Supervised Learning (COMP0078) – Coursework 1 Version 2

Due : 20 November 2024

Submission

You may work individually or in a group of two. When working in a group, both students will receive the same grade. You/your group should produce a pdf with the answer to every question/task below, code should be submitted separately in a zip file (not printed in the pdf). You will not only be assessed on the **correctness/quality** of your answers but also on **clarity of presentation**. Additionally make sure that your code is *well commented*.

Please submit on Moodle:

1. your answers as a pdf file,
2. a zip file with your source code.

Regarding the use of libraries/packages for your code, you should implement regression (with and without basis functions, kernels) using matrix algebra directly and k-NN directly (not calling a function from e.g. scikit-learn to do that for you). Otherwise, libraries are okay in general.

Questions please e-mail `sl-support@cs.ucl.ac.uk` or the moodle *Questions* forum.

1 PART I [50%]

1.1 Linear Regression

Linear regression overview: Given a set of data:

$$\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\} \tag{1}$$

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

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

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

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

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

$$(X\boldsymbol{w} - \boldsymbol{y})^\intercal (X\boldsymbol{w} - \boldsymbol{y}).$$

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 $\boldsymbol{\phi} : \mathbb{R}^n \to \mathbb{R}^k$

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

We use the basis functions to transform the data as follows

$$\{((\phi_1(\boldsymbol{x}_1),\ldots,\phi_k(\boldsymbol{x}_1)),y_1),\ldots,((\phi_1(\boldsymbol{x}_m),\ldots,\phi_k(\boldsymbol{x}_m)),y_m)\}, \tag{4}$$

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

$$\Phi := \begin{pmatrix} \boldsymbol{\phi}(\boldsymbol{x}_1) \\ \vdots \\ \boldsymbol{\phi}(\boldsymbol{x}_m) \end{pmatrix} = \begin{pmatrix} \phi_1(\boldsymbol{x}_1) & \ldots & \phi_k(\boldsymbol{x}_1) \\ \vdots & \ddots & \vdots \\ \phi_1(\boldsymbol{x}_m) & \ldots & \phi_k(\boldsymbol{x}_m) \end{pmatrix}, \quad (m \times k) \tag{5}$$

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

$$(\Phi\boldsymbol{w} - y)^T(\Phi\boldsymbol{w} - y) = \sum_{t=1}^{m}(y_t - \sum_{i=1}^{k} w_i\phi_i(\boldsymbol{x}_t))^2 \tag{6}$$

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.

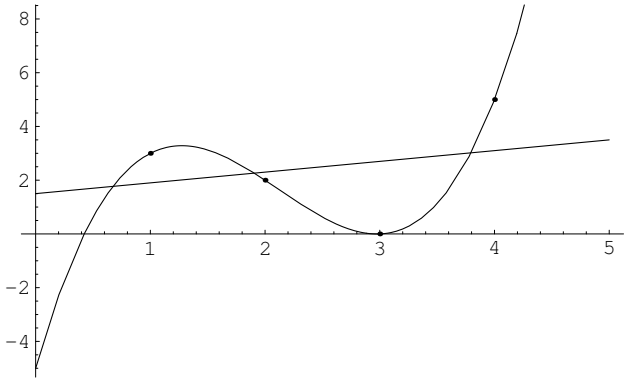


Figure 1: Data set $\{(1,3),(2,2),(3,0),(4,5)\}$ fitted with basis $\{1,x\}$ and basis $\{1,x,x^2,x^3\}$

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.

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

   (c) For each fitted curve $k = 1,2,3,4$ give the mean square error where MSE $= \frac{\text{SSE}}{m}$.

