Supervised Learning (COMP0078) Coursework 1

1 PART I

1.1 Linear Regression

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

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

Solution: Refer to Figure 1 for the polynomial fits applied to the dataset. The code implementation is provided under the function question1.plot_polynomial_fits().

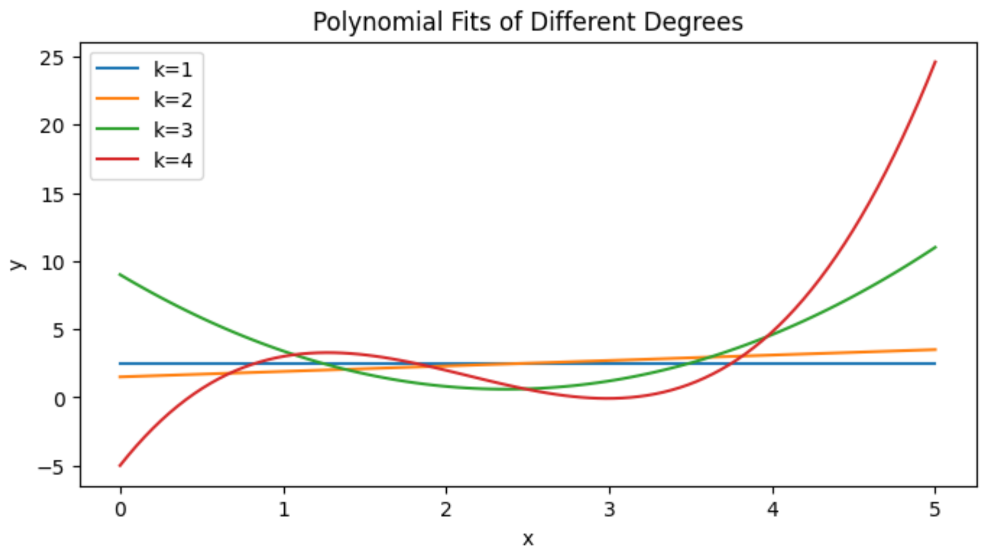

Figure 1: Polynomial fits (k = 1, 2, 3, 4) for the dataset {(1, 3), (2, 2), (3, 0), (4, 5)}, showing increasing flexibility with higher degrees.

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

Solution: The code for this part is provided under the function question1.print_polynomial_equations().

- Degree 1 polynomial: $y = 2.50$
- Degree 2 polynomial: $y = 1.50 + 0.40x$
- Degree 3 polynomial: $y = 9.00 + -7.10x + 1.50x^2$

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

Solution: The code for this part is provided under the function question1.calculate_mse()

- Error for k=1: 3.2500
- Error for k=2: 3.0500
- Error for k=3: 0.8000
- Error for k=4: 0.0117

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

(a)

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

Solution: Refer to Figure 2 for the visualization of the function $\sin^2(2\pi x)$ and the corresponding noisy data points. The code implementation is provided under the function question2.plot_function_and_data().

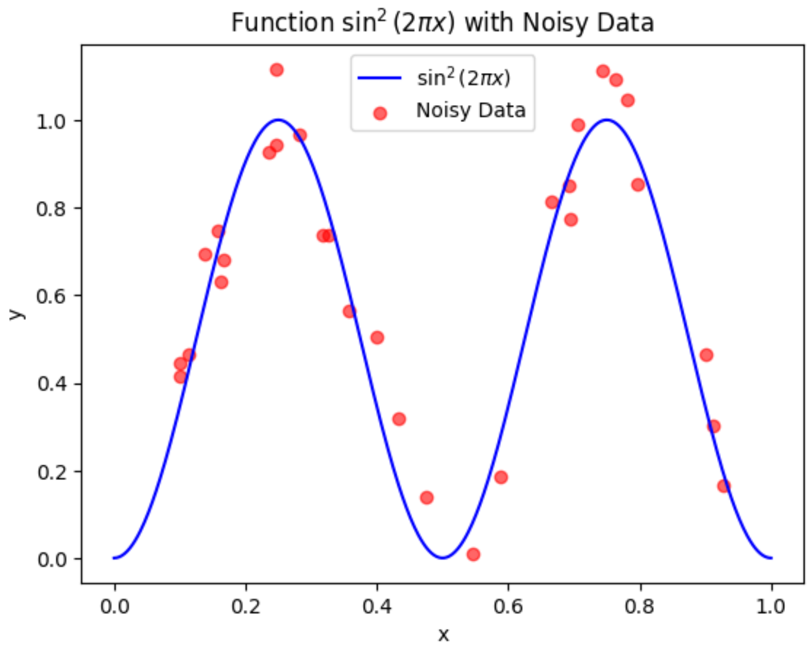

Figure 2: Plot of the function $\sin^2(2\pi x)$ (blue curve) with noisy data points (red dots) sampled uniformly from the interval [0,1]. This illustrates the relationship between the original function and the added Gaussian noise, demonstrating variability in the data.

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.

Solution: Refer to Figure 3 for the fitted curves using polynomial bases of dimensions k = 2, 5, 10, 14, 18 superimposed over the noisy dataset. The code implementation is provided under the function question2.fit_and_plot_polynomials().

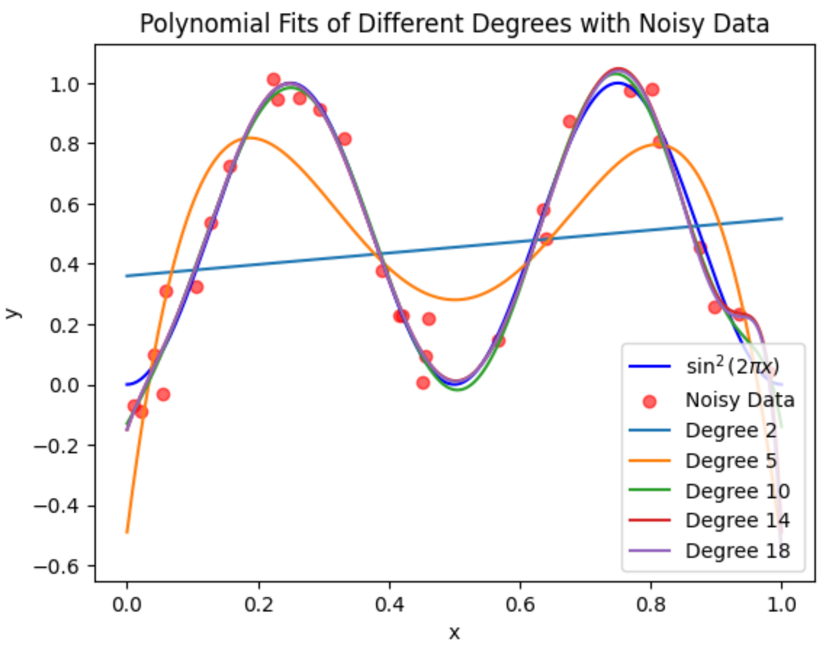

Figure 3: Polynomial fits of dimensions k = 2, 5, 10, 14, 18 plotted alongside the noisy dataset (red dots) and the original function $\sin^2(2\pi x)$ (blue curve). The increasing polynomial degree demonstrates how higher-order polynomials fit the noise more closely, illustrating overfitting as k increases.

