TP SD204

Mathurin MASSIAS

February 5, 2017

Contents

1 PEP8

Please, please follow the coding conventions of PEP8. It makes code much easier to read. Among the most important points are:

- whitespaces around assignment (=), except inside function calls
- whitespaces after commas, but not before
- no whitespace immediately inside parenthesis, brackets or braces

Think about the person who will read your code (and in real life, this will be yourself most of the time) when you write things like:

```
In [ ]: #Question 2
        def solverInv(X,Y,n,p,lambd):
            XT=X.transpose()
            left=(XT.dot(np.linalg.inv(X.dot(XT)+lambd*np.eye(n)))).dot(Y) #resolution de l'equation de droite
            right=((np.linalg.inv(XT.dot(X)+lambd*np.eye(p))).dot(XT)).dot(Y) #resolution de l'equation de gau
            return np.allclose(left,right,1E-02)
```

2 Lists

To access the last element of a list `v`, Python provides the syntax `v[-1]`, which is shorter than `v[len(v) - 1]`.

You can reverse a list with `v[::-1]`. More specifically, the syntax `v[a:b:c]` (where a, b and c are integers) means: from index a (included) to index b (excluded). If a (resp. b) is not specified, it is interpreted as 0 (resp. `len(v)`) if c is positive. If c is negative, a and b default values are switched. So if `v = [1, 2, 3, 4, 5]`, then `v[:-4:-1]` is `[5, 4, 3]`.

Appending elements to a list is costly and lists are not optimized to be performed mathematical operations upon. On the contrary, `np.array`s are, so if you know in advance the number of values you are going to need to store (for example, the error value for a 10 fold cross validation), use a `np.array`.

3 print

Don't systematically print your computation results. If the question asks to center and reduce a dataset, print the means and variances of the predictors rather than the whole dataset. When you load a dataframe, `my_df.head()` gives a good glimpse at the first lines and avoid printing hundreds of lines.
What is the purpose of a cell like the following?

```
In [18]: #Variable à prédire (consomation mpg)
         y = [dataValue[k][0] for k in range(len(dataValue))];
         print('Variable à prédire (consomation mpg): %s' %y);
         #Variables explicatives
         #Filtre de features (mpg, origin, auto_name)
         feature_filter=[features_name[0], features_name[len(features_name)-2], features_name[len(features_nam
         X = data_auto.ix[:, [features_name[k] for k in range(len(features_name))
                              if features_name[k] not in feature_filter]].values;
         #Dimensions
         p = len(X[0]);
         n = len(X);
         print('Variables explicatives: %s' %X);
```

```
Variable à prédire (consomation mpg): [18.0, 15.0, 18.0, 16.0, 17.0, 15.0, 14.0, 14.0, 14.0, 15.0, 15.0, 14.0,
 26.0, 25.0, 24.0, 25.0, 26.0, 21.0, 10.0, 10.0, 11.0, 9.0, 27.0, 28.0, 25.0, 19.0, 16.0, 17.0, 19.0, 18.0, 14
 18.0, 22.0, 19.0, 18.0, 23.0, 28.0, 30.0, 30.0, 31.0, 35.0, 27.0, 26.0, 24.0, 25.0, 23.0, 20.0, 21.0, 13.0, 1
 13.0, 19.0, 15.0, 13.0, 13.0, 14.0, 18.0, 22.0, 21.0, 26.0, 22.0, 28.0, 23.0, 28.0, 27.0, 13.0, 14.0, 13.0, 1
 12.0, 13.0, 18.0, 16.0, 18.0, 18.0, 23.0, 26.0, 11.0, 12.0, 13.0, 12.0, 18.0, 20.0, 21.0, 22.0, 18.0, 19.0, 2
 19.0, 15.0, 24.0, 20.0, 11.0, 20.0, 19.0, 15.0, 31.0, 26.0, 32.0, 25.0, 16.0, 16.0, 18.0, 16.0, 13.0, 14.0, 1
 28.0, 24.0, 26.0, 24.0, 26.0, 31.0, 19.0, 18.0, 15.0, 15.0, 16.0, 15.0, 16.0, 14.0, 17.0, 16.0, 15.0, 18.0, 2
 24.0, 25.0, 24.0, 18.0, 29.0, 19.0, 23.0, 23.0, 22.0, 25.0, 33.0, 28.0, 25.0, 25.0, 26.0, 27.0, 17.5, 16.0, 1
 24.5, 29.0, 33.0, 20.0, 18.0, 18.5, 17.5, 29.5, 32.0, 28.0, 26.5, 20.0, 13.0, 19.0, 19.0, 16.5, 16.5, 13.0, 1
 17.5, 17.0, 15.5, 15.0, 17.5, 20.5, 19.0, 18.5, 16.0, 15.5, 15.5, 16.0, 29.0, 24.5, 26.0, 25.5, 30.5, 33.5, 1
 32.8, 39.4, 36.1, 19.9, 19.4, 20.2, 19.2, 20.5, 20.2, 25.1, 20.5, 19.4, 20.6, 20.8, 18.6, 18.1, 19.2, 17.7, 1
 23.2, 23.8, 23.9, 20.3, 17.0, 21.6, 16.2, 31.5, 29.5, 21.5, 19.8, 22.3, 20.2, 20.6, 17.0, 17.6, 16.5, 18.2, 1
 27.4, 25.4, 23.0, 27.2, 23.9, 34.2, 34.5, 31.8, 37.3, 28.4, 28.8, 26.8, 33.5, 41.5, 38.1, 32.1, 37.2, 28.0, 2
 32.2, 46.6, 27.9, 40.8, 44.3, 43.4, 36.4, 30.0, 44.6, 33.8, 29.8, 32.7, 23.7, 35.0, 32.4, 27.2, 26.6, 25.8, 2
 37.7, 34.1, 34.7, 34.4, 29.9, 33.0, 33.7, 32.4, 32.9, 31.6, 28.1, 30.7, 25.4, 24.2, 22.4, 26.6, 20.2, 17.6, 2
 36.0, 37.0, 31.0, 38.0, 36.0, 36.0, 36.0, 34.0, 38.0, 32.0, 38.0, 25.0, 38.0, 26.0, 22.0, 32.0, 36.0, 27.0, 2
Variables explicatives: [[    8.    307.    130.   3504.     12.     70. ]
 [    8.    350.    165.   3693.     11.5    70. ]
 [    8.    318.    150.   3436.     11.     70. ]
 ...,
 [    4.    135.     84.   2295.     11.6    82. ]
 [    4.    120.     79.   2625.     18.6    82. ]
 [    4.    119.     82.   2720.     19.4    82. ]]
```