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

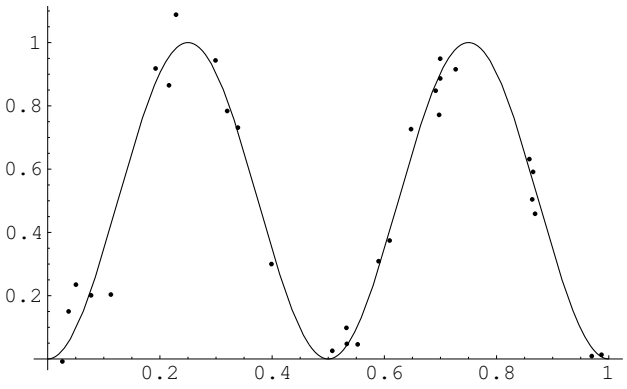
   (a) Define

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

   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 $0 \le x \le 1$ with the points of the above data set superimposed. The plot should resemble



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.[1]

(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).

(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.

(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 a 100 runs (note: plot the ln(avg) rather than the avg(ln)).

3. **[5 pts]:** Now use 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.

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 a 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 data set 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:

[1] Depending on you implementation you may have numerical errors for large values of k this is 'ok' for the purposes of this exercise.

(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.

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

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, $(\boldsymbol{x}_i, 1)$, so that we learn a weight vector $\boldsymbol{w} \in \Re^2$. For each of the twelve regressions, calculate the MSE on the training and test sets and note down the results.

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.

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,

$$\boldsymbol{w}^* = \text{argmin}_{\boldsymbol{w}\in\mathbb{R}^n} \frac{1}{\ell}\sum_{i=1}^{\ell}(\boldsymbol{x}_i^\intercal \boldsymbol{w} - y_i)^2 + \gamma \boldsymbol{w}^\intercal \boldsymbol{w}. \tag{10}$$

i.e, we have replace the sum of the square errors with the mean square error (MSE).[2] For a given kernel function K define the kernel matrix $\boldsymbol{K}$ for a training set of size ℓ elementwise via

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

The dual optimisation formulation after kernelization is

$$\boldsymbol{\alpha}^* = \text{argmin}_{\boldsymbol{\alpha}\in\mathbb{R}^\ell} \frac{1}{\ell}\sum_{i=1}^{\ell}(\sum_{j=1}^{\ell}\alpha_j K_{i,j} - y_i)^2 + \gamma \boldsymbol{\alpha}^\intercal \boldsymbol{K}\boldsymbol{\alpha}. \tag{11}$$

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

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

[2]The motivation is that we wish the regularisation parameter γ to 'scale' somewhat independently of training set size.

where $\mathbf{I}_\ell$ denotes the $\ell \times \ell$ identity matrix. The evaluation of the regression function on a test point can be reformulated as:

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

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 data set using the Gaussian kernel,

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

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 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.

   b. Plot the "cross-validation error" (mean over folds of 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.

   d. Repeat "exercise 4a,c,d" 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.

Method	**MSE train**	**MSE test**
Naive Regression	??.?? $\pm$ σ'?	??.?? $\pm$ σ'?
Linear Regression (attribute 1)	??.?? $\pm$ σ'?	??.?? $\pm$ σ'?
Linear Regression (attribute 2)	??.?? $\pm$ σ'?	??.?? $\pm$ σ'?
Linear Regression (attribute 3)	??.?? $\pm$ σ'?	??.?? $\pm$ σ'?
Linear Regression (attribute 4)	??.?? $\pm$ σ'?	??.?? $\pm$ σ'?
Linear Regression (attribute 5)	??.?? $\pm$ σ'?	??.?? $\pm$ σ'?
Linear Regression (attribute 6)	??.?? $\pm$ σ'?	??.?? $\pm$ σ'?
Linear Regression (attribute 7)	??.?? $\pm$ σ'?	??.?? $\pm$ σ'?
Linear Regression (attribute 8)	??.?? $\pm$ σ'?	??.?? $\pm$ σ'?
Linear Regression (attribute 9)	??.?? $\pm$ σ'?	??.?? $\pm$ σ'?
Linear Regression (attribute 10)	??.?? $\pm$ σ'?	??.?? $\pm$ σ'?
Linear Regression (attribute 11)	??.?? $\pm$ σ'?	??.?? $\pm$ σ'?
Linear Regression (attribute 12)	??.?? $\pm$ σ'?	??.?? $\pm$ σ'?
Linear Regression (all attributes)	??.?? $\pm$ σ'?	??.?? $\pm$ σ'?
Kernel Ridge Regression	??.?? $\pm$ σ'?	??.?? $\pm$ σ'?

