https://vianney.ai/

Theoretical foundations of Machine Learning

TD/TP 1

Il faut toujours rendre à César ce qui lui appartient, je veux remercier Joon Kwon for son aide à la rédaction des fiches de TD and TP.

Exercice 1 We observe $\mathcal{D}_n = ((X_1, Y_1), ..., (X_n, Y_n))$ i.i.d with law P with $X_i \in \{1, \dots, k\}$ and $Y_i \in \{0, 1\}$. We let, for $x \in \{1, \dots, k\}$, $N_x = \text{card}(\{i : X_i = x\})$. We consider the the function $\eta(x) = \mathbb{E}(Y|X = x)$ and define its estimator

$$\hat{\eta}(x) = \begin{cases} \frac{1}{N_x} \sum_{i:X_i=x} Y_i \text{ if } N_x > 0, \\ 0 \text{ else,} \end{cases}$$

and we define the classification rule $\hat{f}(\mathcal{D}_n, x) = \mathbf{1}_{\{\hat{\eta}(x) \geq 1/2\}}$.

1. Show that, for $x \in \{1, \dots, k\}$ fixed,

$$\mathbb{E}\Big(|\hat{\eta}(x) - \eta(x)|\Big| X_1, \dots, X_n\Big) \leq \frac{1}{\sqrt{N_x}} \mathbf{1}_{\{N_x > 0\}} + \mathbf{1}_{\{N_x = 0\}}.$$

   Deduce that

$$\mathbb{E}\Big[R(\hat{f}) - R^*\Big] \leq 2\mathbb{E}\left[\frac{1}{\sqrt{N_X}} \mathbf{1}_{\{N_X > 0\}}\right] + \mathbb{P}(N_X = 0).$$

2. Show that for all x such that $P(X_1 = x) > 0$ we have $N_x \to \infty$ almost surely when $n \to \infty$, then show that the two terms in the right-hand side of the equation above converge to 0.

3. Conclude: show that $\hat{f}$ is universally consistent.

Exercice 2
We consider a random variable X drawn according to a distribution P_θ, where the real parameter θ itself follows a distribution π. To estimate θ, we consider the loss function:

$$L(\theta, a) = \begin{cases} k_2(\theta - a) & \text{if } \theta > a \\ k_1(a - \theta) & \text{else} \end{cases}$$

Show that the Bayes estimator is a quantile (to be determined) of the law of θ given X.

Exercice 3
Suppose that (X, Y) is a couple with distribution $\mathbb{P}$ such that

- $Y \sim Be(p)$ is a Bernoulli.
- for $y \in \{0, 1\}$, $X|Y = y$ has a density $f_y(\cdot)$ with respect to the Lebesgue measure on $\mathbb{R}$.

1. Write the regression function $\eta(x) = \mathbb{P}\{Y = 1|X = x\}$ in terms of f_0 , f_1 and p.
2. Write the optimal classifier f^* for the 0/1 loss.

3. Find f^* and its risk $L^* = \mathbb{P}\{Y \neq f^*(X)\}$ in the following cases
   (a) $f_0(x) = \frac{1}{\theta_0}\mathbb{1}\{x \in [0, \theta_0]\}$ and $f_1(x) = \frac{1}{\theta_1}\mathbb{1}\{x \in [0, \theta_1]\}$ for $\theta_1 > \theta_0 > 0$
   (b) $f_0(x) = \frac{1}{\theta_0}\mathbb{1}\{x \in [0, \theta_0]\}$ and $f_1(x) = \frac{1}{\theta_1 - \theta_0}\mathbb{1}\{x \in [\theta_0, \theta_1]\}$ for $\theta_1 > \theta_0 > 0$
   (c) $f_0(x) = \theta_0 e^{-\theta_0 x}\mathbb{1}\{x \in \mathbb{R}_+\}$ and $f_1(x) = \theta_1 e^{-\theta_1 x}\mathbb{1}\{x \in \mathbb{R}_+\}$ for $\theta_1 > \theta_0 > 0$
   (d) $f_0(x) = \frac{1}{\sqrt{2\pi\sigma_0^2}} e^{-\frac{(x-\theta_0)^2}{2\sigma_0^2}}$ and $f_1(x) = \frac{1}{\sqrt{2\pi\sigma_1^2}} e^{-\frac{(x-\theta_1)^2}{2\sigma_1^2}}$ for $\theta_1 > \theta_0$ and $\sigma_0, \sigma_1 > 0$

Part I
Introduction to Python, Numpy, etc.

