_c(\vec{0}, \vec{0}) = c. \tag{12}$$

A kernel must be positive semidefinite for any combination of vectors $\mathbf{x}, \mathbf{z} \in \mathbb{R}^n$. Thus, it must also be positive semi-definite when $\mathbf{x} = \mathbf{z} = \vec{0}$.

Therefore, our kernel K_c is positive semidefinite if and only if $c \geq 0$. This constraint complies with the the properties of a positive semidefinite kernel above.

We have shown that our kernel

$$K_c(\mathbf{x}, \mathbf{z}) = c + \sum_{i=1}^{n} x_i z_i \tag{13}$$

is positive semidefinite if and only if $c \geq 0$ for $\vec{x}, \vec{z} \in \mathbb{R}^n$.

9.b

We substitute our kernel K_c into the dual optimization formulation after kernelization for alpha with $\gamma = 0$ to obtain

$$\boldsymbol{\alpha}^* = \text{argmin}_{\boldsymbol{\alpha} \in \mathbb{R}^{\boldsymbol{m}}} \frac{1}{m} \sum_{i=1}^{m} \left(\sum_{j=1}^{m} \alpha_j c + \sum_{j=1}^{m} \alpha_j \mathbf{x}_i^T \mathbf{x}_j - y_i \right)^2 \tag{14}$$

$$= \text{argmin}_{\boldsymbol{\alpha} \in \mathbb{R}^{\boldsymbol{m}}} \frac{1}{m} \sum_{i=1}^{m} \left(\left(\sum_{j=1}^{m} \alpha_j c \right)^2 + \left(\sum_{j=1}^{m} \alpha_j \mathbf{x}_i^T \mathbf{x}_j - y_i \right)^2 + 2 \left(\sum_{j=1}^{m} \alpha_j c \sum_{j=1}^{m} \alpha_j \mathbf{x}_i^T \mathbf{x}_j - y_i \right) \right) \tag{15}$$

From this rearrangement we re-write

$$\mathbf{x}_i^T \mathbf{x}_j = K_{i,j} = \sum_{q=1}^{n} x_{iq} x_{jq} \tag{16}$$

which is a kernel that computes the dot product of two vectors $\mathbf{x}_i, \mathbf{x}_j \in \mathbb{R}^n$. Since $\sum_{j=1}^{m} \alpha_j c$ has no i dependency, we can rearrange the expression further to obtain

$$\boldsymbol{\alpha}^* = \arg \min_{\boldsymbol{\alpha} \in \mathbb{R}^{\boldsymbol{m}}} \left(\frac{1}{m} \sum_{i=1}^{m} \left(\sum_{j=1}^{m} \alpha_j K_{i,j} - y_i \right)^2 + c^2 \left(\sum_{j=1}^{m} \alpha_j \right)^2 + 2c \left(\sum_{j=1}^{m} \alpha_j \right) \frac{1}{m} \sum_{i=1}^{m} \left(\sum_{j=1}^{m} \alpha_j K_{i,j} - y_i \right) \right). \tag{17}$$

We have acquired the original dual optimization formulation with $\gamma = 0$ and with an additional two terms both dependant on c and $\boldsymbol{\alpha}$.

Thus, one can interpret the kernel K_c to add a regularisation term, c, to $\boldsymbol{\alpha}^*$ in a linear regression optimisation problem.

We study two extreme cases. When $c = 0$, we recover the original optimisation problem, in

which we apply the kernel trick to linear regression by defining our kernel as the dot product between two vectors

$$K_{c=0}(\mathbf{x}, \mathbf{z}) = \sum_{i=1}^{n} x_i z_i \tag{18}$$

Thus when $c = 0$ we would be applying no regulariser at all.

On the other hand, as c increases, the regularisation term becomes more pronounced, restricting the size of the solution space for our problem. In the extreme case where c is very large, we obtain $\boldsymbol{\alpha} \approx 0$, thus always yielding $y_{test} \approx 0$.