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.

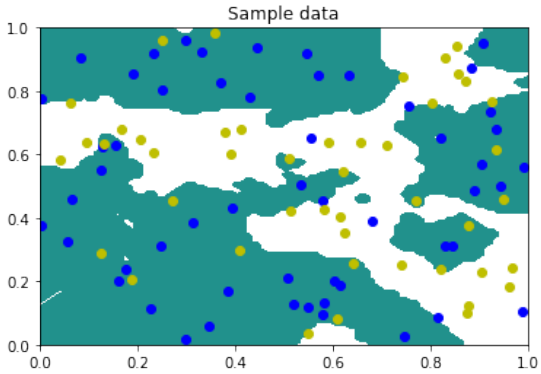


Figure 1: A hypothesis $h_{S,v}$ visualized with $|S| = 100$ and $v = 3$.

2.1.1 Generating the data

A voted-center hypothesis is a function $h_{S,v} : [0,1]^2 \to \{0, 1, \square\}$ where $S = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_{|S|}, y_{|S|})\}$ is a set of labeled centers with each $\boldsymbol{x}_i \in [0,1]^2$ and $y_i \in \{0,1\}$ and v is a positive integer. In this coursework we will always set $v := 3$ for all of of the exercises in this subsection. Where $h_{S,v}(\boldsymbol{x}) = 0$ if the majority of the v nearest $\boldsymbol{x}_i$'s to $\boldsymbol{x}$ in S are '0', similarly for $h_{S,v}(\boldsymbol{x}) = 1$ and finally in certain "corner" cases the "the majority of v ..." will not be well-defined and in that case $h_{S,v}(\boldsymbol{x}) = \square$. See Section 2.1.4 for addressing those corner cases.

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

6. **[5 pts]** *Produce a visualisation 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 $p_{\mathcal{H}}$.

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

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

Generating our noisy data. Given an $h_{S,v}$ the underlying probability distribution $p_h(\boldsymbol{x}, y)$ is determined by sampling an $\boldsymbol{x}$ uniformly at random from $[0,1]^2$ and then its corresponding y value is then generated by flipping biased coin $P(\text{heads}) = 0.8/P(\text{tails}) = 0.2$ if heads comes up then $y = h_{S,3}(\boldsymbol{x})$ otherwise y is sampled uniformly at random from $\{0,1\}$. When you generate training and test sets they will both come from this distribution.

7. **[7 pts]** a) Produce a visualisation using Protocol A (see Figure 2). b) Using roughly 5 sentences explain why the figure is the approximate shape that it is and comment on any interesting features of the figure.

