Introduction to machine learning
Exercise 1

Fall 2025/26

Submission guidelines, please read and follow carefully:

- The exercise **must** be submitted in pairs.
- Submit via Moodle.
- The submission should include two separate files:
    1. A pdf file that includes your answers to all the questions.
    2. The code files for the python question. You must submit a copy of the shell python file provided for this exercise in Moodle, with the required functions implemented by you. **Do not change the name of this file.** In addition, you can also submit other code files that are used by the shell file.
- Your python code should follow the course python guidelines. See the Moodle website for guidelines and python resources.
- Before you submit, **make sure that your code works in the course environment**, as explained in the guidelines. Specifically, **make sure that the test `simple_test` provided in the shell file works**.
- You may only use python modules that are explicitly allowed in the exercise or in the guidelines. If you are wondering whether you can use another module, ask a question in the exercise forum. No module containing machine learning algorithms will be allowed.
- For questions, use the exercise forum, or if they are not of public interest, send an email to the course staff at intromlbgu26@gmail.com
- Grading: Q.1 (python code): 10 points, Q.2: 20 points, Q.3: 15 points, Q.4: 18 points (+3 bonus points), Q.5: 19 points, Q.6: 18 points
    - Unless otherwise specified, a question's points are evenly distributed among its subsections.

Question 1. Implement a function that runs the k-nearest-neighbors algorithm that we saw in class on a given training sample, and a second function that uses the output classifier of the first function to predict the label of test examples. We will use the Euclidean distance to measure the similarity between examples.

The shell Python file `nearest_neighbour.py` is provided for this exercise in Moodle. It contains empty implementations of the functions required below. You should implement them and submit according to the submission instructions.

The first function, `learnknn`, creates the classification rule. Implement the function in the file which can be found on the assignment page in the course's website. The signature of the function should be:

```
def learnknn(k, x_train, y_train)
```

The input parameters are:

- `k` - the number k to be used in the k-nearest-neighbor algorithm.
- `x_train` - a 2-D matrix of size $m \times d$ (rows $\times$ columns), where m is the sample size and d is the dimension of each example. Row i in this matrix is a vector with d coordinates which specifies example x_i from the training sample.
- `y_train` - a column vector of length m (that is, a matrix of size $m \times 1$). The i-th number in this vector is the label y_i from the training sample. You can assume that each label is an integer between 0 and 9.

The output of this function is `classifier`: a data structure that keeps all the information you need to apply the k-NN prediction rule to new examples. The internal format of this data structure is your choice.

The second function, `predictknn`, uses the classification rule that was outputted by `learnknn` to classify new examples. It should also be implemented in the `nearest_neighbour.py` file. The signature of the function should be:

```
def predictknn(classifier, x_test)
```

The input parameters are:

- `classifier` - the classifier to be used for prediction. Only classifiers that your own code generated using `learnknn` can be used here.
- `x_test` - a 2-D matrix of size $n \times d$, where n is the number of examples to test. Each row in this matrix is a vector with d coordinates that describes one example that the function needs to label.

The output is `y_testprediction`, which is a column vector of length n. Label i in this vector describes the label that `classifier` predicts for the example in row i of the matrix `x_test`.

Important notes:

- You may assume all the input parameters are legal.
- The Euclidean distance between two vectors z_1, z_2 of the same length can be calculated using `numpy.linalg.norm(z1-z2)` or `scipy.spatial.distance.euclidean(z1, z2)`.

Example for using the functions (here there are only two labels, 0 and 1):

```
>>> from nearest_neighbour import learnknn, predictknn
>>> k = 1
>>> x_train = np.array([[1,2], [3,4], [5,6]])
>>> y_train = np.array([1, 0, 1])
>>> classifier = learnknn(k, x_train, y_train)
>>> x_test = np.array([[10,11], [3.1,4.2], [2.9,4.2], [5,6]])
>>> y_testprediction = predictknn(classifier, x_test)
>>> y_testprediction
[1, 0, 0, 1]
```

Question 2. Test your k-nearest-neighbor implementation on the hand-written digits recognition learning problem: In this problem, the examples are images of hand-written digits, and the labels indicate which digit is written in the image. The full dataset of images, called MNIST, is free on the web. It includes 70,000 images with the digits 0-9. A numpy format file that you can use, called `mnist_all.npz`, can be found on the assignment page in the course website. For this exercise, we will use a smaller dataset taken out of MNIST, that only includes images with the digits 1, 3, 4, and 6, so that there are only four possible labels. Each image in MNIST has 28×28 pixels, and each pixel has a value indicating how dark it is. Each example is described by a vector listing the $28 \cdot 28 = 784$ pixel values, so we have $X \in \mathbb{R}^{784}$: every example is described by a 784-coordinate vector.