10 Exercise 10

A classifier $\text{sign}\left(f\left(\mathbf{t}\right)\right)$ where

$$f\left(\mathbf{t}\right) = \sum_{i=1}^{m} \alpha_i K_\beta\left(\mathbf{x}_i, \mathbf{t}\right) \tag{19}$$

trained on a dataset $\left(\mathbf{x}_1, y_1\right), \ldots, \left(\mathbf{x}_m, y_m\right) \in \mathbb{R}^n \times \{-1, 1\}$ with a Gaussian kernel

$$K_\beta = \exp\left(-\beta \|\mathbf{x} - \mathbf{t}\|^2\right) \tag{20}$$

simulates a 1-Nearest Neighbor Classifier when β is selected as a large enough positive value such that

$$|\alpha^* K_\beta(\mathbf{x}^*, \mathbf{t})| > |\sum_{i=1}^{m-1} \alpha_i K_\beta(\mathbf{x}_i, \mathbf{t})| \tag{21}$$

where $\mathbf{x}^*$ is the nearest neighbor of $\mathbf{t}$ and $\mathbf{x}_{1:m-1}$ are all other remaining neighbors.

Proof:

For kernelized ordinary least squares regression $\boldsymbol{\alpha} = \mathbf{K}^{-1}\mathbf{y}$ where $\mathbf{K}$ is the kernel matrix

$$\mathbf{K} = \begin{pmatrix} K_\beta(\mathbf{x}_1, \mathbf{x}_1) & \cdots & K_\beta(\mathbf{x}_1, \mathbf{x}_m) \\ \vdots & \ddots & \vdots \\ K_\beta(\mathbf{x}_m, \mathbf{x}_1) & \cdots & K_\beta(\mathbf{x}_m, \mathbf{x}_m) \end{pmatrix} \tag{22}$$

with $\mathbf{x} \in \mathbb{R}^m$. When $\beta \gg 0$, elements of the kernel matrix will evaluate to 1 when the input $\mathbf{x}$ values are equal and close to 0 when they are not, as in the first case the kernel becomes

$$K_\beta = \exp\left(-\beta \|\mathbf{x} - \mathbf{x}\|^2\right) = \exp\left(0\right) = 1 \tag{23}$$

and in the second the kernel becomes

$$K_\beta = \exp\left(-\beta \|\mathbf{x'} - \mathbf{x}\|^2\right) = \exp\left(-\beta d\right) \approx 0 \tag{24}$$

where d is the square of the norm of $\mathbf{x'}$ and $\mathbf{x}$.

