INSTITUTO
UNIVERSITÁRIO
DE LISBOA

Introduction to Machine Learning — 2024/2025
Supervised Learning

> These exercises should be solved using Python notebooks (Jupyter) due to the ability to generate a report integrated with the code. It is assumed you are proficient with programming. All answers must be justified and the results discussed and compared to the appropriate baselines.
> Max score of the assignment is 2 points. Extra exercises (if existing) help achieving the max score by complementing errors or mistakes.
> **Deadline:** end of class in week October, 28th-31th, 2024

In the following exercises the objective is to program algorithms that, given examples and an expected output, learn to mimic the behavior present in the data.

Exercise 1

The "network" in Fig. 1 represents a perceptron with two inputs and an output that can also be described by the following equations:

$$o = f(s), \quad s = w_0 + w_1 \cdot x_1 + w_2 \cdot x_2 \tag{1}$$

$$f(s) = \begin{cases} 1, & \text{if} \quad s > 0.5 \\ 0, & \text{if} \quad s \leq 0.5 \end{cases} \tag{2}$$

1. **Choose one** of the binary operations (AND or OR) and **build two vectors**: one with all the different input combinations of two bit patterns (4 vectors):

   $$\{\{0,0\},\{0,1\},\{1,0\},\{1,1\}\}$$

   where 0 stands for $FALSE$ and 1 for $TRUE$; and another vector containing the target / desired response, d, for each of the corresponding input vectors, as result of the chosen operation, namely: OR $\{0,1,1,1\}$ **or** AND $\{0,0,0,1\}$.

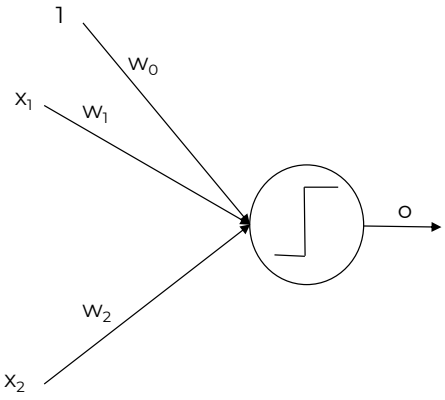

Figure 1: Perceptron

2. **Initialize w_0, w_1, and w_2 to small random values** and, **for each input pattern, calculate the corresponding output**, storing it in vector o.

3. **Calculate** the difference / **error** ($e = d - o$) between the desired response (d) and the output (o), **for each output**.

4. For each error in e, add to the update term for w_0 (Δw_0), w_1 (Δw_1), and w_2 (Δw_2) according to:

$$\Delta w_0 = \Delta w_0 + \alpha \cdot e \tag{3}$$
$$\Delta w_1 = \Delta w_1 + \alpha \cdot x_1 \cdot e \tag{4}$$
$$\Delta w_2 = \Delta w_2 + \alpha \cdot x_2 \cdot e \tag{5}$$

   where $\alpha = 10E - 4$.

5. Prepare your code to cycle through the whole dataset (in this case, 4 examples) several times doing the above procedure (**to train for several "epochs"**).

6. After all examples are presented **(at the end of each epoch), update** w_0, w_1 and w_2 according to:

$$w_0 = w_0 + \Delta w_0 \tag{6}$$
$$w_1 = w_1 + \Delta w_1 \tag{7}$$
$$w_2 = w_2 + \Delta w_2 \tag{8}$$

   so that in the next iteration the error will decrease. **Repeat for 20 epochs**.

   (a) **Plot the value of the error at the end of each epoch**, how does it behave?

   (b) **Plot the value of each weight at the end of each training epoch**. Are the values converging? if so, do they converge to similar values in different runs (with

different random intializations)?

(c) **What is the effect of increasing/decreasing the α parameter?** Can you tell (approximately) what is the "best" value for α?

(d) **How many epochs** (iterations through the whole set) did it take **to get all examples right?** (i.e. $\forall_i : d_i = o_i$). Repeat the experiment 30 times with different random values for the initial weights and present the average and standard deviation of the number of epochs it took to converge.

7. **Generate 2D points using a multivariate Gaussian distribution**.

(a) Use the code in Fig. 2 to generate two sets, each with 500 points (reduce this number if necessary to obtain better visualizations or faster training runs),

(b) Each dataset should have different centers, and sets should have a small overlap.

(c) Add a column and fill it with 0 for the first dataset and 1 for the second, so that you can keep track of which distribution generated each point.

(d) Join and shuffle the dataset.

(e) The plot of the first two columns should be similar to the one presented in Fig. 3.

(f) Write the dataset to a file.

```
import matplotlib.pyplot as plt
import numpy as np
import random

mean = [3, 3]
cov = [[1, 0], [0, 1]]
a = np.random.multivariate_normal(mean, cov, 500).T

mean = [-3, -3]
cov = [[2, 0], [0, 5]]
b = np.random.multivariate_normal(mean, cov, 500).T

c = np.concatenate((a, b), axis = 1)
c = c.T
np.random.shuffle(c)
c = c.T

x = c[0]
y = c[1]

plt.plot(x, y, 'x')
plt.axis('equal')
plt.show()
```