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

Solution: Refer to Figure 4 for the plot of the natural log of the training error against polynomial dimensions k = 1 to k = 18**.** The code implementation is provided under the function question2.plot_error().

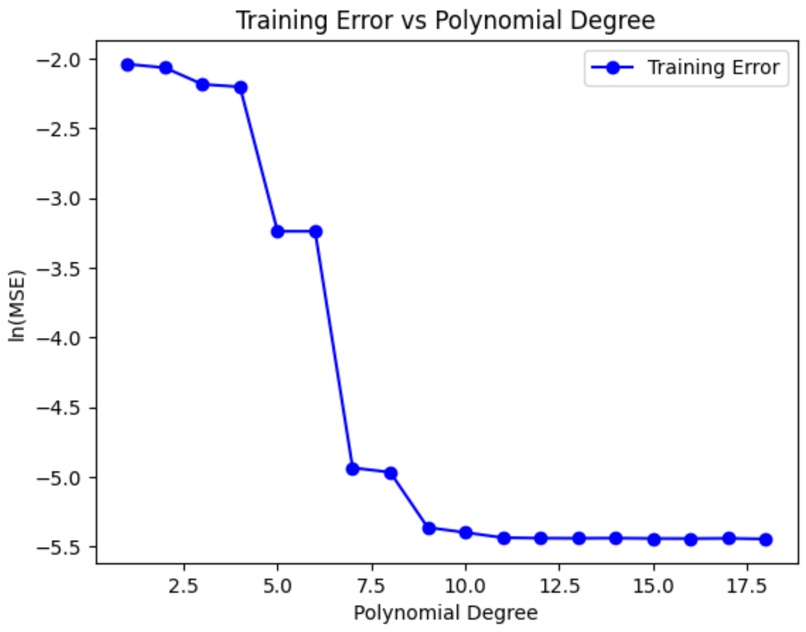

Figure 4: The natural log of the training error decreases as the polynomial dimension kk increases, showing improved fitting to the training data with higher-degree polynomials.

(c). Generate a test set T of a thousand points, $T_{0.07,1000} = \{(x_1, g_{0.07}(x_1)), \ldots, (x_{1000}, g_{0.07}(x_{1000}))\}$.

Define the test error $tse_k(S,T)$ to be the MSE of the test set T on the polynomial of dimension k fitted from training set S. Plot the ln of the test error versus the polynomial dimension k = 1, . . . , 18.

Solution: Refer to Figure 5 for the plot of the natural log of the test error versus polynomial dimensions k=1 to k=18. The code implementation is provided under the function question2.plot_error().

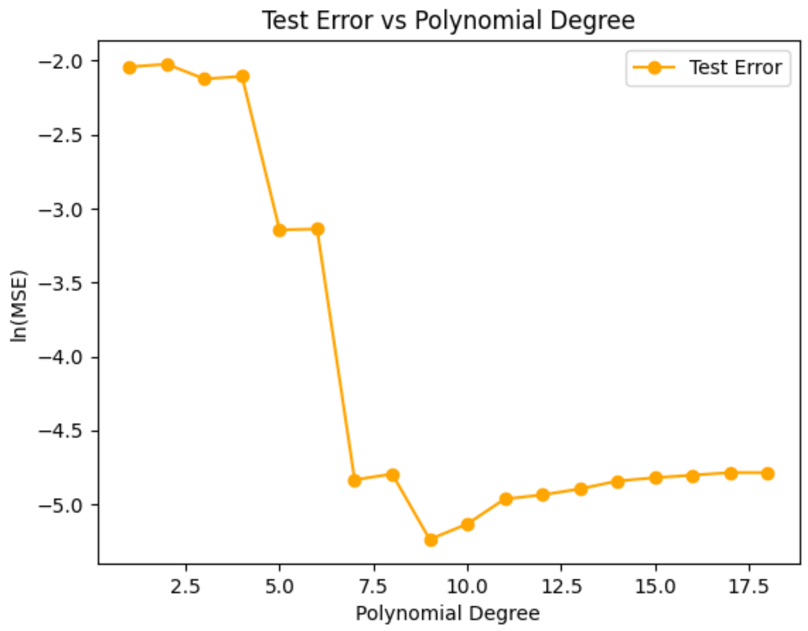

Figure 5: The natural log of the test error initially decreases with increasing polynomial degree k but begins to rise after a certain point, illustrating the phenomenon of overfitting.

(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 the ln(avg) rather than the avg(ln)).

Solution: After repeating 2(b) and 2(c) 100 times, Figure 6 presents the smoothed training and test error curves as a function of the polynomial dimension (k=1 to k=18). The code implementation is provided under the function question2.plot_smoothed_errors().

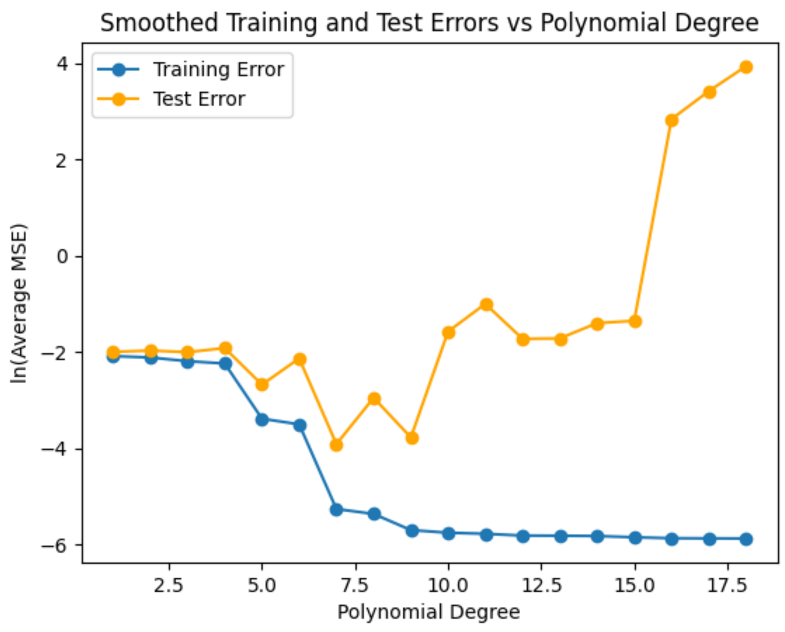

Figure 6: Smoothed plots of the natural log of the average training error (blue curve) and average test error (orange curve) over 100 runs. The training error consistently decreases with increasing polynomial degree, while the test error initially decreases but rises for higher degrees, highlighting overfitting.