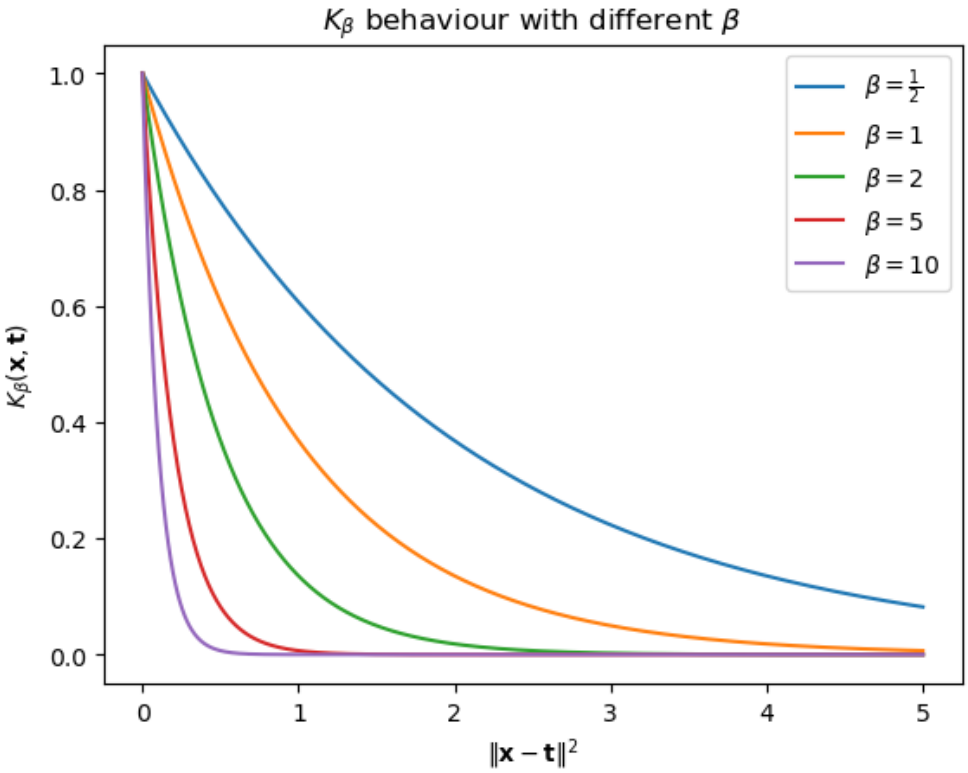

Figure 18: Behavior of the kernel $K_\beta(\mathbf{x}, \mathbf{t})$ with increasing values of β.

Therefore, with $\beta \gg 0$, the kernel matrix becomes

$$\mathbf{K} \approx \begin{pmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{pmatrix} \tag{25}$$

or $\mathbf{K} \approx \mathbf{I}$ where $\mathbf{I}$ is the identity matrix of dimension m. Thus,

$$\mathbf{K} \approx \mathbf{I} = \mathbf{I}^{-1} \approx \mathbf{K}^{-1} \tag{26}$$

and we can simplify $\boldsymbol{\alpha}$ as

$$\boldsymbol{\alpha} = \mathbf{K}^{-1}\mathbf{y} \approx \mathbf{K}\mathbf{y} \approx \mathbf{I}\mathbf{y} = \mathbf{y}. \tag{27}$$

In such a case, there exists a β large enough to make the Gaussian kernel be bigger as the distance from $\mathbf{x}_i$ to $\mathbf{t}$ is smaller. Therefore, when $\beta \gg 0$ the elements in the sum of our function $f(\mathbf{t})$ become

$$\alpha_i K_\beta(\mathbf{x}_i, \mathbf{t}) \approx y_i K_\beta(\mathbf{x}_i, \mathbf{t}) \tag{28}$$

and the classifier $\text{sign}(f(\mathbf{t}))$ for a single element in the sum becomes

$$\text{sign}(\alpha_i K_\beta(\mathbf{x}_i, \mathbf{t})) \approx \text{sign}(y_i K_\beta(\mathbf{x}_i, \mathbf{t})) = \text{sign}(y_i) = -1 \tag{29}$$

when $y_i = -1$ and

$$\text{sign}(\alpha_i K_\beta(\mathbf{x}_i, \mathbf{t})) \approx \text{sign}(y_i K_\beta(\mathbf{x}_i, \mathbf{t})) = \text{sign}(y_i) = 1 \tag{30}$$

when $y_i = 1$. Therefore, our classifier returns the value of the label y_i of $\mathbf{x}_i$. This shows that our classifier will simulate a 1-Nearest Neighbor Classifier when:

$$|\alpha^* K_\beta(\mathbf{x}^*, \mathbf{t})| > |\sum_{i=1}^{m-1} \alpha_i K_\beta(\mathbf{x}_i, \mathbf{t})| \tag{31}$$

Imposing this restriction, when we rewrite our function as

$$f(\mathbf{t}) = y^* K_\beta(\mathbf{x}^*, \mathbf{t}) + \sum_{i=1}^{m-1} y_i K_\beta(\mathbf{x}_i, \mathbf{t}) \tag{32}$$

we can see that $f(\mathbf{t})$ evaluates to a value in the range $(0, 2]$ when $y^* = 1$ or a value in the range $[-2, 0)$ when $y^* = -1$. Therefore, our classifier $\text{sign}(f(\mathbf{t}))$ will return the label from $\{-1, 1\}$ belonging to the nearest neighbor $\mathbf{x}^*$ of $\mathbf{t}$, simulating a 1-Nearest Neighbor Classifier.

