Supervised Learning

November 2024

Part I [50%]

1.1 Linear Regression

Linear regression overview: Given a set of data:

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\} \tag{1}$$

where $x = (x_1, \ldots, x_n)$ is a vector in $\mathbb{R}^n$ and y is a real number. Linear regression finds a vector $\mathbf{w} \in \mathbb{R}^n$ such that the sum of squared errors

$$\text{SSE} = \sum_{t=1}^{m} (y_t - w \cdot x_t)^2 \tag{2}$$

is minimized. This is expressible in matrix form by defining X to be the $m \times n$ matrix

$$X = \begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,n} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m,1} & x_{m,2} & \cdots & x_{m,n} \end{pmatrix}, \tag{3}$$

and defining $\mathbf{y}$ to be the column vector $\mathbf{y} = (y_1, \ldots, y_m)$. The vector $\mathbf{w}$ then minimizes

$$(Xw - y)^\top (Xw - y).$$

In linear regression with basis functions, we fit the data sequence with a linear combination of basis functions $(\phi_1, \phi_2, \ldots, \phi_k)$ where $\phi_i : \mathbb{R}^n \to \mathbb{R}$ which defines a feature map from $\phi : \mathbb{R}^n \to \mathbb{R}^k$:

$$\phi(x) = (\phi_1(x), \ldots, \phi_k(x)), \quad x \in \mathbb{R}^n.$$

We use the basis functions to transform the data as follows

$$\{((\phi_1(x_1), \ldots, \phi_k(x_1)), y_1), \ldots, ((\phi_1(x_m), \ldots, \phi_k(x_m)), y_m)\}, \tag{4}$$

and then applying linear regression above to this transformed dataset. In matrix notation, we may denote this transformed dataset as

$$\Phi := \begin{pmatrix} \phi(x_1) \\ \vdots \\ \phi(x_m) \end{pmatrix} = \begin{pmatrix} \phi_1(x_1) & \cdots & \phi_k(x_1) \\ \vdots & \ddots & \vdots \\ \phi_1(x_m) & \cdots & \phi_k(x_m) \end{pmatrix}, \quad (m \times k) \tag{5}$$

Linear regression on the transformed dataset thus finds a k-dimensional vector $\mathbf{w} = (w_1, \ldots, w_k)$ such that

$$(\Phi w - y)^\top (\Phi w - y) = \sum_{t=1}^{m} \left(y_t - \sum_{i=1}^{k} w_i \phi_i(x_t) \right)^2 \tag{6}$$

is minimized.

A common basis ($n = 1$) used is the polynomial basis $\{\phi_1(x) = 1, \phi_2(x) = x, \phi_3(x) = x^2, \phi_4(x) = x^3, \ldots, \phi_k(x) = x^{k-1}\}$ of dimension k (order $k-1$). In the figure below, we give a simple fit of four points produced by a linear ($k = 2$) and cubic ($k = 4$) polynomial.

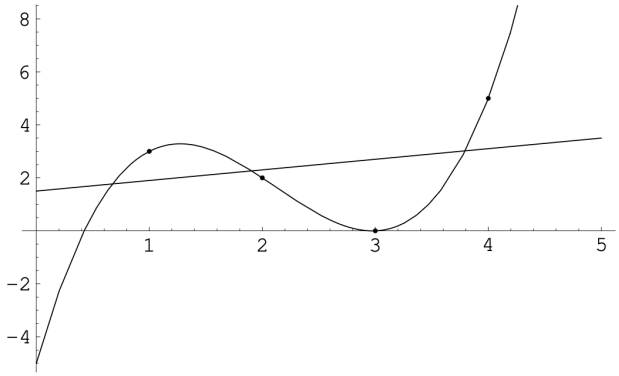

Figure 1: Data set $\{(1,3),(2,2),(3,0),(4,5)\}$ fitted with basis $\{1, x\}$ and basis $\{1, x, x^2, x^3\}$.

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

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

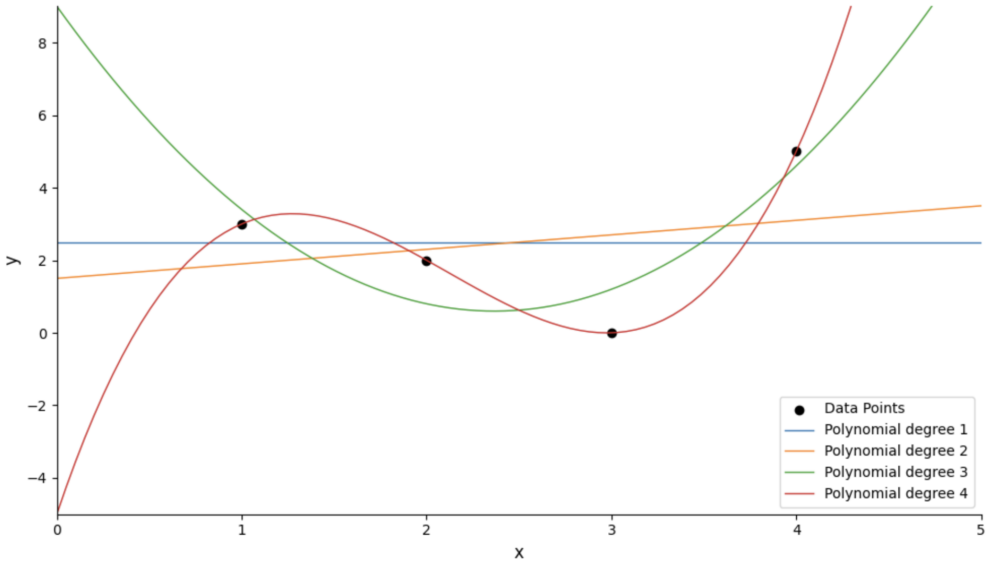

Figure 2: Curves representing the fit of polynomials from degree 1 to 4

(b) Give the equations corresponding to the curves fitted for $k = 1, 2, 3$. The equation corresponding to $k = 4$ is $-5 + 15.17x - 8.5x^2 + 1.33x^3$.