1 First steps with NumPy and Matplotlib

1.1 NumPy

NumPy introduces the `array` data type, which is a multidimensional array of elements. A one-dimensional `array` corresponds to a vector, and a two-dimensional `array` to a matrix. To become familiar with the package, we will execute each line of code below, observe the result, and possibly perform additional manipulations.
We load the package.

```
import numpy as np
```

Here are différent ways to build one-dimensional `arrays`.

```
a = np.array([1,42,18])
b = np.arange(10)
c = np.arange(2,5,.5)
c = np.linspace(0,1,11)
d = np.ones(6)
d = np.zeros(5)
e = np.full(5,3)
```

Here are two different ways to create the same two-dimensional `array`.

```
A = np.array([[1,2,3], [4,5,6], [7,8,9]])
B = np.array([1,2,3,4,5,6,7,8,9]).reshape((3,3))
```

`np.ones`, `np.zeros`, `np.full` and `np.random.random` take a `tuple` as an input to output multidimensional `arrays`.

```
C = np.ones((2,5))
```

We have access to various pieces of information about an `array`.

```
A.ndim
A.size
A.shape
```

We can create `arrays` by combining multiple `arrays`.

```
np.concatenate((a,b))
np.vstack((A,a))
np.hstack((A,A))
```

Operations on `arrays` are performed element-wise and return an `array`.

```
f = np.array([2,3])
g = np.array([1.5, 3])
f+g
f*g
f**g
3*f
f == g
f < g
f <= g
```

For matrix multiplication, either of the following syntaxes may be used:

```
A.dot(a)
np.dot(A,a)
A@a
```

The transpose of a matrix can be computed in the followin way.

```
A.T
```

There are several functions that apply either to all elements of an `array` or along a specific dimension.

```
A.sum()
A.min()
A.max()
A.mean()
A.sum(axis=0)
A.sum(axis=1)
```

Elements of an `array` can be accessed using indices, which by default start at 0.

```
a
a[0]
A
A[1,1]
A[1]
```

We can also specify subsets of indices.

```
b
b[:]
b[1:8]
b[1:8:2]
b[1:8:2][2]
b[::-1]
A
A[:,1]
```

We can also specify a set of indices using a list.

```
a
a[[0,1,1]]
```

We can also use a list of booleans (`True`, `False`) to specify whether each element is selected or not (a recent version of NumPy is required for this functionality).

```
a[[False, True, True]]
```

Finally, we can select the values that satisfy a condition.

```
b[b>5]
```

We can use an `array` as a set of values to iterate over in a `for` loop.

```
for i in a:
    print(i**2)
```

NumPy provides a large number of functions that apply to `array`s component-wise.

```
np.cos(1)
np.cos(a)
```

Question 1. — Create a matrix of size 8×8 containing 0s and 1s in the manner of a chessboard.

Question 2. — Define the matrix

$$M = \begin{pmatrix} 1 & 5 & 9 & 13 & 17 \\ 2 & 6 & 10 & 14 & 18 \\ 3 & 7 & 11 & 15 & 19 \\ 4 & 8 & 12 & 16 & 20 \end{pmatrix},$$

and extract from it the matrix

$$\begin{pmatrix} 6 & 18 & 10 \\ 7 & 19 & 11 \\ 5 & 17 & 9 \end{pmatrix}.$$

1.2 Matplotlib

Matplotlib is a library for creating data visualizations. For the tasks we will focus on, it is sufficient to load `matplotlib.pyplot`.

```
import matplotlib.pyplot as plt
```

One of the most common tasks is to plot the graph of a function. To do this, we need to define a one-dimensional `array` containing the x-values. Below, we define a `array` `x` containing 10 regularly spaced numbers on the interval $[0, 4]$.

```
x = np.linspace(0,4,10)
```

For each value in `x`, we need to define the corresponding y-value for the function we want to plot.

```
y = np.sin(x)
```

We can now generate the figure.

```
plt.plot(x, y)
plt.show()
```

We observe that the graph is simply given by a finite set of points connected by line segments. To obtain a smoother graph, it is advisable to increase the number of x-values over which we compute the function. We can also plot multiple functions on the same figure.

```
x = np.linspace(0, 10, 1000)
plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x))
plt.show()
```

We can customize the style of the plot by adding a third argument of type string to the `plt.plot` function. For example, try `'g-'` or `'-.k'`. Sometimes, we may want to manually set the axis limits. This can be done as follows.

```
plt.plot(x, np.sin(x))
plt.axis([0, 5, -0.5, 0.5])
plt.show()
```

