Introduction to learning and analysis of big data
Exercise 1

Prof. Sivan Sabato

Fall 2022/3

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: 21 points, Q.3: 20 points, Q.4: 21 points. Q.5: 18 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.

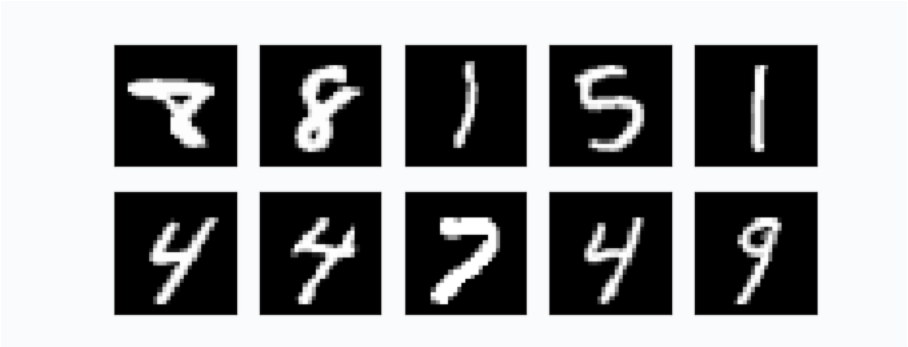

Figure 1: Some examples of images of digits from the data set MNIST

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.

(f) Now, check what happens if the labels are corrupted, as follows: repeat the experiment 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 15% of the examples and change their label to a different label, which you will randomly select from the three other possible labels. Plot the graph of the error as a function of k for this set of experiments.

(g) Compare the two graphs you got for the two experiments on the size of k. 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. 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 48 months and weigh as much as 4kg.

(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
5	2	black	8%
5	2	white	47%
12	1	black	20%
12	1	white	4%
12	2	white	21%

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

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

(d) 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 $\mathcal{D}$ defined above, which we denoted in class by $\mathbb{E}_{S\sim\mathcal{D}^m}[\mathrm{err}(\hat{h}_S, \mathcal{D})]$.

(e) We now have a different distribution $\mathcal{D}'$. In this distribution, a rabbit which is n months old has a probability of $2n\%$ to be black. For instance:

$$\mathbb{P}[\text{rabbit is black} \mid \text{rabbit's age is 4 months}] = 8\%.$$

What is the Bayes-optimal predictor for this distribution?

(f) Can you calculate the error of the Bayes optimal predictor you provided for $\mathcal{D}'$? Explain.

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

age (months)	weight (kg)	color	probability
5	2	black	6%
5	3	white	12%
7	1	black	53%
9	4	white	29%

Calculate the value of the expected error of the Memorize algorithm for sample size $m = 5$ 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. Let $\mathcal{X} = [0,1]$, $\mathcal{Y} = \{0,1\}$. Consider the hypothesis class of thresholds that we defined in class,

$$\mathcal{H}_{\text{th}} := \{f_a \mid a \in [0,1]\}, \text{ where } f_a(x) := \mathbb{I}[x \geq a].$$

Let $\mathcal{D}$ be a distribution over $\mathcal{X} \times \mathcal{Y}$. Suppose that the marginal distribution of $\mathcal{D}$ on $\mathcal{X}$ is uniform on $[0,1]$, and that for $(X,Y) \sim \mathcal{D}$, for any $x \in \mathcal{X}$, $\mathbb{P}[Y = 1 \mid X = x] = \mathbb{I}[x \geq \beta]$, for some fixed $\beta \in [0,1]$. Note that this means the distribution is **realizable** by $\mathcal{H}_{\text{th}}$.

(a) For an integer N, consider a hypothesis class with N evenly-spaced thresholds:

$$\mathcal{H}_{\text{th}}^N := \{f_a \mid a \in \{i/N\}_{i \in \{0,\ldots,N\}}\}, \text{ where } f_a(x) := \mathbb{I}[x \geq a].$$

Using the PAC bounds for finite hypothesis classes that we learned in class, and assuming that $\mathcal{D}$ is realizable by $\mathcal{H}_{\text{th}}^N$, find the sample size that is required so that there is a probability of at least 95% that the output of the ERM algorithm has an error of at most 3% on the distribution. The resulting sample size will be a function of N.

(b) Let $\epsilon \in [0,1]$. Prove that if $a \in [\beta - \epsilon, \beta + \epsilon]$, then $\text{err}(f_a, \mathcal{D}) \leq \epsilon$.

(c) Suppose that we run some ERM algorithm with the hypothesis class $\mathcal{H}_{\text{th}}$ on a sample $S \sim \mathcal{D}^m$. Prove that if there exists some $(x,y) \in S$ such that $x \in [\beta - \epsilon, \beta]$ and there exists some $(x,y) \in S$ such that $x \in [\beta, \beta + \epsilon]$, then the output classifier $\hat{h}_S$ returned by the algorithm is equal to f_a for some $a \in [\beta - \epsilon, \beta + \epsilon]$. Conclude that $\text{err}(\hat{h}_S, \mathcal{D}) \leq \epsilon$.

(d) Prove that for $S \sim \mathcal{D}^m$, the probability that there does not exist a $(x,y) \in S$ such that $x \in [\beta, \beta + \epsilon]$ is $(1-\epsilon)^m$, and that the same holds for the existence of $x \in [\beta - \epsilon, \beta]$. Let $\hat{h}_S$ be the classifier returned by a ERM algorithm with $\mathcal{H}_{\text{th}}$ as defined above. Conclude, using the above sub-questions and the union bound, that

$$\mathbb{P}_{S \sim \mathcal{D}^m}[\text{err}(\hat{h}_S, \mathcal{D}) \leq \epsilon] \geq 1 - 2(1-\epsilon)^m.$$

(e) Based on the above inequality, what sample size is needed (that is, what should the value of m be) if we wish to guarantee that there is a probability of at least 95% that the output of the ERM algorithm has an error of at most 3% on the distribution? Explain the calculation.

(f) Suppose that we do not have any information about the optimal threshold for $\mathcal{D}$ except that it is a rational number. Which approach for bounding the sample complexity is better? The one you used in (a) or the one in (e)? Explain.