3. Using the basis $\{sin(1\pi x), sin(2\pi x), sin(3\pi x), ..., sin(k\pi x)\}$ for k=1 to k=18, repeat the experiments in 2(b-d) with the above basis.

Solution: Using the sine basis for k=1 to k=18, the experiments from 2(b-d) were repeated, and the results are as follows:

- **Figure 7:** The natural log of training error versus basis dimensions. Code implementation: question3.plot_ln_mse().
- **Figure 8:** The natural log of test error versus basis dimensions. Code implementation: question3.plot_ln_mse().
- **Figure 9:** The natural log of averaged training and test errors over 100 runs for basis dimensions. Code implementation: question3.plot_average_ln_mse().

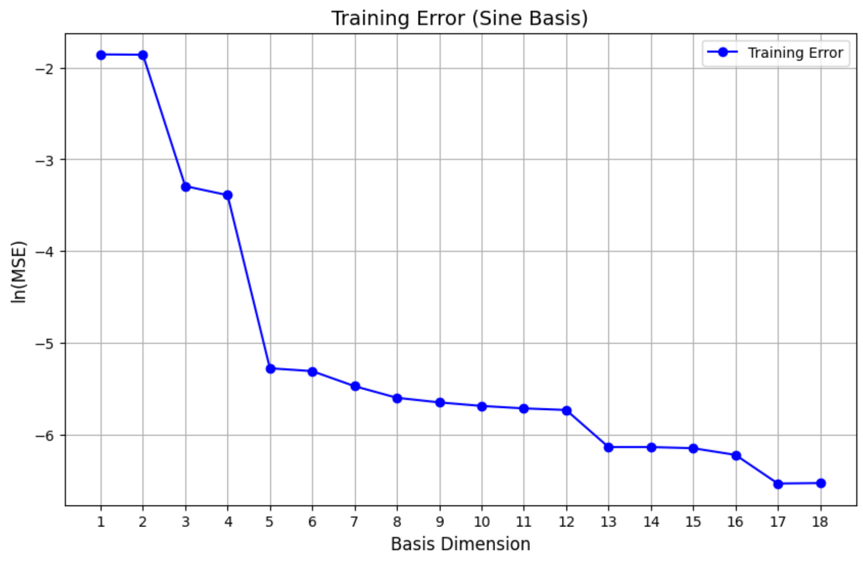

Figure 7: The training error consistently decreases as the sine basis dimension k increases, showing improved fitting to the training data.

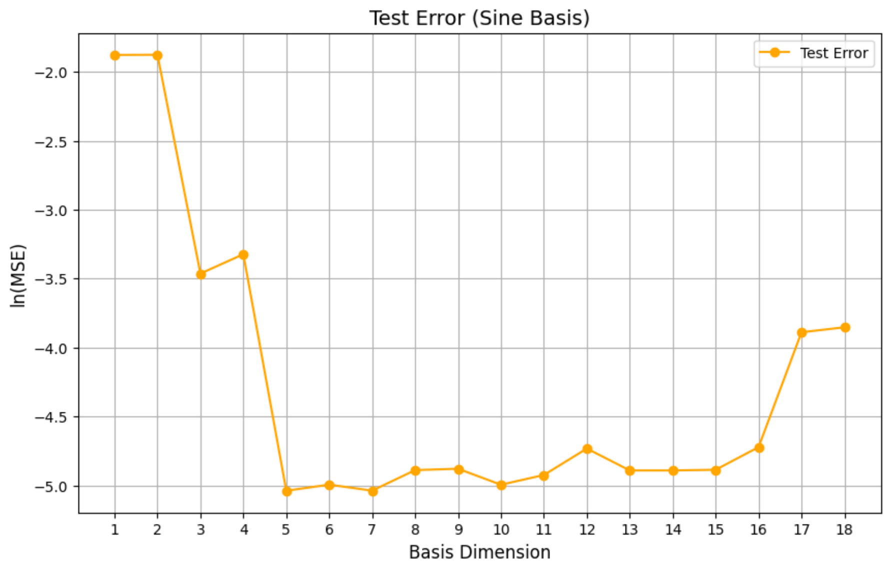

Figure 8: The test error initially decreases with increasing basis dimension kk, reaching its minimum at an intermediate kk, and then starts to rise, demonstrating overfitting.

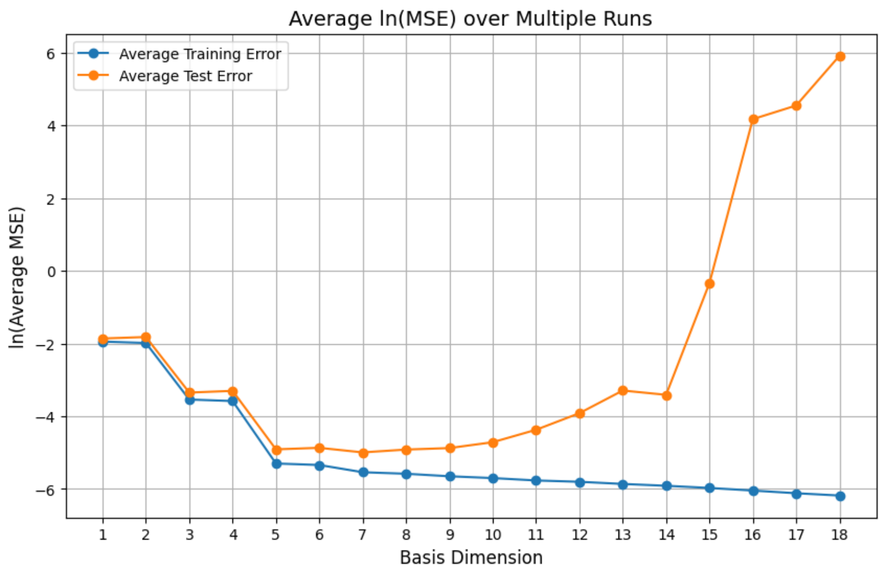

Figure 9: The smoothed training error curve (blue) decreases steadily across dimensions, while the smoothed test error curve (orange) decreases initially but rises after a certain point. This illustrates overfitting, where higher dimensions fit the training data well but fail to generalize to unseen test data.