In other situations, the x and y values we provide do not represent a function, but simply a set of points. We may want to represent them without connecting them with line segments. To do this, we add a style argument, such as `'o'` (which represents a round point).

```
plt.plot([2,1,3],[-1,-1,0],'o')
plt.show()
```

Finally, we can add various elements to annotate the figure.

```
plt.plot(x, np.sin(x), '-g', label='sin(x)')
plt.plot(x, np.cos(x), ':b', label='cos(x)')
plt.title("A sinusoidal curve")
plt.xlabel("x")
plt.ylabel("sin(x)")
plt.legend()
plt.show()
```

Don't forget to invoke the `plt.legend` function, which displays the legend of the curves that were provided via the `label` option in the `plt.plot` function.

2 A one-dimensional regression problem

We are working within a regression framework in one dimension, meaning that the input space is $\mathcal{X} = \mathbb{R}$ and the output space is $\mathcal{Y} = \mathbb{R}$. The predictors are therefore functions of the form $f : \mathbb{R} \to \mathbb{R}$.

2.1 Data Generation

In this lab, we will not work with real data, but instead generate it ourselves. We consider the polynomial function:

$$g(x) = \frac{3}{2}x^3 - x^2 - \frac{3}{4}x + 1$$

which we will use to generate a sample of training data $(X_i, Y_i)_{i\in[m]}$ i.i.d. according to:

$$X_i \sim \text{Unif}([-1,1]) \tag{2.1}$$

$$Y_i = g(X_i) + \frac{1}{20}\zeta_i \tag{2.2}$$

where $\zeta_i \sim \mathcal{N}(0,1)$ is independent of X_i.

QUESTION 3. — Define a `array` `x` containing $m = 15$ real numbers drawn uniformly from $[-1,1]$. You should use the function `np.random.random_sample`.

QUESTION 4. — Define a `array` `y` containing the values $(Y_i)_{i\in[m]}$ defined according to (2.2). You should use the function `np.random.randn`.

QUESTION 5. — Visualize the generated data $(X_i, Y_i)_{i\in[m]}$ as blue points in the plane.

QUESTION 6. — Using the same procedure, generate a test sample $(X'_i, Y'_i)_{i\in[m']}$ of size $m' = 30$, which will be stored in `arrays` `x_test` and `y_test`.

2.2 Linear Regression

We will use the `scikit-learn` library to construct predictors from the training data. In `scikit-learn`, each predictor has a `fit` function that specifies the training data, and a `predict` function that, once the predictor is trained, computes the predictor's prediction on new inputs.
Here, we aim to construct the linear (i.e., affine) predictor that minimizes the least squares error. In other words, the class of predictors considered is:

$$\mathcal{F} = \{f_{a,b} : x \mapsto ax + b\}_{a,b\in\mathbb{R}},$$

and we seek to compute the minimizer of the empirical risk for the loss function $\ell(y, y') = (y - y')^2$:

$$\hat{f} = \operatorname*{arg\,min}_{f\in\mathcal{F}} \left\{ \sum_{i=1}^{m} (Y_i - f(X_i))^2 \right\}.$$

To do this, we will use the `LinearRegression` module, which we load as follows.

```
from sklearn.linear_model import LinearRegression
f = LinearRegression()
```

We have initialized a linear regression predictor `f` above. We now need to provide the training data using the `f.fit` function. This function requires that the input data $(X_i)_{i\in[m]}$ be presented as a 2-dimensional `array` with m rows (one per data point) and 1 column (since the input space $\mathcal{X} = \mathbb{R}$ has dimension 1). However, `x` has only one dimension. Therefore, we will define `X` so that it contains the values of `x` in the correct format, and do the same for `x_test`:

```
X = x[:, np.newaxis]
X_test = x_test[:, np.newaxis]
```

We can now specify the training data.

```
f.fit(X, y)
```

If we write $\hat{f}(x) = \hat{a}x + \hat{b}$, the coefficients $\hat{a}$ and $\hat{b}$ of the predictor $\hat{f}$ are given by `f.coef_` and `f.intercept_`, respectively. The predictions of the predictor $\hat{f}$ on the training data and test data are given by:

```
f.predict(X)
f.predict(X_test)
```

Question 7. — Produce a figure that overlays the points $(X_i, Y_i)_{i\in[m]}$ with the line corresponding to the graph of the predictor $\hat{f}$.

Question 8. — Calculate the average training error and the average test error. Comment.

