Introduction to Machine Learning
Exercise 1

Fall 2024/1

Submission guidelines, **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 them via the course requests system.
- Grading: Q.1 (python code): 20 points, Q.2: 20 points, Q.3: 18 points, Q.4: 18 points. Q.5: 24 points.

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's 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 be also 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 `Ytestprediction`. This 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 `z1, z2` 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 data set 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 data set taken out of MNIST, that only includes images with the digits 2,3,5 and 6, so that there are only four possible labels. Each image in MNIST has 28 by 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 $\mathcal{X} = \mathbb{R}^{784}$: every example is described by a 748-coordinate vector.

Figure 1: Some examples of images of digits from the data set MNIST

The images in MNIST are split to 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 $\hat{h}_S$. Then, the prediction function can predict the labels of the test images in T using $\hat{h}_S$. The error of the prediction rule $\hat{h}_S$ on the test images is $\mathrm{err}(\hat{h}_S, T) = \frac{1}{m} \sum_{(x,y)\in T} \mathbb{I}[\hat{h}_S(x) \neq y]$. Since T is an i.i.d. sample from $\mathcal{D}$, $T \sim \mathcal{D}^m$, the error on T is a good estimate of the error of $\hat{h}_S$ on the distribution $\mathrm{err}(\hat{h}_S, \mathcal{D})$.

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

(c) Did you get different results in different runs with the same sample size? Why?

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

(e) Run your k-nn implementation with a training sample size of 200, for values of k between 1 and 11. Submit a plot of the test errors as a function of k, again averaging 10 runs for each k. What is the optimal value of k you got? Explain the trend of the graph.

Question 3. Let $\mathcal{X} \subseteq \mathbb{R}^d$ be a finite domain, and let $\mathcal{Y} = \{0, 1\}$. Let $\mathcal{D}$ be a distribution over $\mathcal{X} \times \mathcal{Y}$. Suppose that $\mathcal{D}$ has a Bayes-error of zero and that η of $\mathcal{D}$ is c-Lipschitz with respect to the Euclidean distance.

(a) Let $S \sim \mathcal{D}^m$. Prove that for any two pairs $(x_1, y_1), (x_2, y_2) \in S$, if $y_1 \neq y_2$ then $\|x_1 - x_2\| \geq 1/c$.

(b) Let $\mathcal{B}$ be a set of balls of radius $1/(3c)$ that *cover* the space of points $\mathcal{X}$, meaning that every point from $\mathcal{X}$ is in at least one ball in $\mathcal{B}$. Let $S \sim \mathcal{D}^m$ such that for every ball $B \in \mathcal{B}$, there is some pair $(x, y) \in S$ which satisfies $x \in B$. Prove that under this assumption, and the other assumptions on $\mathcal{D}$ given above, $\mathrm{err}(f_S^{nn}, \mathcal{D}) = 0$, where f_S^{nn} is the 1-nearest-neighbor function defined in class.

Question 4. Consider a distribution over a population of rabbits. Each rabbit is either black or white. Our goal is to predict the color of the rabbit based on its age and weight. For each rabbit, we measure their weight in Kg and their age in months. In principle, a rabbit can live until age 42 months and weigh as much as 5kg.

(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 options have zero probability):

age (months)	weight (kg)	color	probability
8	4	black	42%
8	4	white	6%
15	1	black	21%
15	1	white	7%
15	2	white	24%

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 $\mathcal{D}$.

(c) Calculate the Bayes-optimal error of $\mathcal{D}$.

(d) Let $\mathcal{H} : \mathcal{X} \to \mathcal{Y}$ be a hypothesis class that includes only constant functions, that is: functions that assign all examples to the same label. What is the approximation error of $\mathcal{H}$ for the distribution $\mathcal{D}$ defined above?

(e) Consider the following distribution $\mathcal{D}''$:

age (months)	weight (kg)	color	probability
8	4	black	8%
8	5	white	15%
9	2	black	47%
11	6	white	30%

Calculate the value of the expected error of the Memorize algorithm for sample size $m = 3$ using the formula we learned in class. Explain why you are allowed to use this formula for $\mathcal{D}''$ but not for the distribution $\mathcal{D}$ defined above.