Figure 2: Code snipet to generate multivariate Gaussian

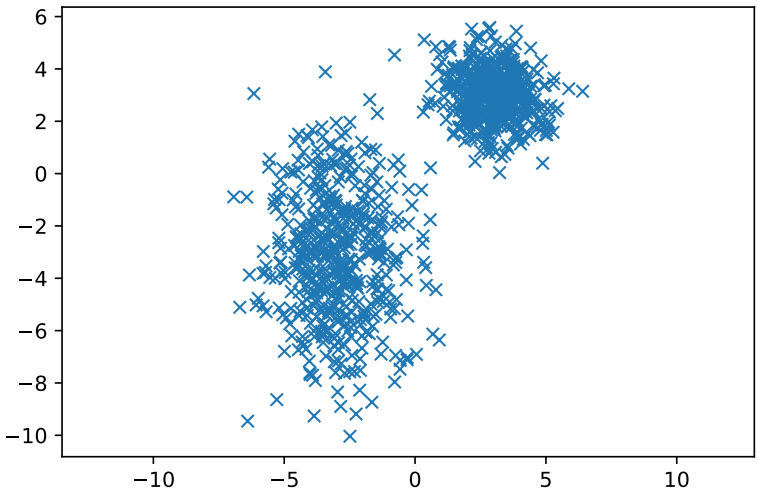

Figure 3: 2D points generated from two multivariate Gaussian distributions

8. **Use the dataset** generated in the previous item **as the training set** for the same perceptron and train it to partition the two datasets (adjust number of epochs if necessary). Notice that the same program learned two different tasks depending on the dataset used.

   Print with different colors:

   (a) Points generated by the first distribution and labeled 1 by the perceptron;

   (b) Points generated by the first distribution and labeled 0 by the perceptron;

   (c) Points generated by the second distribution, labeled 1 by the perceptron;

   (d) Points generated by the second distribution, labeled 0 by the perceptron.

9. **Print/Plot the confusion matrix for the above test**. Can you **relate each of the numbers in the confusion matrix to the points** of a given color on the previously generated figure?

10. **Print the metrics** (accuracy, precision, recall, and F1) for all the tests: **metrics should be an average for 30 tests** with the same parameters but different initial weights.

Exercise 2

Implement a k-NN classifier that is specifically suited for the dataset in `https://archive.ics.uci.edu/ml/datasets/iris`.

Given a dataset containing labelled examples (a *training set*) and a new example (extracted from the test set), the classifier should **calculate the euclidean distance** from the

new example to all the elements of the training set, **choose the k closest** elements of the training set and output this example classification as **the class of the majority** of the k closest training set elements (the k-Nearest Neighbors).

1. **Split the dataset randomly** in two subsets (70% / 30%). Use the bigger subset as the *training set* and the smaller as the *test set*. Run all test examples through the classifier and calculate the number of correct predictions over the total number of examples of the test set. Compare the scores of k-NN classifiers for $k = 3, 7,$ and11. Repeat 30 times, with different dataset splits, for each value of k. Use a boxplot with whiskers graphic to allow easy comparison.
2. **Plot the confusion matrix** of one of the tests for each value of k.
3. Considering the dataset presented in Fig. 3, why should k always be an odd number?

Exercise 3

Using the dataset from the previous exercise, **implement a Naive Bayes classifier**.

1. **Transform by discretizing all columns' values into categories with three possible values (low / medium / high)**. Use a sensible partition for each column. As in the previous exercise, split the dataset randomly in two subsets (70% / 30%). Repeat the process of the previous exercise to obtain evaluation metrics and an example of a confusion matrix (this time, there is no parameter to vary, so only one cycle of 30 repetitions with different dataset partitions).
2. **How does this classifier compare to the k-NN classifier**?

   Remember that the probability of an event/example belonging to a given class $P(Class|X)$ is calculated by:

   $$\mathrm{P}(Class|X) = \frac{\mathrm{P}(X|Class)\,\mathrm{P}(Class)}{\mathrm{P}(X)} \tag{9}$$

   and the decision between classes for a given example X is made by comparing the values for each class of:

   $$\mathrm{P}(X|Class)\,\mathrm{P}(Class) \tag{10}$$

   which is approximated (using the independence assumption) by:

   $$\mathrm{P}(Class)\prod_i \mathrm{P}(X_i|Class). \tag{11}$$

Exercise 4

Use the dataset from the previous exercise, with the categorized values **(high, medium, low)**. Use *Iris-setosa* as your target value ($p+$ are the examples classified as *Iris-setosa* and $p-$ the remaining ones) and create 3 new datasets by putting in each the examples that

have the same values in the first column, i.e. Low DataSet has all elements that have the value low in the first column, Medium Dataset, has all the examples that have $medium$ value in the dataset and High Dataset has all elements that have $high$ value in the first column.

1. **Calculate the entropy of the 4 datasets** (the complete dataset, and the three subsets).

   Remember that a set's (S) entropy is calculated by:

   $$\text{entropy}(S) = -(p+) \times \log_2(p+) - (p-) \times \log_2(p-) \tag{12}$$

   Calculate the gain of the split of S by feature a:

   $$\text{gain}(S, a) = \text{entropy}(S) - \frac{\sum_v (|S_v| \times \text{entropy}(S_v))}{|S|} \tag{13}$$

   where $|S|$ is the number of elements in S and S_v is each of the subsets of S when partitioned by the value of a.

2. **What is the value of $\text{gain}(S, a)$ for the above split?** What does it mean in terms of your ability to classify the elements of S before and after the split?

3. Do the same for all features of your dataset. **Which is the feature with greatest gain?** How can you improve your chances of guessing a random examples' class using this information?

4. Explain how to build a decision tree with this information.