11 Exercise 11

11.a

We show that

$$\mathcal{E}_{\rho_i}(f_i) = \inf_{f:\mathcal{X}\to\mathcal{Y}} \mathcal{E}_{\rho_i}(f) = 0. \tag{33}$$

We have $\mathcal{C} \subset \mathcal{X}$ as a subset of $\mathcal{X}$ (which has infinite cardinality) with cardinality $|\mathcal{C}| = 2n$. Using this, we can change the original integral of $\mathcal{E}_{\rho_i}(f)$ to a sum using the discrete property of $\mathcal{C}$:

$$\mathcal{E}_{\rho_i}(f) = \int_{\mathcal{X}\times\mathcal{Y}} \mathbf{1}_{\{f(x)\neq y\}} d\rho_i\{x, y\} = \sum_{\mathcal{C}\times\mathcal{Y}} \frac{1}{2n} \mathbf{1}_{\{f(x)\neq y\}} \mathbf{1}_{\{f_i(x)=y\}} \tag{34}$$

When $f_i(x) \neq y$, $\rho_i\{x, y\} = 0$ meaning that $\vec{1}_{\{f_i(x)=y\}}$ will always evaluate to 0 in such cases.

In order to minimise $\mathcal{E}_{\rho_i}(f)$, we set $f(x) = f_i(x)$. In this case, if $f(x) = f_i(x) = y$, then $\mathcal{E}_{\rho_i}(f) = 0$ as $\mathbf{1}_{\{f(x)\neq y\}} = 0$. However, if $f(x) = f_i(x) \neq y$, then $\mathcal{E}_{\rho_i}(f) = 0$ as $\mathbf{1}_{\{f_i(x)=y\}} = 0$.

Therefore, by setting $f(x) = f_i(x)$ we acquire

$$\mathcal{E}_{\rho_i}(f_i) = \inf_{f:\mathcal{X}\to\mathcal{Y}} \mathcal{E}_{\rho_i}(f) = 0. \tag{35}$$

11.b

We show that

$$\max_{i=1,\ldots,T} \mathbb{E}_{S\sim\rho_i^n}[\mathcal{E}_{\rho_i}(A(S))] \geq \min_{j=1,\ldots,k} \frac{1}{T} \sum_{i=1}^{T} \mathcal{E}_{\rho_i}(A(S_j^i)). \tag{36}$$

We begin by showing that for any $i = 1, \ldots, T$

$$\mathbb{E}_{S\sim\rho_i^n}[\mathcal{E}_{\rho_i}(A(S))] = \frac{1}{k} \sum_{j=1}^{k} \mathcal{E}_{\rho_i}(A(S_j^i)). \tag{37}$$

By studying $\mathcal{E}_{\rho_i}(A(S))$ we obtain

$$\mathcal{E}_{\rho_i}(A(S)) = \int_{\mathcal{X}\times\mathcal{Y}} \mathbf{1}_{\{f_{[i]}((S_{[j]})_{[n]})\neq y\}} d\rho_i\,(x,y) = \sum_{\mathcal{C}\times\mathcal{Y}} \mathbf{1}_{\{f_i((S_{[j]})_{[n]})\neq y\}} \mathbf{1}_{\{f_i(x)=y\}} \frac{1}{2n} \tag{38}$$

Where $[n] = \{1, ..., n\}, [j] = \{1, ..., k\}$ and $[i] = \{1, ..., T\}$. Thus, the distribution ρ_i constrains the possible training sets that A can receive from S to be only those where $S_{[k]}^{[T]=i}$ equals the i in the subscript of ρ_i: $(S_1^i, ..., S_k^i)$.

Since all k training sets have an equal probability of being sampled, the expectation becomes the average over all $j = 1, ..., k$ training sets. Therefore, we can rewrite the expectation as

$$\mathbb{E}_{S\sim\rho_i^n}[\mathcal{E}_{\rho_i}(A(S))] = \frac{1}{k}\sum_{j=1}^{k} \mathcal{E}_{\rho_i}(A(S_j^i)). \tag{39}$$