The equations of the curves from Fig. 2 are the following:

$$k = 1: \quad y = 2.50$$

$$k = 2: \quad y = 1.50 + 0.40x$$

$$k = 3: \quad y = 9.00 - 7.10x + 1.50x^2$$

$$k = 4: \quad y = -5.00 + 15.17x - 8.50x^2 + 1.33x^3$$

(c) For each fitted curve $k = 1, 2, 3, 4$ give the mean square error where

$$\text{MSE} = \frac{\text{SSE}}{m}.$$

For each curve the MSE is given by:

$$\begin{aligned} k = 1 &: \text{MSE:} \quad 3.25 \\ k = 2 &: \text{MSE:} \quad 3.05 \\ k = 3 &: \text{MSE:} \quad 0.80 \\ k = 4 &: \text{MSE:} \quad 0.00 \end{aligned}$$

2. [10 pts]: In this part, we will illustrate the phenomena of *overfitting*.

(a) Define

$$g_\sigma(x) := \sin^2(2\pi x) + \epsilon, \tag{7}$$

where ϵ is a random variable distributed normally with mean 0 and variance σ^2. Thus, $g_\sigma(x)$ is a *random* function such that $\sin^2(2\pi x)$ is computed and then the normal noise is added on each "call" of the function. We then sample "x_i" uniformly at random from the interval $[0, 1]$ 30 times, creating $(x_1, \ldots, x_{30})$, and apply $g_{0.07}$ to each x, creating the data set

$$S_{0.07,30} = \{(x_1, g_{0.07}(x_1)), \ldots, (x_{30}, g_{0.07}(x_{30}))\}. \tag{8}$$

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

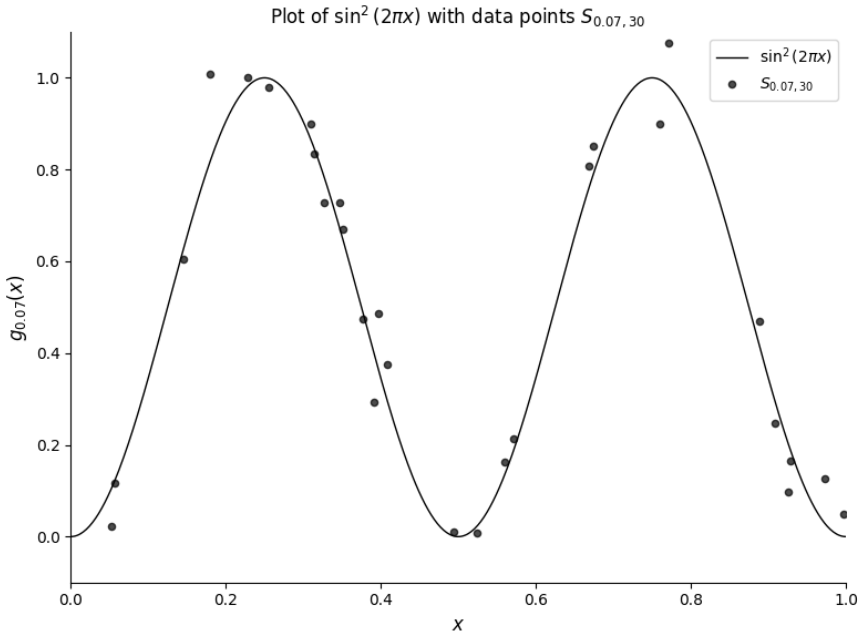

Figure 3: Plot of $\sin^2(2\pi x)$ within the range $0 \leq x \leq 1$ superimposed by points $S_{0.07,30}$

ii. Fit the data set with a polynomial bases of dimension $k = 2, 5, 10, 14, 18$ plot each of these 5 curves superimposed over a plot of data points.

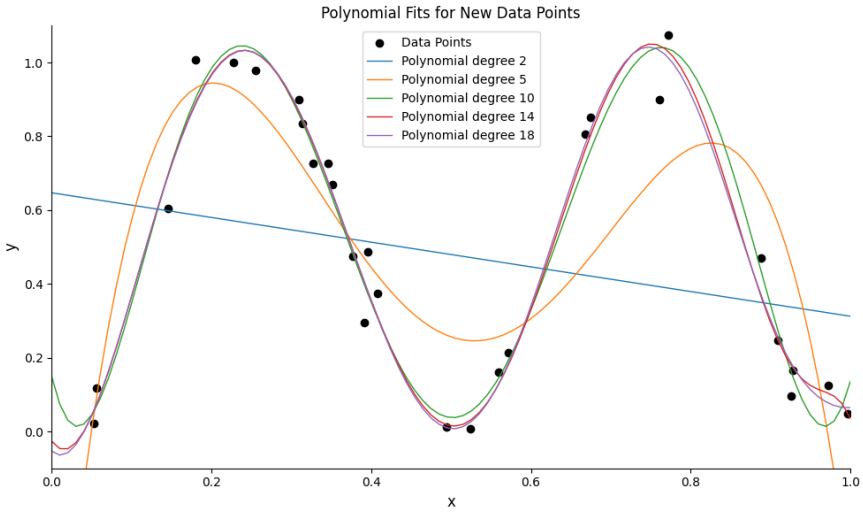

Figure 4: Fit of polynomials of degree [2,5,10,14,18] to data points

(b) Let the training error $\text{te}_k(S)$ denote the MSE of the fitting of the data set S with polynomial basis of dimension k. Plot the natural log (ln) of the training error versus the polynomial dimension $k = 1, \ldots, 18$ (this should be a decreasing function).

Our plot is given in figure 5 by the blue line.

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

$$T_{0.07,1000} = \{(x_1, g_{0.07}(x_1)), \ldots, (x_{1000}, g_{0.07}(x_{1000}))\}. \tag{9}$$

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

Our plot for the test error is given in figure 5 by the orange line. We can see the error starting to increase again after k=7.