Figure 1: Some examples of images of digits from the dataset MNIST

The images in MNIST are split into training images and test images. The test images are used to estimate the success of the learning algorithm: In addition to the training sample S as we saw in class, we have a test sample T, which is also a set of labeled examples.
The k-NN algorithm gets S, and decides on $\widehat{h}_S$. Then, the prediction function can predict the labels of the test images in T using $\widehat{h}_S$. The error of the prediction rule $\widehat{h}_S$ on the test images is:

$$\text{err}(\widehat{h}_S, T) = \frac{1}{m} \sum_{(x,y) \in T} \mathbb{I}[\widehat{h}_S(x) \neq y].$$

Since T is an i.i.d. sample from $\mathcal{D}$, $T \sim D^m$, the error on T is a good estimate of the error of $\widehat{h}_S$ on the distribution $\text{err}(\widehat{h}_S, D)$.

To load all the MNIST data, run the following command (after making sure that the `mnist_all.npz` file is in your working directory):

```
>>> data = np.load('mnist_all.npz')
```

This command will load the MNIST data to a Python dictionary called `data`. You can access the data by referencing the dictionary: `data['train0']`, `data['train1']`, ..., `data['train9']` `data['test0']`, `data['test1']`, ..., `data['test9']`.

To generate a training sample of size m with images only of some of the digits, you can use the function `gensmallm` which is provided in the shell file `nearest_neighbour.py`. The function is used as follows:

```
>>> (X, y) = gensmallm([labelAsample, labelBsample], [A, B], samplesize)
```

The function `gensmallm` selects a random subset from the provided data of labels A and B and mixes them together in a random order, as well as creates the correct labels for them. This can be used to generate the training sample and the test sample.

Answer the following questions in the file "answers.pdf":

(a) (3 points) Run your k-NN implementation with $k = 1$, on several training sample sizes between 1 and 100 (select the values of the training sizes that will make the graph most informative). For each sample size that you try, calculate the error on the full test sample. You can calculate this error, for instance, with the command:

```
    >>> np.mean(y_test != y_testpredict)
```

Repeat each sample size 10 times, each time with a different random training sample, and average the 10 error values you got. Submit a plot of the average test error (between 0 and 1) as a function of the training sample size. Don't forget to label the axes. Present also error bars, which show what is the minimal and maximal error value that you got for each sample size. You can use Matlab's plot command (or any other plotting software).

(b) (3 points) Do you observe a trend in the average error reported in the graph? What is it? How would you explain it?

(c) (3 points) Does the size of the error bars change with the sample size? What trend do you see? What do you think is the reason for this trend?

(d) (4 points) Run your k-NN implementation for each of the following training sample sizes $m \in \{50, 150, 500\}$. For each sample size m,

- run for values of k between 1 and 15;
- create and submit a separate plot of the test errors as a function of k;
  for each k, the reported test error is the average over 30 runs (each with a new randomly chosen sample of m examples).

What is the optimal value of k you got for each of the sample sizes? Explain the trend in each graph and between the graphs of different m values.

(e) (3 points) Check what happens if the labels are corrupted, as follows: repeat the experiments on the values of k as in the previous item, but this time, for every training set and test set that you feed into your functions, first select a random 30% of the examples and change their label to a different label, which you will randomly select from the three other possible labels. Plot the graphs for each sample size of the test error as a function of k for this set of experiments.

(f) (4 points) Compare the two graphs you got for each sample size m for the two experiments from sections (d) and (e). What is the optimal value of k for each experiment? Is there a difference between the two experiments? How do you explain it?

Question 3. Consider a distribution over rabbits and food preferences. Each rabbit likes either carrot or lettuce. For each rabbit, we measure their weight in kg and their age in months. In principle, a rabbit can live until age 48 months and weigh as much as 4 kg.

(a) What should $\mathcal{X}$ and $\mathcal{Y}$ be in this problem? Define $\mathcal{X}$ to be as specific as possible given the problem description.