Additionally, for any set of scalars $\alpha_i, \ldots, \alpha_m$ we have $\max_\ell \alpha_\ell \geq \frac{1}{m}\sum_{\ell=1}^{m} \alpha_\ell \geq \min_\ell \alpha_\ell$. Using this we can show that

$$\begin{aligned}
\max_{i=1,\ldots,T} \mathbb{E}_{S\sim\rho_i^n}[\mathcal{E}_{\rho_i}(A(S))] &= \max_{i=1,\ldots,T} \sum_{j=1}^{k} \mathcal{E}_{\rho_i}(A(S_j^i)) && (40)\\
&\geq \frac{1}{T}\sum_{i=1}^{T} \frac{1}{k}\sum_{j=1}^{k} \mathcal{E}_{\rho_i}(A(S_j^i)) && (41)\\
&= \frac{1}{k}\sum_{j=1}^{k} \frac{1}{T}\sum_{i=1}^{T} \mathcal{E}_{\rho_i}(A(S_j^i)) && (42)\\
&\geq \min_{j=1,\ldots,k} \frac{1}{T}\sum_{i=1}^{T} \mathcal{E}_{\rho_i}(A(S_j^i)) && (43)
\end{aligned}$$

Therefore, we have shown that

$$\max_{i=1,\ldots,T} \mathbb{E}_{S\sim\rho_i^n}[\mathcal{E}_{\rho_i}(A(S))] \geq \min_{j=1,\ldots,k} \frac{1}{T}\sum_{i=1}^{T} \mathcal{E}_{\rho_i}(A(S_j^i)). \tag{44}$$

11.c

We show that

$$\frac{1}{T}\sum_{i=1}^{T} \mathcal{E}_{\rho_i}(A(S_j^i)) \geq \frac{1}{2} \min_{r=1,\ldots,p} \frac{1}{T}\sum_{i=1}^{T} \vec{1}_{\{A(S_j^i)(v_r)\neq f_i(v_r)\}} \tag{45}$$

when we have fixed $j \in \{1, \ldots, k\}$ with $S_j = (x_i, \ldots, x_n)$ and $S_j' = \{v_1, \ldots, v_p\}$ and C as the subset of points not belonging to S_j.

With $p \geq n$, every possible function $f : \mathcal{C} \to \{0, 1\}$. For every function f_i for $i = 1, ..., T$,

we then have

$$\mathcal{E}_{\rho_i}(f) = \frac{1}{2n}\sum_{\mathcal{C}} \vec{1}_{\{f(x)\neq f_i(x)\}} \tag{46}$$

$$\geq \frac{1}{2n}\sum_{r=1}^{p} \vec{1}_{\{f(v_r)\neq f_i(v_r)\}} \tag{47}$$

$$\geq \frac{1}{2p}\sum_{r=1}^{p} \vec{1}_{\{f(v_r)\neq f_i(v_r)\}}. \tag{48}$$

Given that

$$\frac{1}{m}\sum_{\ell=1}^{m} \alpha_m \geq \min_m \alpha_m \tag{49}$$

we can derive

$$\frac{1}{T}\sum_{i=1}^{T} \mathcal{E}_{\rho_i}(A(S_j^i)) \geq \frac{1}{T}\sum_{i=1}^{T}\frac{1}{2p}\sum_{r=1}^{p} \vec{1}_{\{A(S_j^i)(v_r)\neq f_i(v_r)\}} \tag{50}$$

$$= \frac{1}{2p}\sum_{r=1}^{p}\frac{1}{T}\sum_{i=1}^{T} \vec{1}_{\{A(S_j^i)(v_r)\neq f_i(v_r)\}} \tag{51}$$

$$\geq \frac{1}{2}\min_{r=1,...,p}\frac{1}{T}\sum_{i=1}^{T} \vec{1}_{\{A(S_j^i)(v_r)\neq f_i(v_r)\}}. \tag{52}$$

Therefore, we have shown that

$$\frac{1}{T}\sum_{i=1}^{T} \mathcal{E}_{\rho_i}(A(S_j^i)) \geq \frac{1}{2}\min_{r=1,...,p}\frac{1}{T}\sum_{i=1}^{T} \vec{1}_{\{A(S_j^i)(v_r)\neq f_i(v_r)\}}. \tag{53}$$