To remain concise when printing floats, the number of decimals displayed by numpy can be set through `np.set_printoptions()`.

4 Python 2 vs Python 3

In Python 2, `1 / 2` is `0` (euclidean division). To avoid this, either make at least one of the arguments a float (`1. / 2` or `1 / 2.`), or use `from __future__ import division`. Euclidean division may harm you silently when mutliplying something by `1 / n`, which is `0` in Python 2.

5 numpy

Rather than using `X.transpose()`, use the much shorter syntax `X.T`. Also note that this operation is "free" (it does not create a new array), so there is no need to create a new variable (but modifying `X.T` modifies `X`).

Unless you want to do broadcasting (which is a bit complicated at first but extremely powerful), most of the time there is no need to have vectors of shape `(n, 1)`: `(n, )` is enough.

When computing the eigenvalues of a hermitian matrix, use `np.linalg.eigh` to ensure that the result is real valued. Numerical error may result in complex eigenvalues otherwise, whose imaginary parts are discarded when plotted, resulting on plots who look noisy.

Random generators of numpy can generate N-dimensionals arrays, so there is no need to simulate a flat array and to reshape it later. This overcomplicates things:

```
In [ ]: X = random.normal(0,4,size = n*p);
        X = X.reshape(n,p);
```

The previous snippet uses semi-colons which are very unPythonic and useless at the end of lines. The only acceptable usecase for them is to put multiple statements on the same line (using `%timeit` for example).

6 Reinventing the wheel

Unless explicitly asked, do not reimplement basic function (vector norm, distance between vectors...): try to find a built-in function which does what you want. It'll save you time and makes it easier for the reader. Please avoid this:

```
In [ ]: #function inverse to get the inverse matrix
        def inverse(a):
            s=np.linalg.svd(a, full_matrices=1, compute_uv=1)
            U=s[0]
            diag=s[1]
            V=s[2]
            _diag=1./diag

            a_i=np.dot(np.dot((V.T),np.diag(_diag)),(U.T))
            return(a_i)
```

If you think that a function is commonly used (and the question does not ask you to implement it yourself), then look for a built-in version rather than recoding you own.

7 Variable names

Use explicit variable names (avoid naming your function `iteration()`); even if it's a temporary variable used for debugging, use a reasonable name. It's best to code in english, but if you code in french, don't mix languages. Use short variable names, but don't use cryptic

abbreviations (or if you must, at least explain the name in a comment).

Use snake_case (words in lower case, words separated by _) for variables. Functions should have names reflecting what they do, instead of `question_24()`. Only classes should have names in CamelCase (capitalized words joined). So call your function `plot_convergence` instead of `PlotConvergence`.

Don't prefix all your variable names with _ inside your function: Python implements scoping (values not returned are not accessible outside of function), so there is no need for "private variables" inside functions.

8 Comments

It's very nice to specify the number of a question in a comment at the beginning of a cell. Comments are most often welcome, but make them useful... What do you think of the following one?

```
In [ ]: egaliteGauche = np.dot(np.inv(np.dot(X.T, X) + p * alpha * np.eye(p)),
                                np.dot(X.T, y)) # égalité de gauche
```

9 Warnings

Do not ignore warnings, they are there for a reason. If it's a deprecation warning (in sklearn for example), then use the new submodule/function/syntax suggested in the warning. If it's a warning because casting complex values to real discards the imaginary part, ask yourself how you ended up with imaginary values. If it's a division by zero warning, did you divide something by 0 on purpose? Warnings are here to help...

10 for loops

Use them. Avoid the following: not only is tedious to go through dozens of lines like this, but if you want to change something, you must change it 4 or 8 times.

```
In [ ]: U, s1, V = np.linalg.svd(X1, full_matrices=False)
        U, L1, V =np.linalg.svd(np.dot(np.transpose(X1),X1)/n)
        U, s2, V = np.linalg.svd(X2, full_matrices=False)
        U, L2, V =np.linalg.svd(np.dot(np.transpose(X2),X2)/n)
        U, s3, V = np.linalg.svd(X3, full_matrices=True)
        U, L3, V =np.linalg.svd(np.dot(np.transpose(X3),X3)/n)
        U, s4, V = np.linalg.svd(X4, full_matrices=False)
        U, L4, V =np.linalg.svd(np.dot(np.transpose(X4),X4)/n)
```

11 Plots

Give them names, label the axes, and most importantly comment them in a Markdown cell. They are plotted to emphasize on something.

12 import

Put all of your imports in a cell at the beginning of your notebook. Do not use `import *`: when you use an imported function, the reader must know where it comes from. It is also dangerous to `%pylab inline`, since it overrides silently built-in functions such as `all()`. Do not reimport modules at the beginning of each question, once is enough.