(b) We have a distribution $\mathcal{D}$ over rabbits with the following probabilities (all other values have zero probability):

age (months)	weight (kg)	preferred food	probability
7	1	carrot	0%
7	1	lettuce	10%
7	2	carrot	0%
7	2	lettuce	50%
13	1	carrot	15%
13	1	lettuce	0%
13	2	carrot	0%
13	2	lettuce	25%

Denote the Bayes-optimal predictor for $\mathcal{D}$ by h_{bayes}. Write the value of $h_{\text{bayes}}(x)$ for each x in the support of $\mathcal{D}$. What is the Bayes-optimal error of $\mathcal{D}$?

(c) Suppose we now only measure the age of the rabbits, and so we have a distribution $\mathcal{D}'$ which is exactly like $\mathcal{D}$ except that the weight of the rabbit is not measured, only the age. Provide a table of probabilities for this distribution.

(d) Denote the Bayes-optimal predictor for $\mathcal{D}'$ by h'_{bayes}. Write the value of $h'_{\text{bayes}}(x)$ for each x in the support of $\mathcal{D}'$. What is the Bayes-optimal error of $\mathcal{D}'$?

(e) Suppose that the *Memorize* learning algorithm that we saw in class gets a sample $S \sim \mathcal{D}^m$ and outputs a predictor. Give a formula for the expected error of Memorize as a function of m for the distribution $\mathcal{D}$ that we defined above. Recall that this expected error is formally denoted $\mathbb{E}_{S\sim\mathcal{D}^m}\left[\operatorname{err}(\widehat{h}_S, \mathcal{D})\right]$. Calculate the value of the expected error for $m = 2$.

Question 4. In this question, you will show that even in a very simple distribution, 1-Nearest Neighbor (1-NN) does not necessarily approach the Bayes-optimal error, and can have an error that is arbitrarily close to twice the Bayes-optimal error. You should prove your claims, using basic probability and expectation properties.

Let $\mathcal{X} = \{a\}$, and let $\mathcal{Y} = \{0, 1\}$. Let $\mathcal{D}$ be a distribution over $\mathcal{X} \times \mathcal{Y}$. Recall the definition of the function η for $\mathcal{D}$:

$$\eta(x) := \Pr_{(X,Y)\sim\mathcal{D}}[Y = 1 \mid X = x].$$

Denote $\psi := \eta(a)$.

Note that in our case, there is only one possible input value, $x = a$. In the 1-NN algorithm, assume that for any test input x, if the training sample S includes more than one example that is nearest to x (there is a tie), one of these points from the sample is selected uniformly at random and its label is chosen as the prediction. Let $\widehat{h}_S$ be the (possibly random) prediction rule of the 1-NN algorithm for a given training sample S.

(a) (4 points) Show that for a fixed S, a single number that depends on S (call it β_S) controls the probability $\Pr\left[\widehat{h}_S(a) = 1\right]$. Give a formula for β_S as a function of

$$S = \{(x_1, y_1), \dots, (x_m, y_m)\}.$$

Note that here the probability is over the randomness of $\widehat{h}_S$, since S is fixed.

(b) (4 points) Calculate $\operatorname{err}\left(\widehat{h}_S, \mathcal{D}\right)$ as a function of β_S and ψ.

(c) (4 points) Calculate the expectation of β_S over samples, denoted $\mathbb{E}_{S\sim\mathcal{D}^m}[\beta_S]$, as a function of ψ.

(d) (3 points) Calculate the expected error of the 1-NN algorithm for $\mathcal{D}$ over random samples, denoted by

$$\operatorname{err} := \mathbb{E}_{S\sim\mathcal{D}^m}\left[\operatorname{err}\left(\widehat{h}_S, \mathcal{D}\right)\right],$$

as a function of ψ.

(e) (3 points) Calculate the error of the Bayes-optimal rule for $\mathcal{D}$, denoted by $\operatorname{err}_{\text{bayes}}$, as a function of ψ.

(f) **(Bonus subsection: 3 points)**
Show that for any $\epsilon > 0$, there exists a value of ψ such that the ratio

$$\frac{\operatorname{err}}{\operatorname{err}_{\text{bayes}}}$$

is larger than $2 - \epsilon$. Conclude that without knowing anything about the distribution, we cannot guarantee a factor lower than two (relative to the Bayes-optimal error) when using the 1-NN algorithm.