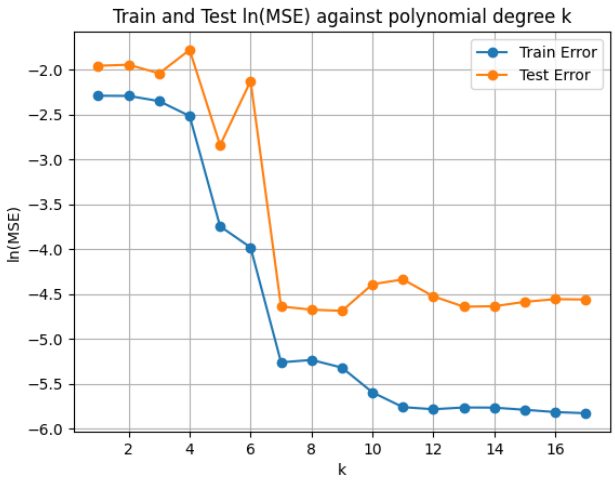

Figure 5: Train and test natural logarithm of the MSE for polynomials of degree k

(d) For any given set of random numbers, we will get slightly different training curves and test curves. It is instructive to see these curves smoothed out. For this part, repeat items (b) and (c), but instead of plotting the results of a single "run," plot the average results of 100 runs (note: plot ln(avg) rather than the avg(ln)).

Looking at figure 6 we can clearly see the phenomenon of overfitting when averaging over 100 runs, the test error curve shoots up when the train error curve goes down.

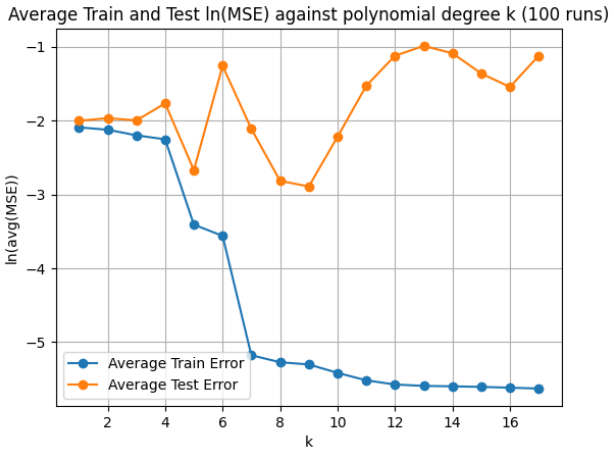

Figure 6: Graph of the average train error and test error when averaging for 100 runs

3. [5 pts]:

Now use the basis (for $k = 1, \ldots, 18$)

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

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

In figure 7 we can see the train and test error for value of k from 1 to 18 with a sine basis.

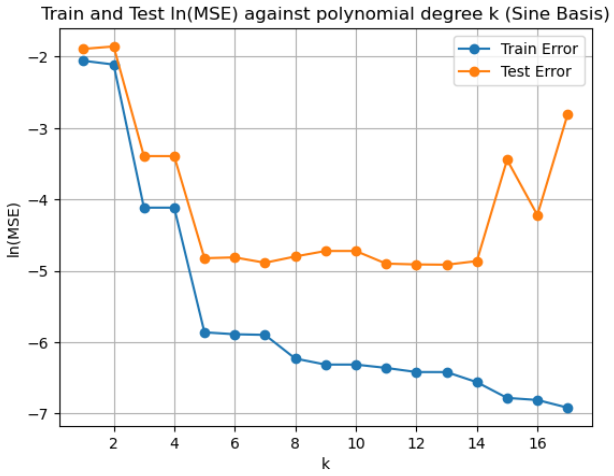

Figure 7: Graph of the train and test error for values of k from 1 to 18 with a sine basis.

In figure 8 we can see the train and test error for value of k from 1 to 18 with a sine basis averaged over 100 runs.

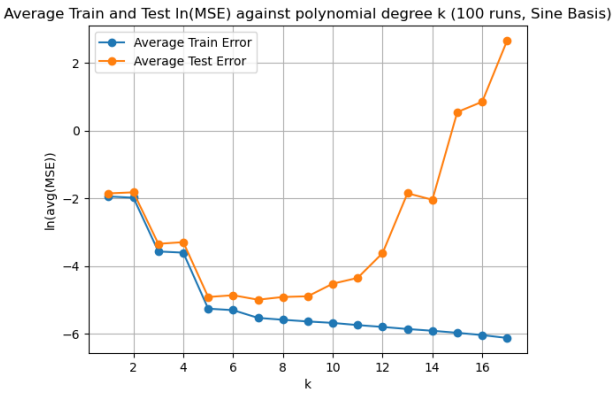

Figure 8: Graph of the train and test error for values of k from 1 to 18 with a sine basis averaged over 100 runs.

1.2 Filtered Boston housing and kernels

In this section, we will use kernel methods to extend linear regression. Boston housing is a classic dataset where you are given 13 values, and the goal is to predict the 14th, which is the median house price. Instead of the original dataset, we will instead use a modified dataset removing the ethically-suspect "column B." Thus, we will use 12 attributes to predict the 13th. The unmodified dataset is described in more detail at http://www.cs.toronto.edu/~delve/data/boston/bostonDetail.html. There are 506 entries which we will split into train and test. The **filtered** Boston housing dataset as a ".csv" file is located at http://www.cs.ucl.ac.uk/staff/M.Herbster/boston-filter.

4. [10 pts]: Baseline versus full linear regression.

Rather than use all of our attributes for prediction, it is often useful to see how well a baseline method works for a problem. In this exercise, we will compare the following:

(a) Predicting with the mean y-value on the training set.

(b) Predicting with a single attribute and a bias term.

(c) Predicting with all the attributes.

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

a. **Naive Regression.** Create a vector of ones that is the same length as the training set using the function `ones`. Do the same for the test set. By using these vectors, we will be fitting the data with a constant function. Perform linear regression on the training set. Calculate the MSE on the training and test sets and note down the results.

Average Training MSE: 82.83 Average Test MSE: 87.82

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

The constant function in part "a" simply predicts the average MEDV value from the training set for every observation, ignoring any influence of features like crime rate or number of rooms. This naive approach assumes that all variability in housing prices is random and does not capture patterns in the data. It serves as a baseline: if a more complex model cannot outperform this constant prediction, it suggests that the model may not be effectively using the available features to predict MEDV.

c. **Linear Regression with single attributes.** For each of the twelve attributes, perform a linear regression using only the single attribute but incorporating a bias term so that the inputs are augmented

with an additional 1 entry, $(x_i, 1)$, so that we learn a weight vector $\mathbf{w} \in \mathbb{R}^2$. For each of the twelve regressions, calculate the MSE on the training and test sets and note down the results.