2.3 Polynomial Regression

Let $n \geqslant 1$ be an integer. We now consider the least squares regression on polynomial functions of degree at most n. In other words, we consider the class of predictors:

$$\mathcal{F}_n^{\text{poly}} = \left\{ f_{a_0,\dots,a_n} : x \mapsto \sum_{k=0}^{n} a_n x^n \right\}_{(a_0,\dots,a_n)\in\mathbb{R}^{n+1}}.$$

We wish to compute the predictor:

$$\hat{f} := \operatorname*{arg\,min}_{f\in\mathcal{F}_n^{\text{poly}}} \left\{ \sum_{i=1}^{m} (Y_i - f(X_i))^2 \right\}.$$

We will see that this problem can be reduced to a linear regression in dimension n. Consider the map $\psi : \mathbb{R} \to \mathbb{R}^n$ defined by:

$$\psi(x) = (x, x^2, \dots, x^n).$$

For coefficients $(a_0, \dots, a_n) \in \mathbb{R}^{n+1}$, let $\mathbf{a} = (a_1, \dots, a_n) \in \mathbb{R}^n$, and the corresponding polynomial $f_{a_0,\dots,a_n}$ can then be written as:

$$f_{a_0,\dots,a_n}(x) = \langle \psi(x), \mathbf{a} \rangle + a_0.$$

Let $\mathcal{X}' = \mathbb{R}^n$ and $\mathcal{F}'$ be the class of linear (i.e., affine) predictors on $\mathcal{X}'$ to $\mathbb{R}$:

$$\mathcal{F}' = \left\{ f'_{(a_0,\dots,a_n)} : (x_1, \dots, x_n) \mapsto \sum_{k=1}^{n} a_k x_k + a_0 \right\}.$$

Let $\hat{f}'$ be the predictor obtained by linear regression on the data $(\psi(X_i), Y_i)_{i\in[m]} \in (\mathcal{X}' \times \mathbb{R})^m$:

$$\hat{f}' = \operatorname*{arg\,min}_{f'\in\mathcal{F}'} \left\{ \sum_{i=1}^{m} (Y_i - f'(\psi(X_i)))^2 \right\}.$$

Then, we have the relationship:

$$\hat{f} = \hat{f}' \circ \psi.$$

In other words, polynomial regression (on $\mathcal{X} = \mathbb{R}$) can be computed using linear regression in a higher dimension (on $\mathcal{X}' = \mathbb{R}^n$).

Question 9. — In this question, we want to calculate the predictor $\hat{f}$ for $n = 2$. We will implement the function ψ introduced above using tools provided by `scikit-learn`.

```
from sklearn.preprocessing import PolynomialFeatures
psi = PolynomialFeatures(2, include_bias=False).fit_transform
```

a) Compute the predictor $\hat{f}$ using `LinearRegression()`.

b) Overlay the graph of $\hat{f}$ with the points representing the training data.

c) Calculate the average training and test errors, and compare them with the errors obtained in the previous section.

Question 10. — Now we want to generalize the computation for any $n \geqslant 1$ (which corresponds to the maximal degree of the polynomials).

a) Inspired by the previous question, write a function that takes the integer n as an argument and returns two elements: the predictor $\hat{f}$ and the function ψ.

b) Compute the predictor for $n \in \{3, 4, 13, 14\}$ and visualize it (you may need to specify limits for the abscissa).

c) For $n = 3$, compare the coefficients of the predictor with those of the function g that generated the data.

d) What happens for $n = 14$?

e) Plot a figure with the abscissa $n = 1, \ldots, 14$ and the ordinate the graphs of the average training and test errors. Comment.

2.4 LASSO Regularization (Optional)

We now consider polynomial regression with least squares and LASSO regularization:

$$\hat{f} = \underset{f \in \mathcal{F}_n^{\text{poly}}}{\arg\min} \left\{ \frac{1}{2m} \sum_{i=1}^{m} (Y_i - f(X_i))^2 + \alpha \sum_{k=1}^{n} |a_k| \right\},$$

where $\alpha > 0$ is a parameter to be chosen.
With `sklearn`, we initialize a linear regression with LASSO regularization as follows:

```
from sklearn.linear_model import Lasso
f = Lasso(0.1)
```

where the argument corresponds to the choice of the α parameter.

Question 11. — Inspired by the previous section, compute for $n = 14$, polynomial regression with LASSO regularization for different values of the parameter (e.g., $\alpha \in \{10^{-6}, 10^{-5}, \ldots, 10^{-1}, 1\}$), and visualize the results. What do we observe about the obtained polynomials? About their coefficients? Their degrees?