Question 5. Consider a classification problem with an input domain $\mathcal{X}$, a label domain $\mathcal{Y}$, and a probability distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$.

The probability distribution $\mathcal{D}$ is realizable for the hypothesis class $\mathcal{H}$.

We are given K statistically-independent samples $S_1, \dots, S_K$. For $j \in \{1, \dots, K\}$, the sample S_j includes m_j examples of input-label pairs that were independently and identically distributed (i.i.d.) drawn from $\mathcal{D}$.

The goal is to learn a predictor from the hypothesis class $\mathcal{H}$. The learning is performed by an algorithm that returns a predictor $\widehat{h}_{SK} : \mathcal{X} \to \mathcal{Y}$ that achieves minimal empirical error on each of the K samples:

$$\widehat{h}_{SK} \in \left\{ h \,\middle|\, \forall j \in \{1, \dots, K\},\ h \in \underset{f\in\mathcal{H}}{\operatorname{argmin}}\, \widehat{\operatorname{err}}\left(f, S_j\right) \right\}$$

where

$$\widehat{\operatorname{err}}\left(f, S_j\right) = \frac{1}{m_j} \sum_{(x,y)\in S_j} \mathbb{I}[f(x) \neq y]$$

is the empirical error of the predictor $f \in \mathcal{H}$ on the sample S_j.

Recall: $\mathbb{I}\left[\text{condition}\right]$ is an indicator function that returns 1 if the condition is satisfied, and return 0 otherwise.

Important: The distribution $\mathcal{D}$ is realizable by $\mathcal{H}$ and therefore a predictor $\widehat{h}_{SK}$ exists.

Let $\epsilon \in (0, 1)$ be a predetermined constant value.

(a) (10 points) A bad predictor $h_{\text{bad}} \in \mathcal{H}$ satisfies $\text{err}(h_{\text{bad}}, \mathcal{D}) > \epsilon$.

Mathematically prove (including detailed justifications for the steps of the proof) the following upper bound:

$$\mathbb{P}_{\{S_j \sim \mathcal{D}^{m_j}\}_{\forall j \in \{1,\ldots,K\}}}\Big[\forall j \in \{1,\ldots,K\},\ \widehat{\text{err}}\left(h_{\text{bad}}, S_j\right) = 0\Big] \leq (1-\epsilon)^b$$

where $b = \sum_{j=1}^{K} m_j$.

(b) (9 points) **Formulate** a mathematical condition on $\{m_j\}_{j=1}^{K}$ that guarantees the prediction performance of

$$\mathbb{P}_{\{S_j \sim \mathcal{D}^{m_j}\}_{\forall j \in \{1,\ldots,K\}}}\Big[\text{err}\left(h_{\text{bad}}, \mathcal{D}\right) \leq \epsilon\Big] \geq 1 - \delta$$

where $\epsilon, \delta \in (0,1)$.

The condition formula in this section may use **only** $\epsilon, \delta, K, |\mathcal{H}|, \{m_j\}_{j=1}^{K}$ and standard mathematical operators.

In this section 5(b), a detailed mathematical proof is not mandatory, but you should clearly explain and justify how you formulated the mathematical condition.

Question 6. Consider a binary classification problem with input space $\mathcal{X} = \mathbb{R}$ and label space $\mathcal{Y} = \{0, 1\}$.

Recall: $\mathbb{I}[\text{condition}]$ is an indicator function that returns 1 if the condition is satisfied, and return 0 otherwise.

(a) (10 points) Consider the following function for a real input $x \in \mathbb{R}$:

$$f_c(x) = \mathbb{I}\left[|x - c| < 1\right]$$

where $c \in \mathbb{R}$ is a parameter of the function.

We define the hypothesis class $\mathcal{H} = \{f_c \mid c \in \mathbb{R}\}$.

What is the VC dimension of $\mathcal{H}$?

Prove your answer in detail.

(b) (8 points) Consider the following function for a real input $x \in \mathbb{R}$:

$$f_{c,r}(x) = \mathbb{I}\left[|x - c| < r\right]$$

where $c, r \in \mathbb{R}$ are two parameters of the function.

We define the hypothesis class $\widetilde{\mathcal{H}} = \{f_{c,r} \mid c, r \in \mathbb{R}\}$.

What is the VC dimension of $\widetilde{\mathcal{H}}$?

Prove your answer. In the proof you can refer to parts of the proof for the previous subsection.