Attribute	Average Training MSE	Average Test MSE
CRIM	70.15	76.35
ZN	72.87	75.05
INDUS	63.32	67.63
CHAS	80.51	85.25
NOX	67.65	72.12
RM	42.87	45.44
AGE	71.18	75.25
DIS	77.66	82.43
RAD	70.55	75.60
TAX	64.38	69.23
PTRATIO	62.06	64.15
LSTAT	37.92	40.03

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

Average Training MSE: 21.94 Average Test MSE: 24.63

1.3 Kernelised Ridge Regression

For nonlinear regression, the dual version will prove important. Obviously, linear regression is not capable of achieving a good predictive performance on a nonlinear data set. Here, the dual formulation will prove extremely useful, in combination with the *kernel trick*.

For our exercises, we will use a slight variation on the optimisation (as compared to the notes) that defines ridge regression for a training set with ℓ examples,

$$\mathbf{w}^* = \arg\min_{\mathbf{w}\in\mathbb{R}^n} \frac{1}{\ell}\sum_{i=1}^{\ell} \left(\mathbf{x}_i^\top \mathbf{w} - y_i\right)^2 + \gamma \mathbf{w}^\top \mathbf{w}. \tag{10}$$

i.e., we have replaced the sum of the square errors with the mean square error (MSE). For a given kernel function K, define the kernel matrix $\mathbf{K}$ for a training set of size ℓ elementwise via

$$K_{i,j} := K(\mathbf{x}_i, \mathbf{x}_j).$$

The dual optimisation formulation after kernelization is

$$\boldsymbol{\alpha}^* = \arg\min_{\boldsymbol{\alpha}\in\mathbb{R}^\ell} \frac{1}{\ell}\sum_{i=1}^{\ell} \left(\sum_{j=1}^{\ell} \alpha_j K_{i,j} - y_i\right)^2 + \gamma \boldsymbol{\alpha}^\top \mathbf{K}\boldsymbol{\alpha}. \tag{11}$$

Now, if we use $\mathbf{y} := (y_1, \ldots, y_\ell)^\top$ to denote a vector that contains the y-values of the training set, we may solve in the dual as follows:

$$\boldsymbol{\alpha}^* = \left(\mathbf{K} + \gamma \mathbf{I}_\ell\right)^{-1} \mathbf{y}. \tag{12}$$

$\mathbf{I}_\ell$

denotes the $\ell \times \ell$ identity matrix. The evaluation of the regression function on a test point can be reformulated as:

$$y_{\text{test}} = \sum_{i=1}^{\ell} \alpha_i^* K(\mathbf{x}_i, \mathbf{x}_{\text{test}}), \tag{13}$$

where the K is the kernel function.

5. [20 pts] "Kernel Ridge Regression"

In this exercise, we will perform kernel ridge regression (KRR) on the dataset using the Gaussian kernel,

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right). \tag{14}$$

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

a. Create a vector of γ values $[2^{-40}, 2^{-39}, \ldots, 2^{-26}]$ and a vector of σ values $[2^7, 2^{7.5}, \ldots, 2^{12.5}, 2^{13}]$ (recall that σ is a parameter of the Gaussian kernel; see equation (14)). Perform kernel ridge regression on the *training set* using five-fold cross-validation to choose among all pairings of the values of γ and σ. Choose the γ and σ values that perform the best to compute the predictor (by then retraining with those parameters on the training set) that you will use to report the test and training error.

γ^* (optimal regularisation strength): 2.33e-10

σ^* (optimal kernel coefficient): 1024,

Optimal Cross-Validation (MSE): 16.97

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

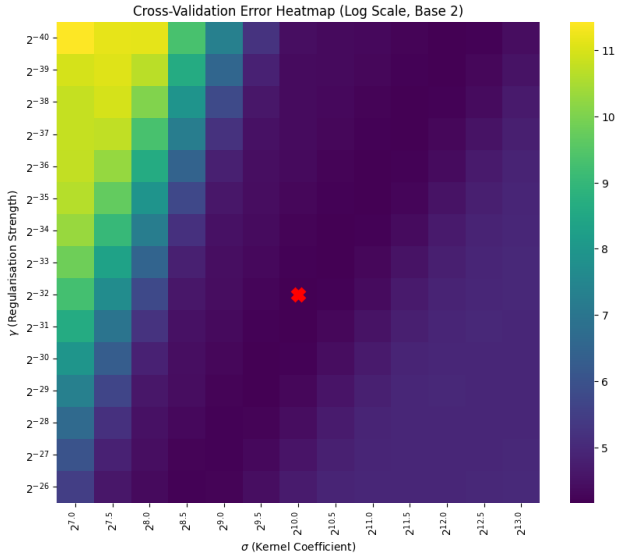

Figure 9: Heatmap of the cross-validation error as a function of gamma and sigma.

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

Training Error (MSE): 8.05

Test Error (MSE): 12.59

d. Repeat "exercise 5a,c" and "exercise 5a,c" over 20 random $2/3, 1/3$ splits of your data. Record the train/test error and the standard deviations (σ') of the train/test errors and summarise these results in the following type of table. *Additional clarification:* When repeating 5a,c, you will thus be generating 20 best (γ, σ) pairs.