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. **[8 pts]** a) Produce a visualisation using Protocol B (see Figure 3). I.e, plotting m versus optimal k b) Using roughly 4 sentences explain why the figure is the approximate shape that it is.

```
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 2: 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 3: 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.

2.1.4 Additional Clarifications

Note both the k-NN algorithm and the hypothesis h can produced "undefined" results in certain corner cases for individual $\boldsymbol{x}$'s. Resolve these case by generating either the label or prediction uniformly at random.

3 PART III [30%]

3.1 Questions

Notation: We overload $[\cdot]$ as follows $[n] := \{1, 2, \ldots, n\}$ if n is a positive integer and $[\texttt{pred}] = 1$ if `pred` is a logical predicate which is true and $[\texttt{pred}] = 0$ otherwise.

9. **[5 pts]** *Kernel modification* Consider the function $K_c(\boldsymbol{x}, \mathbf{z}) := c + \sum_{i=1}^{n} x_i z_i$ where $\boldsymbol{x}$, $\mathbf{z} \in \mathbb{R}^n$.

   (a) For what values of $c \in \mathbb{R}$ is K_c a positive semidefinite kernel? Give an argument supporting your claim. ("The closer your argument is to a proof the more likely it is to receive full credit.")

   (b) Suppose we use K_c as a kernel function with linear regression (least squares). Explain how c influences the solution.

10. **[10 pts]** Suppose we perform linear regression with a Gaussian kernel $K_\beta(\boldsymbol{x}, \mathbf{t}) = \exp(-\beta\|\boldsymbol{x} - \mathbf{t}\|^2)$ to train a classifier on a dataset $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m) \in \mathbb{R}^n \times \{-1, 1\}$. Thus obtaining a function $f : \mathbb{R}^n \to \mathbb{R}$ which is of the form $f(\mathbf{t}) = \sum_{i=1}^{m} \alpha_i K_\beta(\boldsymbol{x}_i, \mathbf{t})$. The corresponding classifier is then $\text{sign}(f(\mathbf{t}))$. This classifier depends on the parameter β selected for the kernel. In what scenario will chosen β enable the trained linear classifier to simulate a 1-Nearest Neighbor classifier trained on the same dataset?

Note $\beta \in \mathbb{R}$ is a fixed scalar which may be selected as a function of $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m$ and the test point $\mathbf{t}$, i.e., we may use a function so that $\beta = \hat{\beta}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m, \mathbf{t})$ an exact function $\hat{\beta}(\cdot)$ need not be given just an argument that one exists.

Give an argument supporting your reasoning. ("The closer your argument is to a proof the more likely it is to receive full credit.")

11. **[15 pts]** We will show that there is no *"free lunch"* in machine learning, namely that without making assumptions on the learning setting, we cannot provide guarantees on how good an algorithm can be at solving a learning problem.

    We consider a binary classification setting with $\mathcal{X}$ and $\mathcal{Y} = \{-1, 1\}$ the input and output sets. For any probability distribution ρ on $\mathcal{X} \times \mathcal{Y}$ and any function $f : \mathcal{X} \to \mathcal{Y}$, we denote

    $$\mathcal{E}_\rho(f) = \int_{\mathcal{X}\times\mathcal{Y}} \mathbf{1}_{\{f(x)\neq y\}} \, d\rho(x, y)$$

    the misclassification excess risk with respect to ρ. Here $\mathbf{1}_{\{f(x)\neq y\}}$ is equal to 0 when $f(x) = y$ and 1 otherwise. In the following, always assume $\mathcal{X}$ to have infinite cardinality (i.e. infinite number of elements). Denote $S = (x_i, y_i)_{i=1}^n$ a dataset of n input-output pairs. In the following you will be showing that, for any algorithm $A : S \mapsto (f : \mathcal{X} \to \mathcal{Y})$ and for any integer $n \in \mathbb{N}$, there exists a distribution ρ such that

    - $\inf_{f:\mathcal{X}\to\mathcal{Y}} \ \mathcal{E}_\rho(f) = 0$
    - $\mathbb{P}_{S\sim\rho^n}\left(\mathcal{E}_\rho(A(S)) > \frac{1}{8}\right) \geq \frac{1}{7}$