11.d

We show that for any $r = 1, ..., p$ it holds that

$$\frac{1}{T}\sum_{i=1}^{T} \mathbf{1}_{\{A\left(S_j^i\right)(v_r)\neq f_i(v_r)\}} = \frac{1}{2}. \tag{54}$$

For any $r = 1, ..., p$ we can partition $\mathcal{Y}^C$ into $\frac{T}{2}$ disjoint pairs where for any pair (f_i, f_i') we have that for each $c \in C$, $f_i(c) \neq f_i'(c)$ if and only if $c = v_r$. In other words, (f_i, f_i') will always produce identical outputs for elements $c \in C$, and always produce complementary outputs for elements v_r. For such a pair, it must be true that $S_j^i = S_j^{i'}$, i.e. that the input-output sets are identical for all $c \in C$. Therefore it follows that

$$\mathbf{1}_{\{A\left(S_j^i\right)(v_r)\neq f_i(v_r)\}} + \mathbf{1}_{\{A\left(S_j^{i'}\right)(v_r)\neq f_i'(v_r)\}} = 1 \tag{55}$$

as when $c = v_r$ when have

$$A\left(S_j^i\right)(v_r) = A\left(S_j^{i'}\right)(v_r) \tag{56}$$

and with

$$f_i(v_r) \neq f_i'(v_r) \tag{57}$$

whenever

$$A\left(S_j^i\right)(v_r) = f_i(v_r) \tag{58}$$

it must be true that

$$A\left(S_j^{i'}\right)(v_r) \neq f_i'(v_r). \tag{59}$$

In this way, whenever one term evaluates to 1, the other must evaluate to 0. This yields

$$\frac{1}{T}\sum_{i=1}^{T} \mathbf{1}_{\{A\left(S_j^i\right)(v_r) \neq f_i(v_r)\}} = \frac{1}{2}. \tag{60}$$

as with the entirely of $\mathcal{Y}^C$ partitioned into disjoint pairs (f_i, f_i'), for every term which evaluates to 1, another will evaluate to 0. In total, half of the terms will evaluate to 1 and half to 0. When divided by the number of functions, T, we obtain $\frac{1}{2}$ as the result.

11.e

Markov's inequality states that for a non-negative random variable Z

$$\forall a \geq 0, P[Z \geq a] \leq \frac{\mathbb{E}[Z]}{a}. \tag{61}$$

With Z taking values in $[0,1]$ and $\mathbb{E}[Z] = \mu$, for any $a \in (0,1)$ we show that

$$P[Z > 1-a] \geq \frac{\mu - (1-a)}{a}. \tag{62}$$

Let Y be a non-negative random variable $Y = 1 - Z$ with $\mathbb{E}[Y] = 1 - \mathbb{E}[Z] = 1 - \mu$. Applying Markov's inequality to Y we obtain

$$P[Z \geq a] = P[Z \leq 1-a] = P[1-Z \geq a] = P[Y \geq a] \leq \frac{\mathbb{E}[Y]}{a} = \frac{1-\mu}{a}. \tag{63}$$

Therefore,

$$P[Z > 1-a] \geq 1 - \frac{1-\mu}{a} = \frac{a+\mu-1}{a} = \frac{\mu-(1-a)}{a}. \tag{64}$$

11.f

Our results in (d) show that

$$\frac{1}{T}\sum_{i=1}^{T} \mathbf{1}_{\{A\left(S_j^i\right)(v_r) \neq f_i(v_r)\}} = \frac{1}{2} \tag{65}$$

and our results in (c) show that

$$\frac{1}{T}\sum_{i=1}^{T} \mathcal{E}_{\rho_i}(A(S_j^i)) \geq \frac{1}{2} \min_{r=1,\ldots,p} \frac{1}{T}\sum_{i=1}^{T} \mathbf{1}_{\{A(S_j^i)(v_r) \neq f_i(v_r)\}} \tag{66}$$

which we can combine to show that

$$\frac{1}{T}\sum_{i=1}^{T} \mathcal{E}_{\rho_i}(A(S_j^i) \geq \frac{1}{2} \min_{r=1,\ldots,p} \frac{1}{2} = \frac{1}{4}. \tag{67}$$