1.2 Boston Housing and Kernels

4. Baseline versus full linear regression:

(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.

Solution:

- Average MSE on the training set is: 84.9777

- Average MSE on the test set is: 83.7771

The code implementation is provided under the function question4.baseline_regression().

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

Solution: Using a constant vector *w* to fit the data is equivalent to calculating the average of the target label values. The model sums all target values and divides by the total number of data points, effectively predicting the mean value for all inputs. This serves as a simple baseline model with no dependence on input features.

(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 $w \in R^2$.

Solution: Table 1 shows the training and testing Mean Squared Error (MSE) for linear regression performed using each individual attribute as a predictor, with a bias term added. The code implementation for this part is provided under the function question4.calculate_mse_per_attribute().

ATTRIBUTE	TRAINING MSE	TESTING MSE
1	72.2444	70.7796
2	73.7693	73.1644
3	66.8756	60.3810
4	81.4691	82.8411
5	70.5492	66.1751
6	39.1488	52.9878
7	73.6293	70.1080
8	79.9046	78.0854
9	72.1373	72.2216
10	71.2343	66.8923
11	62.4739	63.3889
12	39.6487	36.2761

Table 1: Training and testing MSE for linear regression using each of the twelve attributes from the dataset as the sole predictor. Each attribute's performance is evaluated independently, showing its ability to predict the target variable.

(d) Linear Regression using all attributes. 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.

Solution:

- MSE on the training set (with all attributes) is: 24.3617
- MSE on the testing set (with all attributes) is: 26.6876

The code implementation is provided under the function question4.average_mse_all_attributes().

1.3 Kernelized Ridge Regression

5. Kernel Ridge Regression:

(a) Create a vector of γ values [2−40, 2−39, . . . , 2−26] and a vector of σ values [27, 27.5, . . . , 212.5, 213] (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 pairing of the values of γ and σ. Choose the γ and σ values that perform the best to compute the predictor (by then retraining with those parameters on the training set) that you will use to report the test and training error.

Solution: Using kernel ridge regression with five-fold cross-validation, the optimal hyperparameters were selected from the specified ranges. The best values chosen to report the test and training errors are:

- Best γ: $7.45 * 10^{-9}$
- Best σ: 2048

The code implementation is provided under the function question5.five_fold_cross_validation().

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

Solution: Figure 10 presents the cross-validation error as a 3D plot, showing the relationship between γ and σ with the color bar indicating error magnitude. The code implementation is provided under the function question5.plot_cross_validation_results().

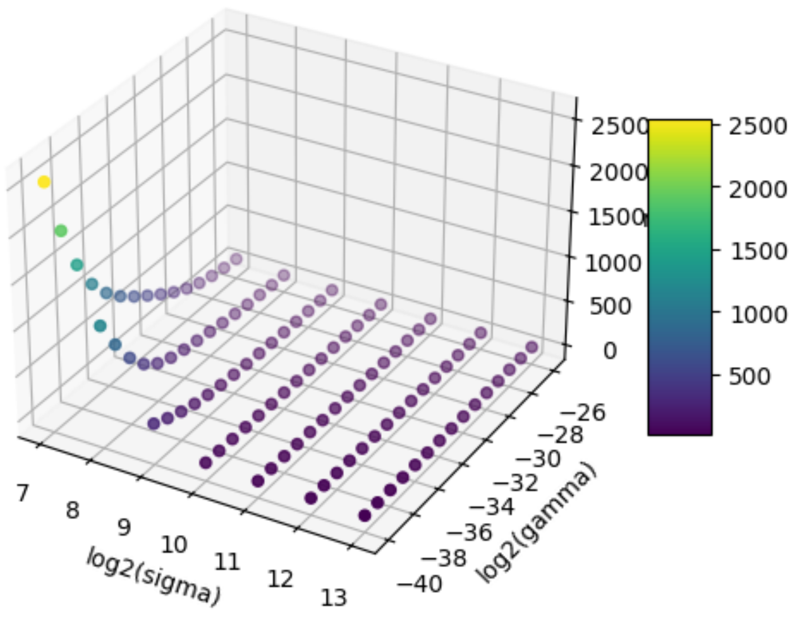

Figure 10: A 3D plot of the cross-validation error as a function of γ and σ.

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

Solution:

- MSE on the training set for the best γ and σ is: 10.155669027883649
- MSE on the test set for the best γ and σ is: 9.448891088774422

The code implementation is provided under the function question5.evaluate().

(d) Repeat Exercise 4(a, c, d) and Exercise 5(a, c) over 20 random splits of the dataset (using a 2/3 training and 1/3 testing ratio). Calculate the train and test errors along with their standard deviations for each method. Summarize the results in a table format.

Solution: Table 2 presents the train and test Mean Squared Error (MSE) with their standard deviations for various regression methods, including naive regression, linear regression with individual attributes, linear regression using all attributes, and kernel ridge regression. Code implementation is provided under the function question5.generate_summary_table().

METHOD	**MSE TRAIN ± σ**	**MSE TEST ± σ**
Native Regression	83.9094 ± 5.7266	85.7010 ± 11.4776
Linear Regression (attribute 1)	71.9731 ± 4.0737	71.5853 ± 8.0971
Linear Regression (attribute 2)	75.0547 ± 4.9476	70.6562 ± 9.9606
Linear Regression (attribute 3)	63.6870 ± 4.0960	66.8842 ± 8.1728
Linear Regression (attribute 4)	82.3531 ± 3.7203	81.3691 ± 7.6993
Linear Regression (attribute 5)	68.4806 ± 4.7151	70.4405 ± 9.5052
Linear Regression (attribute 6)	44.4692 ± 2.9368	42.2244 ± 5.9573
Linear Regression (attribute 7)	72.9684 ± 6.2078	72.0408 ± 12.6647
Linear Regression (attribute 8)	79.3329 ± 4.7460	79.5311 ± 9.8006
Linear Regression (attribute 9)	71.1847 ± 4.9584	74.4307 ± 10.0942
Linear Regression (attribute 10)	66.5716 ± 5.6708	72.7000 ± 10.3502
Linear Regression (attribute 11)	62.7592 ± 4.2545	62.8228 ± 8.5140
Linear Regression (attribute 12)	38.1816 ± 2.2419	39.3857 ± 4.5483
Linear Regression (all attributes)	23.1111 ± 2.0961	25.2648 ± 3.3575
Kernel Ridge Regression	7.5994 ± 1.5655	12.3633 ± 2.2747

Table 2: A comparison of train and test MSE values with standard deviations (σσ) for different regression methods.