Question 12. — Plot the training and test errors for the different values of α (we will use a logarithmic scale on the abscissa to improve readability). Comment.

Part II
K-Nearest Neighbors

We will work with Ronald Fischer's dataset, dating back to 1936, which contains data on iris flowers.

```
import numpy as np
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt

iris = sns.load_dataset("iris")
```

`iris` contains the data in the form of a DataFrame: this is a data type provided by the Pandas library, which allows storing data in the form of a table where the columns are named, and provides a wide range of functions for exploring and manipulating this data. We can display the first 5 rows with the following command.

```
iris.head()
```

We see that the first 4 columns correspond to the sepal length, sepal width, petal length, and petal width (all these measurements are in centimeters). These 4 variables will form the input. The fifth column corresponds to the iris species (which can be: Setosa, Versicolor, or Virginica) and is the output. This is a classification problem: we want to build a predictor that predicts the species based on the other characteristics. Furthermore, the `DataFrame` contains 150 rows: this is the number of data points.
We first want to simplify the dataset by reducing the dimensionality of the input space from 4 to 2. The goal is to determine which two explanatory variables seem most promising for predicting the species. To do this, we will use the `sns.pairplot` function from the Seaborn library.

```
sns.set()
sns.pairplot(iris, hue="species");
plt.show()
```

Question 13. — Choose the two explanatory variables that seem most promising for predicting the species, that is, the ones that, in the figure, seem to best separate the different species. Create an `array X` containing the columns corresponding to the two selected explanatory variables, and an `array y` containing the column corresponding to the species (we will need to use `iris.values`, which returns all the data from the `DataFrame iris` as a `NumPy array`).
We now wish to split the dataset into two samples: a training set of size 90, and a test set of size 60. One idea would be to use the first 90 data points to form the training sample and the last 60 to form the test sample.

```
X_train = X[:90]
y_train = y[:90]
X_test = X[90:]
y_test = y[90:]
```

Question 14. — Observe the resulting samples and explain why they might be problematic.
We can address the previous issue in the following way.

```
from sklearn.utils import shuffle
X, y = shuffle(X,y)

X_train = X[:90]
y_train = y[:90]
X_test = X[90:]
y_test = y[90:]
```

We can now train a k-NN predictor, for example with $k = 3$.

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
```

The default distance used is the Euclidean distance. Classification predictors in `scikit-learn` provide a `.score` function (here `knn.score`), which gives the proportion of correct predictions on a provided sample.

QUESTION 15. — What is the relationship between the score defined above and the empirical risk?

QUESTION 16. — Calculate the k-NN predictor for $k \in \{1, \dots, 20\}$. Plot the score curves calculated on the training and test samples, respectively. Deduce a value of k that seems to have produced the best predictor.

QUESTION 17. — Using the code written earlier, define a function `best_knn_score` that takes two `arrays` `X` and `y` as arguments, builds the training and test samples, calculates the k-NN predictors for $k \in \{1, \dots, 20\}$, and finally returns the best score calculated on the test sample. Run this function several times. What do you notice? What is this due to?

QUESTION 18. — Write a function `best_knn_score_avg` that takes `arrays` `X` and `y` as arguments, runs `best_knn_score(X, y)` 100 times, and returns the average of the 100 obtained scores.
The k-NN algorithm is sensitive to the scale used for each explanatory variable. To demonstrate this, we will modify the scale of one explanatory variable and observe the consequence on the quality of the obtained predictor. We begin by creating a copy `X_` of `X`.

```
X_ = X.copy()
```

The command `X_ = X` would not create a copy, but simply a second name for the same object: modifications to `X_` would then also affect `X`.

QUESTION 19. — Modify the `array` `X_` by converting the data of one of the two explanatory variables from centimeters to meters. Does using `X_` instead of `X` affect the quality of the constructed predictors? Repeat this by converting the other explanatory variable to meters (while keeping the first one in centimeters).
When you want to ensure that the influence of an explanatory variable is neither too weak nor too strong due to its scale, it is recommended to normalize the data. In other words, for each explanatory variable, you apply an affine transformation so that the resulting values have a mean of zero and a standard deviation of 1. `scikit-learn` provides a function for this.

```
from sklearn import preprocessing
X_scaled = preprocessing.scale(X)
```

QUESTION 20. — Does normalizing the data result in a better outcome here?

QUESTION 21. — If we consider all 4 initially available explanatory variables, is the resulting predictor better?