Our results' table is the following:

Method	**MSE train**	**MSE test**
Naive Regression	82.83 ± 4.68	87.82 ± 9.37
Linear Regression (attribute 1)	70.15 ± 4.70	76.35 ± 10.83
Linear Regression (attribute 2)	72.87 ± 4.95	75.05 ± 9.92
Linear Regression (attribute 3)	63.32 ± 4.70	67.63 ± 9.46
Linear Regression (attribute 4)	80.51 ± 4.19	85.25 ± 8.88
Linear Regression (attribute 5)	67.65 ± 4.28	72.12 ± 8.67
Linear Regression (attribute 6)	42.87 ± 2.90	45.44 ± 5.85
Linear Regression (attribute 7)	71.18 ± 4.63	75.25 ± 9.34
Linear Regression (attribute 8)	77.66 ± 4.92	82.43 ± 9.86
Linear Regression (attribute 9)	70.55 ± 4.29	75.60 ± 8.67
Linear Regression (attribute 10)	64.38 ± 4.42	69.23 ± 8.90
Linear Regression (attribute 11)	62.06 ± 3.95	64.15 ± 8.00
Linear Regression (attribute 12)	37.92 ± 2.10	40.03 ± 4.41
Linear Regression (all attributes)	21.94 ± 1.83	24.63 ± 3.81
Kernel Ridge Regression	8.24 ± 0.81	12.57 ± 2.10

2 PART II [20%]

2.1 k-Nearest Neighbors

In this section, you will implement the k-NN algorithm and explore its performance as a function of k. Given a new data point, the k-NN algorithm attributes its label to the majority label of its k nearest neighbors. See here for details.

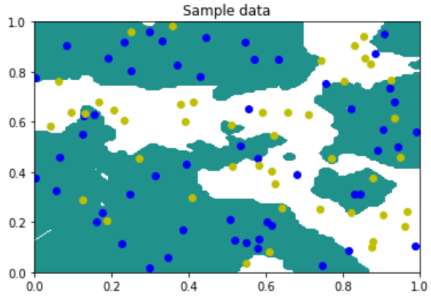

Figure 10: A hypothesis $h_{S,v}$ visualized with $|S| = 100$ and $v = 3$.

2.1.1 Generating the data

A voted-center hypothesis is a function $h_{S,v} : [0,1]^2 \to \{0,1\}$, where $S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_{|S|}, y_{|S|})\}$ is a set of labeled centers with each $x_i \in [0,1]^2$ and $y_i \in \{0,1\}$ and v is a positive integer. In this coursework, we will always set $v := 3$ for all of the exercises in this subsection. Where $h_{S,v}(x) = 0$ if the majority of the v nearest x_i to x in S are 0; similarly for $h_{S,v}(x) = 1$ and finally in certain "corner" cases where the "majority of v" will not be well-defined and in that case $h_{S,v}(x) = 0$. See Section 2.1.4 for addressing these corner cases.

In Figure 10, one such hypothesis is visualized with $|S| = 100$ and $v = 3$. The white areas in the mapping to 0 and the turquoise areas to 1, the corresponding centers are green and blue.

[5 pts] Produce a visualization of an $h_{S,v}$ similar to the figure.

First, we will need to generate a random h. Do this by sampling 100 centers uniformly at random from $[0, 1]^2$ with 100 corresponding labels sampled uniformly at random from $\{0, 1\}$. Later, we will need to generate more random hypotheses. Call this distribution over hypotheses generated by this random process P_h.

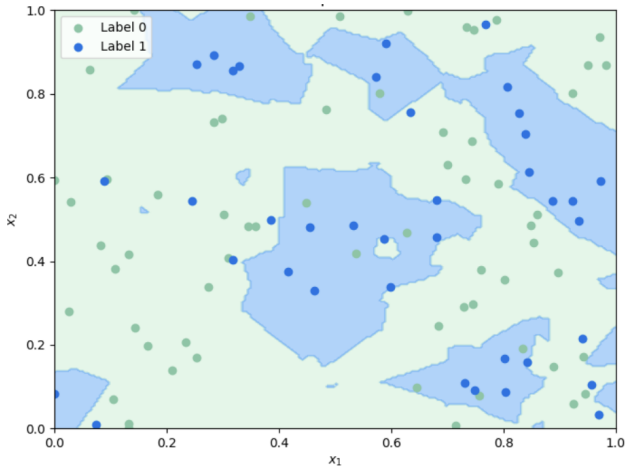

Figure 11: Plot of $h_{S,v}$

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

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

Generating our noisy data. Given an $h_{S,v}$, the underlying probability distribution $p(x, y)$ is determined by sampling x uniformly at random from $[0, 1]^2$ and then its corresponding y value is then generated by flipping based on $P(\text{heads}) = 0.5P(h_{S,v}(x) = 1)$. When you generate training and test sets, they will both come from this distribution.

[7 pts] a) Produce a visualization using Protocol A (see Figure 14). b) By using roughly 5 sentences, explain why the figure is the approximate shape that it is and comment on any interesting features of the figure.

For low values of k, the clusters are small enough that they over-fit to the dataset. From the bias-variance trade-off perspective, we can say that the variance is too high, and the model captures noise in the training data. As we increase k, the clusters become larger and the model gradually increases in bias as it cannot fit the data as effectively. It begins to miss finer details that have predictive capability. The optimal k for this dataset is found at k = 7.

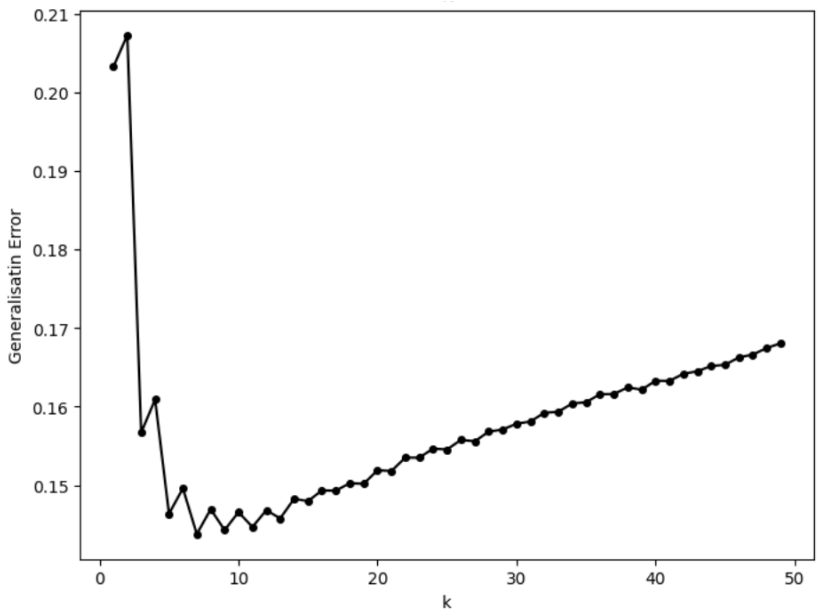

Figure 12: Generalisation error fro different values of k

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

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

[8 pts] a) Produce a visualisation using Protocol B (see Figure 15). I.e, plotting m versus optimal k b) Using roughly 4 sentences explain why the figure is the approximate shape that it is.