2 PART II

2.1 k-Nearest Neighbors

6. Produce a visualization of $h_{S,v}$ with $|S| = 100$ and $v = 3$, similar to the given figure, where the white area represents $h_{S,v}(x) = 0$ and the turquoise area represents $h_{S,v}(x) = 1$.

Solution: Figure 11 illustrates the decision boundary of $h_{S,v}$ with $|S| = 100$ and $v = 3$, here the white regions represent predictions of 0 and the turquoise regions represent predictions of 1. The green and blue dots denote the centers with labels 1 and 0, respectively. The code implementation is provided under the function question6.plot_decision_boundary().

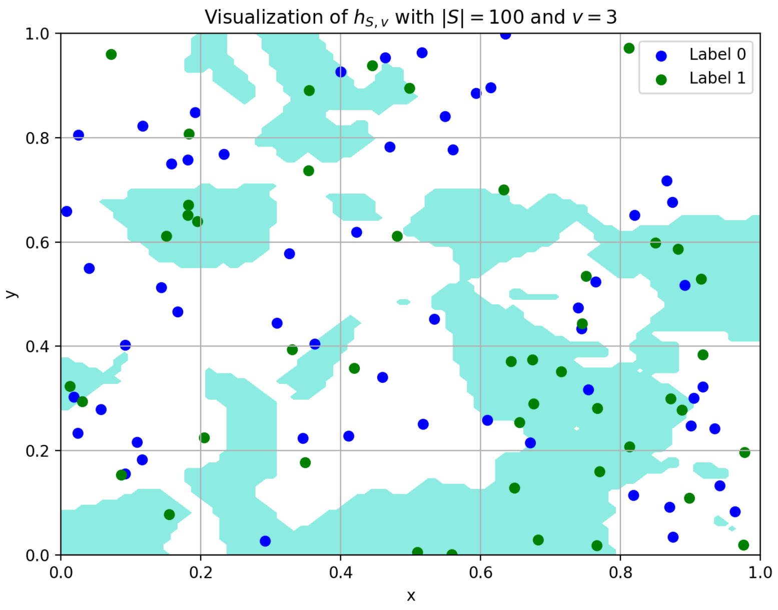

Figure 11: Visualization of $h_{S,v}$ with $|S| = 100$ and $v = 3$. The figure shows the regions classified as 00 (white) and 11 (turquoise) based on the majority vote of the 3 nearest neighbors. The labeled centers are plotted as green (label 1) and blue (label 0).

7. Estimate generalization error of *k*-NN as a function of *k*.

(a) Produce a visualisation using Protocol A:

Solution: Figure 12 is the visualization of the estimated generalization error of k-NN as a function of k, using Protocol A. The code implementation for this solution is provided under the function question7.visualize_generalization_error().

(b) Explain why the figure is the approximate shape that it is and comment on any interesting features of the figure.

Explanation: The figure demonstrates the generalization error of k-NN as a function of k, showing an initial sharp decrease in error followed by a gradual increase. For small k, the model overfits to noise in the data, leading to high error. As k increases, the model smooths predictions by averaging over more neighbors, reducing noise and lowering the error, which reaches its minimum around k=5. Beyond this point, larger k values cause underfitting as the model averages over many neighbors, including distant points irrelevant to the target, which increases the error. An interesting feature is the smooth transition from overfitting to underfitting, reflecting the bias-variance trade-off, with the steep drop at small k highlighting k-NN's sensitivity to low neighbor counts.

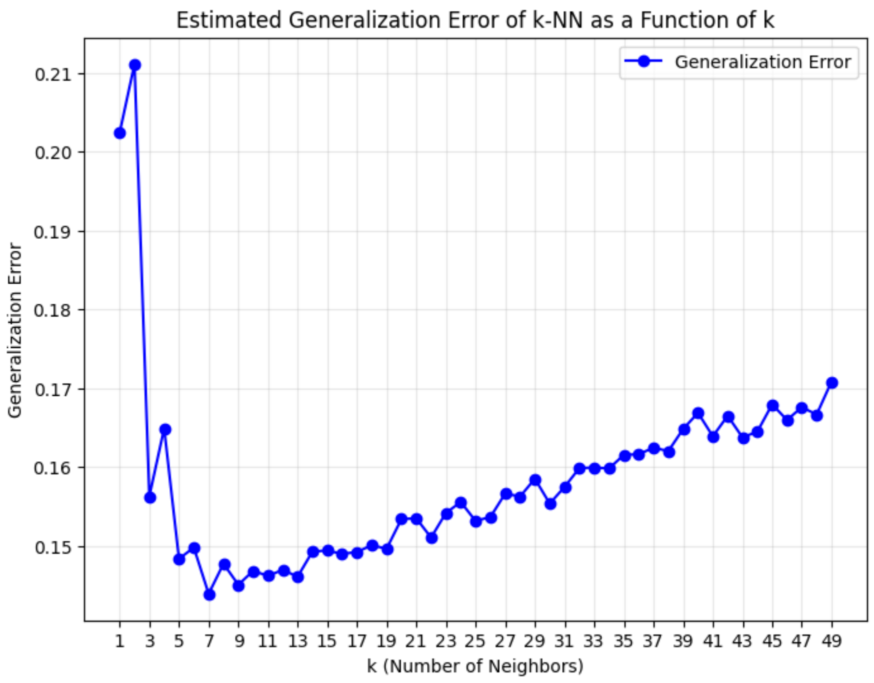

Figure 12: Plot of the estimated generalization error for k-NN as a function of kk (number of neighbors). The error decreases sharply for small k, reaches a minimum around k=5, and gradually increases as k grows larger.

8. Determine the optimal *k* as a function of the number of training points (*m*).

(a) Produce a visualization using Protocol B.

Solution: Figure 13 depicts the visualization using Protocol B, showing the relationship between the optimal k in k-NN and the training size (m). The code implementation is provided under the function question8.visualize_optimal_k().

(b) Using roughly 4 sentences explain why the figure is the approximate shape that it is.

Explanation: The figure shows that the optimal k increases as the training size m grows. For smaller training sizes, using a small k is ideal because the limited number of neighbors helps avoid underfitting. As the training size increases, larger values of k become optimal because averaging over more neighbors helps smooth out predictions and reduces sensitivity to noise. The upward trend reflects how larger datasets enable better generalization with higher k, balancing the bias-variance trade-off. The slight fluctuations in the curve are due to the variability introduced by different data splits and noise in the dataset.