This implies that

$$\min_{j=1,\ldots,k} \frac{1}{T} \sum_{i=1}^{T} \mathcal{E}_{\rho_i}(A(S_j^i) = \frac{1}{4}. \tag{68}$$

Our results in (b) show that

$$\max_{i=1,\ldots,T} \mathbb{E}_{S\sim\rho_i^n}[\mathcal{E}_{\rho_i}(A(S))] \geq \min_{j=1,\ldots,k} \frac{1}{T} \sum_{i=1}^{T} \mathcal{E}_{\rho_i}(A(S_j^i)) \tag{69}$$

which, when combined with the result above yields

$$\max_{i=1,\ldots,T} \mathbb{E}_{S\sim\rho_i^n}[\mathcal{E}_{\rho_i}(A(S))] \geq \frac{1}{4}. \tag{70}$$

This means that for every algorithm A that receives a training set of n examples from $\mathcal{X}$ there exists a function $f : \mathcal{X} \to \mathcal{Y}$ and a distribution ρ over $\mathcal{X}$ such that $\mathcal{E}_\rho = 0$ and

$$\mathbb{E}_{S\sim\rho_i^n}[\mathcal{E}_{\rho_i}(A(S))] \geq \frac{1}{4} \tag{71}$$

The inequality proven in (e) shows that

$$P\left[Z > 1 - a\right] \geq \frac{\mu - (1-a)}{a} \tag{72}$$

for which we can substitute a for $1 - a$ to obtain

$$P\left[Z > a\right] \geq \frac{\mu - a}{1-a}. \tag{73}$$

By substituting $a = \frac{1}{8}$ and $\mu = \frac{1}{4}$ from $\mathbb{E}_{S\sim\rho_i^n}[\mathcal{E}_{\rho_i}(A(S))] \geq \frac{1}{4}$ we can obtain

$$P\left[Z > \frac{1}{8}\right] \geq \frac{\frac{1}{4} - \frac{1}{8}}{1 - \frac{1}{8}} = \frac{1}{7}. \tag{74}$$

Combining the above results, we obtain

$$P_{S\sim\rho^n}\left(\mathcal{E}_\rho\left(A\left(S\right)\right) > \frac{1}{8}\right) \geq \frac{1}{7}. \tag{75}$$

11.g

11.i

No-Free-Lunch Theorem: Let A be any learning algorithm for the task of binary classification with respect to the 0 - 1 loss over a domain $\mathcal{X}$. Let n be any number smaller than, $\frac{|\mathcal{X}|}{2}$, representing a training set size. Then, there exists a distribution ρ over $\mathcal{X} \times \{0, 1\}$ such that:

1. There exists a function $f : \mathcal{X} \to \{0, 1\}$ such that $\mathcal{E}_\rho(f) = 0$
2. With probability of at least $\frac{1}{7}$ over the choice of $S \sim \rho^n$ such that $\mathcal{E}_\rho(A(S)) \geq \frac{1}{8}$

11.ii

The No-Free-Lunch Theorem states that regardless of which algorithm A we use, the misclassification excess risk we obtain of A will never be similar to $\inf_{f\in H}\mathcal{E}_\rho\left(f\right)$ for some ρ. In order for a space $\mathcal{H}$ of functions to be learnable, the misclassification excess risk for A and $\inf_{f\in H}\mathcal{E}_\rho\left(f\right)$ must be similar for any ρ, meaning that the space $\mathcal{Y}^X$ is not learnable.

11.iii

The No-Free-Lunch Theorem proves there is no universal learner. We showed that no learner can succeed in all learning tasks, as formalised by the above results. The No-Free-Lunch Theorem states that for every learner, there exists a task on which it fails, even though that task can be successfully learned by another learner. If we take our proposed case as an example, a trivial successful learner would be an ERM learner with the hypothesis class $\mathcal{H} = \{f\}$, or more generally, ERM with respect to any finite hypothesis class that contains f and whose size satisfies the equation $n \geq 8 \log\left(\frac{7|\mathcal{H}|}{6}\right)$.