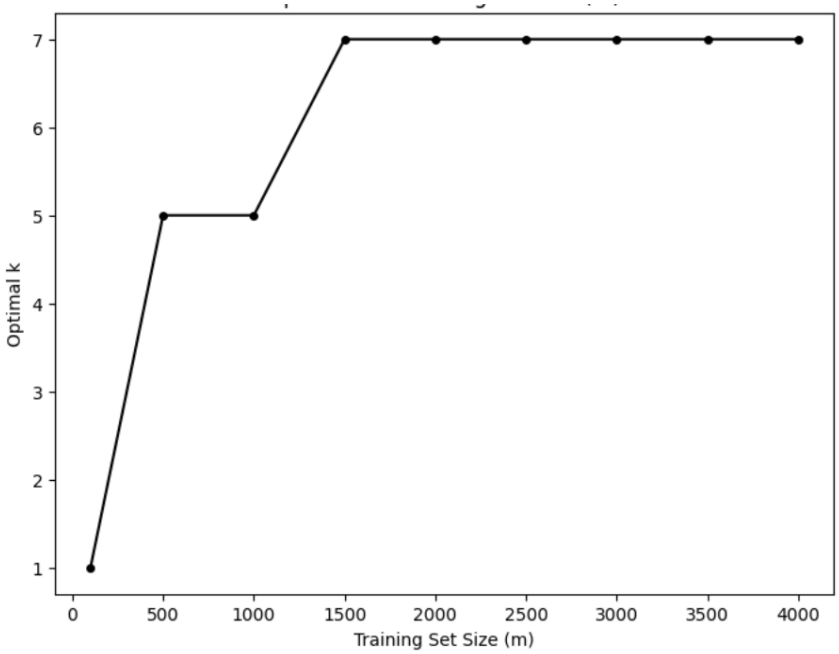

Figure 13: Optimal k vs Training Set Size (m)

As the training set size grows, the optimal k increases because larger training sets make it possible to use a larger neighbourhood without over-fitting to noise. For large m, the optimal k stabilises. This happens because additional data points do not significantly change the distribution of the data or improve the generalisation further. The optimal k is large enough to balance the trade-off between variance and bias.

2.1.4 Additional Clarification

```
For each k ∈ {1,...,49}
    Do 100 runs ...
        Sample a h from p_H
        Build a k-NN model with 4000 training points sampled from p_h(x,y).
        Run k-NN estimate generalisation error (for this run)
          using 1000 test points sampled from p_h(x,y)
    The estimated generalization error (y-axis) is then the mean
      of these 100 ''run'' generalisation errors.
```

Figure 14: Experimental Protocol A: k-NN. For computational purposes, you may reorder the loops in the protocol as long as this is done in a principled way.

```
For each m ∈ {100,500,1000,1500,...,4000}
    Do 100 runs ...
         For each k ∈ {1,...,49}
              Sample a h from p_H
              Build a k-NN model with m training points sampled from p_h(x,y).
              Run k-NN estimate generalisation error (for this run)
                  using 1000 test points sampled from p_h(x,y)
               The estimated optimal k (for this run) is then the k
                 with minimal estimated generalisation error.
    The estimated optimal k (y-axis) is then the mean
      of these 100 ''run'' optimal 'k's.
```

Figure 15: Experimental Protocol B: k-NN. For computational purposes, you may reorder the loops in the protocol, as long as this is done in a principled way.

Note both the k-NN algorithm and the hypothesis $h_{S,v}$ can produce "undefined" results in certain corner cases for individual x's. Resolve these cases by generating either the label or prediction uniformly at random.

Part III [30%]

9. Kernel Modification

(a) To determine for which values of $c \in \mathbb{R}$ the kernel

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

is positive semi-definite, we note that a kernel K is positive semi-definite if, for any set of points $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\} \subset \mathbb{R}^n$, the matrix $[K(\mathbf{x}_i, \mathbf{x}_j)]$ is positive semi-definite. This means that for any vector $\mathbf{a} \in \mathbb{R}^m$,

$$\sum_{i=1}^{m} \sum_{j=1}^{m} a_i a_j K(\mathbf{x}_i, \mathbf{x}_j) \geq 0.$$

We can rewrite $K_c(\mathbf{x}, \mathbf{z})$ as:

$$K_c(\mathbf{x}, \mathbf{z}) = c + \mathbf{x} \cdot \mathbf{z},$$

where $\mathbf{x} \cdot \mathbf{z}$ is the standard inner product, a known positive semi-definite kernel.

The constant c adds a uniform shift to the kernel matrix. If $c \geq 0$, this preserves the positive semi-definiteness of the inner product kernel. If $c < 0$, the kernel matrix may gain negative eigenvalues, violating positive semi-definiteness.

Therefore, K_c remains positive semi-definite if $c \geq 0$.

(b) When we use $K_c(\mathbf{x}, \mathbf{z}) := c + \mathbf{x} \cdot \mathbf{z}$ in linear regression, c affects the solution by uniformly shifting all values in the kernel matrix K. Each entry is $K_{ij} = c + \mathbf{x}_i \cdot \mathbf{x}_j$, so increasing c raises the similarity scores between all data points equally.

For small values of c, this shift keeps the original structure of the kernel matrix largely intact, preserving distinctions between points and adding slight regularisation by increasing stability.

If c becomes very large, it overwhelms the dot product values, making all entries in K nearly equal. This effectively treats all data points as equally similar, leading to underfitting, as the model can no longer distinguish between points and fails to capture meaningful patterns.

In summary, c influences the balance between stability and sensitivity: a small positive c stabilises the model, while a large c leads to underfitting by erasing differences between data points.