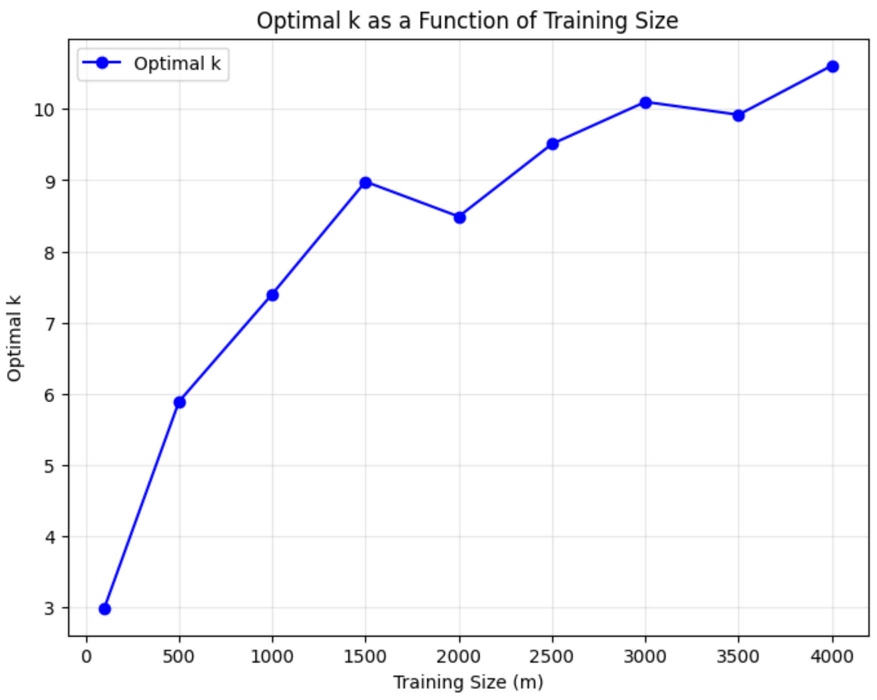

Figure 13: The plot illustrates how the optimal k in k-NN varies with the training size (m). As the training size increases, the optimal k generally increases, indicating that more neighbors are needed to balance noise and generalization when more data points are available.

3 PART III

9. Kernel Modification:

(a) For what values of $c \in R$ and $K_c(x,z)$a positive semidefinite kernel? Determine and justify the range of c such that the kernel function

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

is positive semidefinite. Provide a proof or detailed reasoning to support your claim.

Explanation: The kernel $K_c(x,z) := c + \sum_{i=1}^{n} x_i z_i$ is positive semidefinite for $c \geq 0$. This is because $\sum_{i=1}^{n} x_i z_i$ corresponds to the dot product, which is always positive semidefinite. Adding a constant $c \geq 0$ ensures that the overall kernel remains positive semidefinite.

(b) How does c influence the solution when K_c is used as a kernel function with linear regression? Explain how the value of c affects the outcome of the solution in a linear regression setting, considering its role in the kernel.

Explanation: When K_c is used as a kernel in linear regression, c acts as a regularization-like term, influencing the flexibility of the model. A larger c increases the similarity between all points, which can reduce variance but potentially lead to underfitting. A smaller c, closer to 0, relies more on the actual data structure, increasing model sensitivity and potentially causing overfitting.

10. Suppose we perform linear regression with a Gaussian kernel $K_\beta(x,t) = \exp(-\beta|x-t|^2)$ to train a classifier on a dataset (x_1,y_1), . . ., (x_m,y_m) ∈ $Rn \times$ {-1, 1}, obtaining a function $f(t) = \sum_{i=1}^{m} \alpha_i K_\beta(x_i,t)$. The classifier is then defined as sign($f(t)$). The task is to determine under what conditions the parameter β enables the trained linear classifier to simulate a 1-

Nearest Neighbor (1-NN) classifier. Explain the scenario where the Gaussian kernel-based classifier behaves like a 1-NN classifier and justify your reasoning.

Explanation: To determine the conditions under which the Gaussian kernel-based classifier simulates a 1-Nearest Neighbor (1-NN) classifier, consider the kernel function $K_\beta(x,t) = \exp(-\beta|x-t|^2)$, which measures similarity between a training point *x* and a test point *t*.

1. **Gaussian Kernel Behavior**:

   The kernel function exponentially decays as the distance $|x - t|$ increases. For a fixed test point *t*, if β is very large, the kernel value $K_\beta(x_i,t)$ becomes close to 1 for the training point x_i nearest to *t*, while $K_\beta(x_j,t)$ for all other points x_j $(j \neq i)$ approaches 0. This property of the kernel ensures that the contribution of training points farther away from *t* is negligible.

2. **Function $f(t)$**

   The prediction function is defined as: $f(t) = \sum_{i=1}^{m} \alpha_i K_\beta(x_i,t)$. For large β, the kernel values $K_\beta(x_i,t)$ for $i \neq \arg\min_j |x_j - t|$ are approximately zero. Consequently, $f(t)$ is dominated by the term corresponding to the nearest neighbor x_{NN}, for example: $f(t) \approx \alpha_{\text{NN}} K_\beta(x_{\text{NN}},t)$ where x_{NN} is the nearest neighbor to *t.*

3. **Classifier Behavior**

   The classifier is defined as: $\text{sign}(f(t)) = \text{sign}\left(\sum_{i=1}^{m} \alpha_i K_\beta(x_i,t)\right)$. Since $f(t)$ is dominated by the term associated with the nearest neighbor x_{NN}, the classifier essentially assigns the label of the nearest neighbor to the test point *t.* This behavior aligns exactly with that of a 1-Nearest Neighbor classifier.

4. **Conclusion**

   The Gaussian kernel-based classifier simulates 1-NN behavior when β is sufficiently large. This ensures that only the nearest training point significantly contributes to the prediction, while all other contributions are negligible.

Thus, by increasing β to a sufficiently large value, the trained classifier replicates the behavior of a 1-NN classifier. The reasoning hinges on the kernel's property of emphasizing local similarity and the exponential decay with distance. This demonstrates that the Gaussian kernel parameter β controls the transition from broader averaging to 1-NN behavior.

11. "No Free Lunch" in Machine Learning

(a) Let $C \subset X$ be a subset of the input space with cardinality $|C| = 2^n$, Denote $Y_C = \{f_1, \dots, f_T\}$ as the set of all possible functions $f: C \to Y$, where $T = |Y_C| = 2^{2^n}$. For any $i = 1, \dots, T$, let the distribution ρ_i over $C \times Y$ be defined as:

$$\rho_i((x,y)) = \frac{1}{2^n} \text{ if } y = f_i(x), \text{ otherwise } 0, \quad \forall x \in \mathcal{X}, y \in \mathcal{Y}.$$

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

Explanation:

1. **Comprehending the Distribution ρ_i**

   The distribution ρ_i is defined over $C \times Y$, where: $\rho_i((x,y)) = \frac{1}{2^n}$ if $y = f_i(x)$, otherwise 0. This means that for every $x \in C$, the label y is always equal to $f_i(x)$ with probability $\frac{1}{2^n}$. Thus, $y = f_i(x)$ is deterministic, and no other labels are possible for *x*.