    where $\mathbb{P}_{S\sim\rho^n}$ (and also $\mathbb{E}_{S\sim\rho^n}$ in the following) denotes the probability (resp. expectation) with respect to the random variable $S = (x_i, y_i)_{i=1}^n$ with each pair (x_i, y_i) independently sampled from ρ. We organized this proof along multiple steps (if you find you are not able to prove one step, don't get stuck! Go to the next step and assume the previous one to be true). Give an argument supporting your reasoning. ("The closer your argument is to a proof the more likely it is to receive full credit."):

    (a) Let $C \subset \mathcal{X}$ be a subset of $\mathcal{X}$ with cardinality (i.e. number of elements) $|C| = 2n$. Denote with $\mathcal{Y}^C = \{f_1, \ldots, f_T\}$ the set of all possible functions $f : C \to \mathcal{Y}$. We have $T = |\mathcal{Y}^C| = 2^{2n}$. For any $i = 1, \ldots, T$ denote with ρ_i the distribution over $C \times \mathcal{Y}$ such that

    $$\rho_i(\{(x, y)\}) = \begin{cases} \frac{1}{2n} & \text{if } y = f_i(x) \\ 0 & \text{otherwise} \end{cases} \qquad \forall x \in \mathcal{X}, y \in \mathcal{Y}$$

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

    (b) Let C^n be the set of all possible *input* datasets (of size n) of points in C. Then $C^n = \{S_1, \ldots, S_k\}$ with $k = |C^n| = (2n)^n$. For every $j = 1, \ldots, k$ and input set $S_j = (x_1, \ldots, x_n) \in C^n$, denote $S_j^i = \{(x_1, f_i(x_1)), \ldots, (x_n, f_i(x_n))\}$ the corresponding input-output dataset for every $i = 1, \ldots, T$. Show that

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

    (Hint: show that for any $i = 1, \ldots, T$

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

    Hint 2: use the fact for any set of scalars $\alpha_1, \ldots, \alpha_m$ we have $\max_\ell \alpha_\ell \geq \frac{1}{m}\sum_{\ell=1}^m \alpha_\ell \geq \min_\ell \alpha_\ell$).

    (c) Fix $j \in \{1, \ldots, k\}$ with $S_j = \{(x_1, \ldots, x_n\}$ and denote $R_j = \{v_1, \ldots, v_p\}$ the subset of points of C that do not belong to S_j. Show that

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

(hint: lower bound the risk with respect to the errors only over S'_j. Use again the fact that $\frac{1}{m}\sum_{\ell=1}^{m}\alpha_m \geq \min_m \alpha_m$)

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

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

(hint: show that for any $v \in R_j$ we can partition $\mathcal{Y}^C$ into $T/2$ pairs $(f_i, f_{i'})$ such that $f_i(x) \neq f_{i'}(x)$ if and only if $x = v$. Show that $\mathbf{1}_{\{A(S_j^i)(v)\neq f_i(v)\}} + \mathbf{1}_{\{A(S_j^{i'})(v)\neq f_{i'}(v)\}} = 1$.)

(e) We prove an auxiliary result: let Z be a random variable with values in $[0,1]$ and let $\mathbb{E}[Z] = \mu$. Show that for any $a \in (0,1)$

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

(hint: use Markov's inequality.)

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

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

(hint: use $Z = \mathcal{E}_\rho(A(S))$ in the previous bound. Use previous results to lower bound $\mathbb{E}_{S\sim\rho_i^n} Z$ for all i).

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

i. Summarize the above results in the formal statement of a theorem.

ii. Consider the definition of *learnable space of function*:
DEFINITION. A space $\mathcal{H}$ of functions $f : \mathcal{X} \to \mathcal{Y}$ is **learnable** if there exists an algorithm A such that, for any $\epsilon, \delta \in (0,1)$ and any distribution ρ over $\mathcal{X}\times\mathcal{Y}$ there exists an integer $\bar{n}$ such that for any $n \geq \bar{n}$

$$\mathbb{P}_{S\sim\rho^n}\left(\mathcal{E}_\rho(A(S)) - \inf_{f\in\mathcal{H}}\mathcal{E}_\rho(f) \leq \epsilon\right) \geq 1-\delta$$

Compare this definition with the results above. Does the No-Free-Lunch theorem imply that the space $\mathcal{Y}^{\mathcal{X}}$ of all functions $f : \mathcal{X} \to \mathcal{Y}$ is (or is not) *learnable*? Why?

iii. Comment on the implications of the No-Free-Lunch theorem with respect to the design of a machine learning algorithm.