This is somewhat analogous to introducing a soft margin in SVMs, where allowing slight violations of the margin constraint can improve generalisation. Similarly, adding c provides regularisation, making the model less sensitive to specific data variations and balancing between a strict fit and better generalisation.

10. Gaussian Kernel Classifier

To enable a Gaussian kernel-based linear classifier to simulate a 1-Nearest Neighbour (1-NN) classifier, we choose a large value of β in the Gaussian kernel $K_\beta(x, t) = \exp(-\beta\|x - t\|^2)$. As $\beta \to \infty$, the kernel value $K_\beta(x_i, t)$ decays rapidly with increasing distance between x_i and t. For sufficiently large β, this decay ensures that $K_\beta(x_i, t)$ is close to zero for all points x_i except the nearest point x_k to t. Consequently, the function

$$f(t) = \sum_{i=1}^{m} \alpha_i K_\beta(x_i, t)$$

is dominated by the term involving the nearest neighbour x_k, making $f(t) \approx \alpha_k K_\beta(x_k, t)$. The classifier then outputs $\text{sign}(f(t))$, which, for large β, primarily reflects the label y_k of the nearest neighbour. In summary, selecting β to be sufficiently large effectively isolates the influence of the nearest neighbour on $f(t)$, enabling the classifier to behave like a 1-NN classifier by basing its decision solely on the label of the closest point.

11. No Free Lunch Theorem

(a) Let $C \subset \mathcal{X}$ be a subset of the input space with cardinality $|C| = 2n$. Denote by $\mathcal{Y}^C = \{f_1, \ldots, f_T\}$ the set of all possible functions from C to $\mathcal{Y} = \{-1, 1\}$. There are $T = |\mathcal{Y}^C| = 2^{2n}$ such functions. For each function $f_i \in \mathcal{Y}^C$, we define a distribution ρ_i over $C \times \mathcal{Y}$ such that $\rho_i(\{x, y\}) = \frac{1}{2n}$ if $y = f_i(x)$, and 0 otherwise. This distribution assigns a probability of $\frac{1}{2n}$ to each pair (x, y) where $y = f_i(x)$, and zero otherwise.

The expected loss $\mathbb{E}_{\rho_i}(f_i)$ is then calculated as follows:

$$\mathbb{E}_{\rho_i}(f_i) = \sum_{x \in C} \sum_{y \in \{-1,1\}} \mathbf{1}_{\{f_i(x) \neq y\}} \rho_i(\{x, y\}).$$

Since $\rho_i(\{x, y\})$ is non-zero only when $y = f_i(x)$, the indicator function $\mathbf{1}_{\{f_i(x) \neq y\}}$ is always zero.

Therefore, we conclude that $\mathbb{E}_{\rho_i}(f_i) = 0$. This establishes that

$$\inf_{f:\mathcal{X}\to\mathcal{Y}} \mathbb{E}_{\rho}(f) = 0.$$

(b) Let C^n denote the set of all possible datasets of size n consisting of points in C, i.e., $C^n = \{S_1, \ldots, S_k\}$ where $k = |C^n| = (2n)^n$. For each $j = 1, \ldots, k$ and for each input set $S_j = \{x_1, \ldots, x_n\} \in C^n$, define the corresponding dataset for function f_i as

$$S_j^i = \{(x_1, f_i(x_1)), \ldots, (x_n, f_i(x_n))\}.$$

This dataset S_j^i is constructed by associating each $x \in S_j$ with its output under f_i.

We seek to prove that

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

Since the distribution is ρ_i as defined we have the following:

$$\mathbb{E}_{S\sim\rho_i^m} \mathbb{E}_{\rho_i}(A(S)) = \frac{1}{k} \sum_{j=1}^{k} \mathbb{E}_{\rho_i}(A(S_j^i))$$

Using the second hint which says that for any set of scalars $\alpha_1, \ldots, \alpha_m$ we have $\max_\ell \alpha_\ell \geq \frac{1}{m} \sum_{\ell=1}^{m} \alpha_\ell \geq \min_\ell \alpha_\ell$., i.e. the max is greater than the average which is itself greater than the min, we have:

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

(c) To establish a lower bound on the risk, we introduce a subset $S_j' \subset C$ such that S_j' contains elements not present in S_j. Specifically, for each dataset S_j, let $S_j' = \{v_1, \ldots, v_p\} \subset C \setminus S_j$, where $p \geq m$. This subset S_j' is chosen so that it consists of points disjoint from S_j, meaning the learning algorithm A has not encountered any elements of S_j' during training on S_j.

We want to show the following inequality:

$$\frac{1}{T} \sum_{i=1}^{T} \mathcal{E}_{\rho_i}(A(S_j^i)) \geq \frac{1}{2} \min_{v\in R_j} \frac{1}{T} \sum_{i=1}^{T} \mathbf{1}\{A(S_j^i)(v) \neq f_i(v)\}.$$

The risk $\mathcal{E}_{\rho_i}(A(S_j^i))$ measures the average error of the algorithm A, trained on S_j^i, over the dataset C under the distribution ρ_i. It is defined as:

$$\mathcal{E}_{\rho_i}(A(S_j^i)) = \int_{x,y} \mathbf{1}\{A(S_j^i)(x) \neq y\} \, d\rho_i(x, y).$$

Since ρ_i assigns probability only to pairs $(x, f_i(x))$, where $x \in C$ and $f_i(x)$ is the true output:

$$\mathcal{E}_{\rho_i}(A(S_j^i)) = \frac{1}{|C|} \sum_{x\in C} \mathbf{1}\{A(S_j^i)(x) \neq f_i(x)\}.$$

The dataset C can be partitioned into two subsets: the training set S_j and its complement R_j, where $|C| = 2n$. Substituting this partition, the risk becomes:

$$\mathcal{E}_{\rho_i}(A(S_j^i)) = \frac{1}{2n}\left(\sum_{x\in S_j} \mathbf{1}\{A(S_j^i)(x) \neq f_i(x)\} + \sum_{v\in R_j} \mathbf{1}\{A(S_j^i)(v) \neq f_i(v)\}\right).$$