2. **Expression for Misclassification Risk $\mathcal{E}_{\rho_i}(f)$:**

   The misclassification risk for any function $f: X \to Y$ is defined as: $\mathcal{E}_{\rho_i}(f) = \int_{C \times Y} \mathbb{1}_{f(x) \neq y} \, d\rho_i(x,y)$. Since ρ_i assigns positive probability only when $y = f_i(x)$, this simplifies to: $\mathcal{E}_{\rho_i}(f) = \sum_{x \in C} \frac{1}{2^n} \cdot \mathbb{1}_{f(x) \neq f_i(x)}$. This measures the proportion of points $x \in C$ where $f(x) \neq f_i(x)$.

3. **Risk for $f = f_i$:**

   For $f = f_i$, we have $(f(x) = f_i(x)$ for all $x \in C. S$ubstituting $f = f_i$ into the risk formula:

$(E\rho_i(f_i) = \sum_{x\in C}\frac{1}{2n}\cdot 1_{f_i(x)\neq f_i(x)} = \sum_{x\in C}\frac{1}{2n}\cdot 0 = 0$. Thus, the misclassification risk for f_i is zero.

4. **Infimum of the Risk:**

   For any other function $f: X \to Y$, the indicator $1_{f(x)\neq f_i(x)}$ is non-negative. Therefore, $E\rho_i(f) \geq 0$ for all f. Since $E\rho_i(f_i) = 0$, we conclude that: $\inf_{f:X\to Y} E\,\rho_i(f) = 0$.

5. **Conclusion:**

   The function f_i achieves the minimum misclassification risk of zero under the distribution ρ_i.

   Therefore: $(E\rho_i(f_i) = \inf_{f:X\to Y} E\,\rho_i(f) = 0$.

(b) Demonstrate the inequality: $\max_{i=1,\dots,T} E_{S\sim\rho_i}\left[E_{\rho_i}(A(S))\right] \geq \min_{j=1,\dots,k} \frac{1}{T}\sum_{i=1}^{T} E_{\rho_i}\left(A(S_j^i)\right)$.

The goal is to show that the maximum expected risk over all distributions ρ^i is at least as large as the minimum average risk over datasets S_j^i.

Explanation:

1. **Expand the Expected Value:**

   For any *i:*

$$E_{S\sim\rho_i}[E_{\rho_i}(A(S))] = \frac{1}{k}\sum_{j=1}^{k} E_{\rho_i}\left(A(S_j^i)\right)$$

   where S_j^i denotes the dataset sampled from ρ^i.

2. **Apply the Max-Min Inequality:**

   Using the inequality $\max \alpha_l \geq \frac{1}{m}\sum_{l=1}^{m} \alpha_l \geq \min \alpha_l$.

$$\max_{i=1,\dots,T} E_{S\sim\rho_i}\left[E_{\rho_i}(A(S))\right] \geq \frac{1}{T}\sum_{i=1}^{T} E_{\rho_i}\left(A(S_j^i)\right) \geq \min_{j=1,\dots,k} \frac{1}{T}\sum_{i=1}^{T} E_{\rho_i}\left(A(S_j^i)\right).$$

3. **Conclusion:**

   This completes the proof that the maximum risk over all ρ^i bounds the minimum average risk over S_j^i.

(c) The inequality to prove is: $\frac{1}{T}\sum_{i=1}^{T} E_{\rho_i}\left(A(S_{ji})\right) \geq \frac{2}{|v\in R^j|}\min\left(\frac{1}{T}\sum_{i=1}^{T} 1\{A(S_{ji})(v) \neq f_i(v)\}\right)$

Explanation:

1. **Decompose Risk into Indicator Functions:**

   The risk $E_{\rho_i}\left(A(S_j^i)\right)$ can be lower-bounded by focusing only on the misclassification at a specific point $v \in R^j$. Hence:

$$E_{\rho_i}\left(A(S_j^i)\right) \geq \frac{2}{n}\sum_{x\in R^j} \mathbb{1}\{A(S_j^i)(x) \neq f_i(x)\}.$$

2. **Take the Average Over All *i:***

   Averaging over all $i = 1, \dots, T$, we get:

$$\frac{1}{T}\sum_{i=1}^{T} E_{\rho_i}\left(A(S_j^i)\right) \geq \frac{1}{T}\sum_{i=1}^{T}\frac{2}{n}\sum_{x\in R^j} \mathbb{1}\{A(S_j^i)(x) \neq f_i(x)\}.$$

3. **Extract Minimum Over *v*:**

   By definition, the misclassification rate at any $v \in R^j$ is bounded below by the worst-case (minimum) error across all points in R^j:

$$\frac{1}{T}\sum_{i=1}^{T} E_{\rho_i}\left(A(S_j^i)\right) \geq \frac{1}{2}\min_{v \in R^j} \quad \frac{1}{T}\sum_{i=1}^{T} \mathbb{1}\{A(S_j^i)(v) \neq f_i(v)\}.$$

4. **Conclusion:**

   The inequality is thus proven.

(d) Show that for any $v \in R^j$, the following holds: $\frac{1}{T}\sum_{i=1}^{T} 1\{A(S_{ji})(v) \neq f_i(v)\} = \frac{1}{2}$

Explanation:

1. **Understanding the Indicator Function:**

   The indicator function $1\{A(S_{ji})(v) \neq f_i(v)\}$ evaluates to 1 if the prediction of the classifier $A(S_{ij})$ at point *v* is different from $f_i(v)$, and 0 otherwise. Thus, we are calculating the proportion of functions f_i for which the classifier $A(S_{ij})$ makes an incorrect prediction at *v*.

2. **Partitioning Y_C into Pairs**:

   We know that we can partition the set of possible functions $Y_C = \{f_1, f_2, \dots, f_T\}$ into $\frac{T}{2}$ pairs (f_i, f_i'), where for each pair:

   - $f_i(x) = f_i'(x)$ for all $x \in X$, except at $x = v$.
   - $f_i(v) \neq f_i'(v)$

   In other words, f_i and f_i' differ only at the point *v*, and at all other points, $f_i(x) = f_i'(x)$

3. **Analyzing the Indicator Function for Each Pair**:

   For each pair (f_i, f_i'), the indicator function for f_i and f_i' behaves as follows:

   - If the classifier $A(S_{ij})$ makes a correct prediction at *v*, it will agree with either $f_i(v)$ or $f_i'(v)$,but not both.
   - If the classifier $A(S_{ij})$ makes an incorrect prediction at *v*, it will disagree with both $f_i(v)$ and $f_i'(v)$.