Question 5. A farmer believes that apples that drop from their tree are tasty only if they are sufficiently close to the tree trunk. To base this belief, the farmer tasted 20 apples that dropped from the same tree; for each apple the following information was recorded:

- $x = [x_h, x_v]^T \in \mathbb{R}^2$: The two-dimensional coordinate where the apple dropped. Here, the tree trunk coordinate is defined as $(0, 0)$, $x_h \in \mathbb{R}$ is the apple's horizontal offset from the tree trunk, $x_v \in \mathbb{R}$ is the apple's vertical offset from the tree trunk.
- $y \in \{0, 1\}$: A binary label that specifies if the apple tasted good ($y = 1$), or bad ($y = 0$).

The sample S of 20 apples that the farmer examined are shown as circle markers in the following figure, where the marker location denotes the apple's offset $[x_h, x_v]$ from the tree trunk and the marker color denotes the taste label (blue is $y = 1$, red is $y = 0$):

The goal of the farmer is to learn a rule that predicts if a newly dropped apple is tasty based on its offset from the tree. Namely, given a new offset $[x_h, x_v]$, predict the unknown taste label y. The input domain here is $\mathcal{X} = \mathbb{R}^2$ and the output (label) domain is $\mathcal{Y} = \{0, 1\}$.

We define the absolute-value thresholding function for a real valued input $z \in \mathbb{R}$ as (note the direction of the inequality sign)

$$f_\tau^{\text{abs}}(z) := \mathbb{I}[|z| \leq \tau]$$

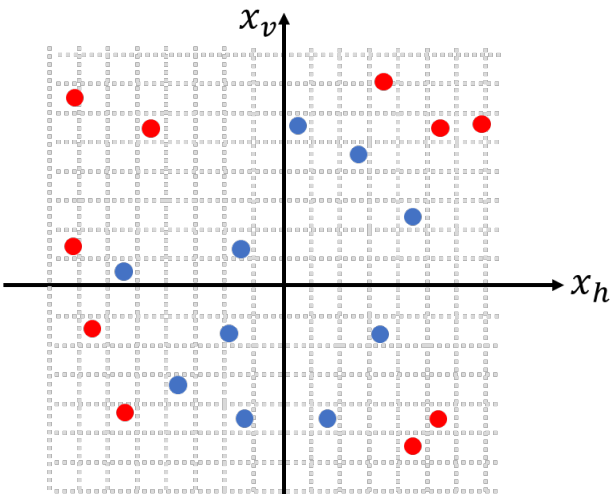

Figure 2: The sample S of 20 apples. Auxiliary grid appears as evenly-spaced dotted gray lines.

where $|z|$ is the absolute value of z, $\tau > 0$ is the threshold, and $\mathbb{I}[\text{condition}]$ is the indicator function that returns 1 if the condition is satisfied, and 0 otherwise.

(a) We define the rectangular thresholding function for $x = [x_h, x_v]^T \in \mathbb{R}^2$ as

$$f^{\text{rec}}_{\tau_h,\tau_v}(x) := f^{\text{abs}}_{\tau_h}(x_h) \cdot f^{\text{abs}}_{\tau_v}(x_v).$$

Then, the hypothesis class of rectangular thresholding functions is defined as

$$\mathcal{H}_{\text{rec}} := \{f^{\text{rec}}_{\tau_h,\tau_v} \mid \tau_h, \tau_v \in \mathbb{R}\}.$$

Considering the sample S of 20 apples in Figure 2, what is the empirical error achieved by ERM with the hypothesis class of rectangular thresholding functions $\mathcal{H}_{\text{rec}}$?

(b) Given a function $g : \mathbb{R}^2 \to \mathbb{R}$, the input $[x_h, x_v]$ can be transformed into a single feature $g(x_h, x_v) \in \mathbb{R}$. For a given g and a threshold $\tau \in \mathbb{R}$, we define the thresholding function for an input $x = [x_h, x_v]$ as

$$f_{g,\tau}(x) := f^{\text{abs}}_{\tau}(g(x_h, x_v))$$

and the corresponding hypothesis class is

$$\mathcal{H}_g := \{f_{g,\tau} \mid \tau \in \mathbb{R}\}.$$

Mathematically formulate a function g for which

- the ERM with $\mathcal{H}_g$ achieves zero empirical error on the sample S of 20 apples from Figure 2,
- a prediction rule provided by ERM with $\mathcal{H}_g$ can be assumed to generalize to new apples better than a prediction rule provided by ERM with the rectangular hypothesis class $\mathcal{H}_{\text{rec}}$ from the previous section.

Explain your answer.