Since each term in the summation is non-negative, the second term provides a lower bound:

$$\mathcal{E}_{\rho_i}(A(S_j^i)) \geq \frac{1}{2n}\sum_{v\in R_j} \mathbf{1}\{A(S_j^i)(v) \neq f_i(v)\}.$$

Taking the average over T iterations:

$$\frac{1}{T}\sum_{i=1}^{T} \mathcal{E}_{\rho_i}(A(S_j^i)) \geq \frac{1}{T}\sum_{i=1}^{T}\frac{1}{2n}\sum_{v\in R_j} \mathbf{1}\{A(S_j^i)(v) \neq f_i(v)\}.$$

Rewriting:

$$\frac{1}{T}\sum_{i=1}^{T} \mathcal{E}_{\rho_i}(A(S_j^i)) \geq \frac{1}{2n}\sum_{v\in R_j}\frac{1}{T}\sum_{i=1}^{T}\mathbf{1}\{A(S_j^i)(v) \neq f_i(v)\}.$$

Define $a_v = \frac{1}{T}\sum_{i=1}^{T}\mathbf{1}\{A(S_j^i)(v) \neq f_i(v)\}$, the average error at $v \in R_j$. Using the mean-minimum inequality provided by the hint:

$$\frac{1}{|R_j|}\sum_{v\in R_j} a_v \geq \min_{v\in R_j} a_v.$$

Substitute a_v back:

$$\frac{1}{T|R_j|}\sum_{v\in R_j}\sum_{i=1}^{T}\mathbf{1}\{A(S_j^i)(v) \neq f_i(v)\} \geq \min_{v\in R_j}\frac{1}{T}\sum_{i=1}^{T}\mathbf{1}\{A(S_j^i)(v) \neq f_i(v)\}.$$

Multiply by $\frac{|R_j|}{2n}$ (where $|R_j| = n$):

$$\frac{1}{T}\sum_{i=1}^{T}\frac{1}{2n}\sum_{v\in R_j}\mathbf{1}\{A(S_j^i)(v) \neq f_i(v)\} \geq \frac{|R_j|}{2n}\min_{v\in R_j}\frac{1}{T}\sum_{i=1}^{T}\mathbf{1}\{A(S_j^i)(v) \neq f_i(v)\}.$$

Substitute $|R_j| = n$:

$$\frac{1}{T}\sum_{i=1}^{T} \mathcal{E}_{\rho_i}(A(S_j^i)) \geq \frac{1}{2}\min_{v\in R_j}\frac{1}{T}\sum_{i=1}^{T}\mathbf{1}\{A(S_j^i)(v) \neq f_i(v)\}.$$

(d) With the same assumptions as above, we now aim to show that for any $v \in R_j$,

$$\frac{1}{T}\sum_{i=1}^{T}\mathbf{1}_{\{A(S_j^i)(v)\neq f_i(v)\}} = \frac{1}{2}.$$

To achieve this, we utilize a partition of the function set $\mathcal{Y}^C$ into $T/2$ pairs of functions $(f_i, f_{i'})$ such that for each pair $(f_i, f_{i'})$, the functions differ only on a single input point $x = v$, i.e., $f_i(x) \neq f_{i'}(x)$ if and only if $x = v$. Thus, for each pair $(f_i, f_{i'})$,

$$\mathbf{1}_{\{A(S_j^i)(v)\neq f_i(v)\}} + \mathbf{1}_{\{A(S_j^{i'})(v)\neq f_{i'}(v)\}} = 1,$$

ensuring that across all pairs, the error indicator $\mathbf{1}_{\{A(S_j^i)(v)\neq f_i(v)\}}$ is $\frac{1}{2}$ on average. Therefore,

$$\frac{1}{T}\sum_{i=1}^{T}\mathbf{1}_{\{A(S_j^i)(v)\neq f_i(v)\}} = \frac{1}{2}.$$

(e) We now prove an auxiliary result. Let Z be a random variable with values in $[0,1]$ and expected value $\mathbb{E}[Z] = \mu$. We wish to show that for any $a \in (0,1)$,

$$\mathbb{P}(Z > 1-a) \geq \frac{\mu - (1-a)}{a}.$$

To derive this result, we apply *Markov's inequality*. Let $W = 1 - Z$, so $W \geq 0$ and $W \leq 1$. Then $Z > 1-a$ is equivalent to $W < a$. Since $\mathbb{E}[W] = 1-\mu$, Markov's inequality gives

$$\mathbb{P}(W \geq a) \leq \frac{\mathbb{E}[W]}{a} = \frac{1-\mu}{a}.$$

Therefore,

$$\mathbb{P}(Z > 1-a) = 1 - \mathbb{P}(W \geq a) \geq 1 - \frac{1-\mu}{a} = \frac{\mu-(1-a)}{a}.$$

(f) We now combine the results of the previous steps to show that for any algorithm A and any integer n, there exists a distribution ρ over $\mathcal{X}\times\mathcal{Y}$ such that

$$\mathbb{P}_{S\sim\rho^m}\left(\mathbb{E}_\rho(A(S)) > \frac{1}{8}\right) \geq \frac{1}{7}.$$

Let $Z = \mathbb{E}_\rho(A(S))$, representing the expected error of the algorithm A on a dataset S. From the previous steps, we have established that $\mathbb{E}_{S\sim\rho^m}[Z] \geq \frac{1}{4}$. Applying the auxiliary result from part (e) with $\mu = \frac{1}{4}$ and $a = \frac{7}{8}$, we obtain

$$\mathbb{P}\left(Z > 1 - \frac{7}{8}\right) = \mathbb{P}\left(Z > \frac{1}{8}\right) \geq \frac{\frac{1}{4} - \left(1-\frac{7}{8}\right)}{\frac{7}{8}}.$$

Simplifying this expression, we get

$$\mathbb{P}\left(Z > \frac{1}{8}\right) \geq \frac{\frac{1}{8}}{\frac{7}{8}} = \frac{1}{7}.$$

Thus, we have shown that there exists a distribution ρ for which the probability that the expected error of A exceeds $\frac{1}{8}$ is at least $\frac{1}{7}$, completing the proof. □