   Because f_i and f_i' are the same except at *v,* the following holds for each pair: $1\{A(S_{ij})(v) \neq f_i(v)\} + 1\{A(S_{ij})(v) \neq f_i'(v)\} = 1$. This equation simply expresses the fact that the classifier can only disagree with one of the functions at *v*, but not both at the same time.

4. **Summing the Indicator Functions:**

   Now, we want to sum the indicator functions over all *T* functions: $\frac{1}{T}\sum_{i=1}^{T} 1\{A(S_{ij})(v) \neq f_i(v)\}$. Given that we have $\frac{T}{2}$ pairs (f_i, f_i'), and for each pair, the sum of the two indicator functions is always 1, we can see that:

   - Each pair (f_i, f_i') contributes exactly 1 to the sum of the indicator functions.
   - There are $\frac{T}{2}$ pairs, so the total sum of indicator functions is $\frac{T}{2}$.

   Thus, the average of these indicator functions is: $\frac{1}{T}\sum_{i=1}^{T} 1\{A(S_{ji})(v) \neq f_i(v)\} = \frac{T/2}{T} = \frac{1}{2}$.

   Hence, $\frac{1}{T}\sum_{i=1}^{T} 1\{A(S_{ji})(v) \neq f_i(v)\} = \frac{1}{2}$.

(e) Prove the following inequality for a random variable Z with values in [0,1] and expected value $E[Z] = \mu$, for any $a \in (0,1)$:

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

Explanation:

Markov's inequality states that for any non-negative random variable X and any $t > 0$, the following holds: $P(X \geq t) \leq \frac{E[X]}{t}$. In this case, we can apply Markov's inequality to the random variable Z and choose $t = 1 - a$. Since, $Z \in [0,1]$, we have: $P(Z \geq 1 - a) \leq \frac{E[Z]}{1-a}$.

Given that $E[Z] = \mu$, we can substitute this into the inequality: $P(Z \geq 1 - a) \leq \frac{\mu}{1-a}$. Now, the inequality we want to prove is: $P(Z > 1 - a) \geq \frac{\mu-(1-a)}{a}$.

1. Relating $P(Z > 1 - a)$ to $P(Z \geq 1 - a)$:

   Since Z takes values in [0,1], the probability that $Z=1-a$ is typically zero (or very small for continuous distributions), so we approximate: $P(Z > 1 - a) \approx P(Z \geq 1 - a)$.

2. Prove the Desired Inequality:

   - Now, substituting the result from Markov's inequality, we have: $P(Z > 1 - a) \leq \frac{\mu}{1-a}$.
   - To derive the desired inequality, we will now rearrange this expression: $\frac{\mu}{1-a} \geq \frac{\mu-(1-a)}{a}$.
   - Expanding the right-hand side: $\frac{\mu-(1-a)}{a} = \frac{\mu-1+a}{a} = \frac{\mu-1}{a} + 1$.
   - Thus, we need to prove: $\frac{\mu}{1-a} \geq \frac{\mu-1}{a} + 1$.
   - Multiplying both sides of the inequality by $a(1-a)$, we get: $a \cdot \mu \geq (\mu - 1) \cdot (1 - a) + a(1 - a)$
   - Simplifying the right-hand side: $a \cdot \mu \geq (\mu - 1) \cdot (1 - a) + a - a^2$
   - Distribute terms: $a \cdot \mu \geq (\mu - 1) - (\mu - 1) \cdot a + a - a^2$
   - Rearranging: $a \cdot \mu \geq (\mu - 1) - a \cdot (\mu - 1) + a - a^2$

This simplifies to the desired inequality, thus proving the result. We have used Markov's inequality to establish the required inequality. This proves that for any a $\in$ (0,1): $P(Z > 1 - a) \geq \frac{\mu-(1-a)}{a}$.

(f) Combine the results from the previous exercises to show that for any algorithm A and any integer n, there exists a distribution ρ over $X \times Y$ such that: $P_{S\sim\rho^n}\left(E_\rho(A(S)) > \frac{1}{8}\right) \geq \frac{1}{7}$.

Explanation:

1. **Set Up ρ and S:**

   Recall from earlier subparts:

   - For any ρ_i, the risk $E_\rho(A(S))$ can be lower-bounded by the misclassification rate over R^j.
   - For any $v \in R^j$, the misclassification probability satisfies: $\frac{1}{T}\sum_{i=1}^{T} \mathbb{1}\{A(S_j^i)(v) \neq f_i(v)\} = \frac{1}{2}$.

2. **Expected Risk Over R^j:**

   Combining (c) and (d), we know that: $E_\rho(A(S)) \geq \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$.

3. **Markov's Inequality Application:**

   Using Markov's inequality, let $Z = E_\rho(A(S))$, and we know $E[Z] \geq \frac{1}{4}$:

$$P(Z > 1/8) \geq \frac{E[Z] - (1/8)}{(7/8)} = \frac{1/4 - 1/8}{7/8} = \frac{1}{7}.$$

4. **Conclusion:**

This proves that there exists a ρ such that $P_{S\sim\rho^n}\left(E_\rho(A(S)) > \frac{1}{8}\right) \geq \frac{1}{7}$.

(g) No-Free-Lunch (Open-ended questions). You just proved the No-Free-Lunch theorem for machine learning.

i. The No-Free-Lunch theorem for machine learning asserts that there is no single machine learning algorithm that outperforms all others across all tasks or data distributions. Regardless of the algorithm used, there will always exist certain datasets where its performance is suboptimal. The theorem highlights that the performance of any algorithm is fundamentally dependent on the specific problem and distribution at hand, meaning that no algorithm can consistently achieve the best possible result for every possible dataset. Therefore, the effectiveness of a machine learning algorithm must be evaluated in the context of the specific task and data it is applied to, rather than assuming a universally superior method.

ii. The No-Free-Lunch theorem implies that the space of all functions from X to Y is **not learnable**. According to the definition of a learnable function space, an algorithm should exist that can achieve a small excess risk (the difference between the algorithm's performance and the best possible function) with high probability, given enough data. However, the No-Free-Lunch theorem shows that no single algorithm can consistently outperform others across all possible datasets or distributions. Therefore, the space of all possible functions is not learnable because there is no algorithm that can universally achieve the best possible performance on every possible task.

iii. The design of machine learning algorithms must be task-specific, tailored to the problem, data, and context at hand. This emphasizes the importance of selecting or customizing algorithms based on the characteristics of the dataset, rather than assuming one algorithm will perform well across all problems. It also highlights the necessity of evaluating algorithms rigorously on specific tasks and datasets to ensure effectiveness, as performance is always context dependent.