MACHINE LEARNING IN PYTHON: A TUTORIAL v0.4 (20211122)

*Giorgio De Nunzio – University of Salento and INFN branch of Lecce*

This is a simple tutorial[[1]](#footnote-1) about machine learning in python. It will use several python packages (imported into our code as modules), in particular sklearn.[[2]](#footnote-2)

Open a windows or linux or mac shell.

Go to the directory (folder) where your work will be; in my case this is in the Documents folder:

C:\Users\Giorgio De Nunzio\Documents\MACHINE\_LEARNING\_COURSE\

Create a directory named PROJECTS! Do it from the operating system GUI, or

mkdir PROJECTS

Then enter this directory from the command line with

cd PROJECTS

In what follows I suppose you have a 3.5+ version of python (maybe 2.7 will do). The anaconda environment[[3]](#footnote-3) is a good choice but also vanilla python will be ok.

Create a python virtual environment[[4]](#footnote-4); maybe (if you have several versions of python or the one you want is not in the path) you have to fully name the python path:

"C:\Users\Giorgio De Nunzio\AppData\Local\Programs\Python\Python39\python" -m venv ml\_venv

or maybe (if python is in the path, as it should) it is enough to give:

python -m venv ml\_venv

or

python3 -m venv ml\_venv

This command creates a directory called ml\_env which contains some basic python stuff and where the packages (i.e. libraries) we need will be installed.

The equivalent command for Anaconda (from an Anaconda prompt) is conda create --name ml\_venv while deleting an environment is done by conda env remove --name ml\_venv

Anaconda builds the virtual environments in its directories (in Windows, it is somewhere like C:\Users\...\.conda\envs\ml\_venv, so no ml\_venv subdir will be visible in the current folder.

When working in Anaconda, don’t forget[[5]](#footnote-5) to update the framework from time to time with

conda update conda

conda update python (update)

or

conda install python=3.9 (upgrade between major python version)

after starting the anaconda prompt as an administrator. It might be better to do upgrades in a virtual environments instead of the base one.

If after activating the virtual environment in Anaconda, you get this warning:

*Warning:*

*This Python interpreter is in a conda environment, but the environment has*

*not been activated. Libraries may fail to load. To activate this environment*

*please see https://conda.io/activation*

you can try conda install pip

Back to vanilla python, if we take a look with dir ml\_venv or ls ml\_venv we see something like:

02/10/2021 11:28 <DIR> .

02/10/2021 11:28 <DIR> ..

02/10/2021 11:28 <DIR> Include

02/10/2021 11:28 <DIR> Lib

02/10/2021 11:28 132 pyvenv.cfg

02/10/2021 11:28 <DIR> Scripts

Activate the virtual environment:

* Linux: source ml\_venv/bin/activate
* Windows: ml\_venv\Scripts\activate
* Anaconda: conda activate ml\_venv

(when necessary, we deactivate with: deactivate)

The prompt is now:

(ml\_venv)

Let us install the necessary packages in this environment (when the environment is activated, the “right” python should be in the path and be called, so no path in python call should be needed, but check).[[6]](#footnote-6) When using Anaconda, equivalent commands exist which are issued as conda install something, so check for your environment. Moreover, many library are already installed in Anaconda and similar frameworks, so, again, check! In what follows I will generally consider a naïve python environment.

python -m pip install --upgrade pip

python -m pip install numpy scipy matplotlib pandas sklearn mlxtend seaborn

Anaconda:

conda install numpy scipy matplotlib pandas seaborn scikit-learn xgboost

conda install mlxtend --channel conda-forge

Remark the following message I got from Anaconda when installing scikit-learn:

*Windows 64-bit packages of scikit-learn can be accelerated using scikit-learn-intelex.*

*More details are available here: https://intel.github.io/scikit-learn-intelex*

*For example:*

*$ conda install scikit-learn-intelex*

*$ python -m sklearnex my\_application.py*

Not investigated yet!

I recommend working directly in the interpreter or writing your scripts and running them on the command line rather than using IDEs. Keep things simple and focus on machine learning, not the toolchain.

Start python.

(ml\_venv) C:\Users\Giorgio De Nunzio\Documents\MACHINE\_LEARNING\_COURSE\PROJECTS>python

Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit (AMD64)] on win32

Type "help", "copyright", "credits" or "license" for more information.

>>>

Important: you have several choices[[7]](#footnote-7) to give commands to python, e.g.:

1. from the python command line, just type in, or copy/paste: python is an interpreter so it will reply command per command, preserving in memory the variables you create; this way you can follow a step-by-step incremental way of working
2. from the os shell, with python ./path/to/script.py
3. from script files: insert groups of commands in files, to be put in your working directory, then call them from within the python shell with[[8]](#footnote-8):

exec(open("./path/to/script.py").read(), globals())

This will execute a script and put all its global variables in the interpreter's global scope (the normal behavior in most scripting environments).

Let’s start!

Give the following commands, which import and then check the library versions. You do not need to give the commands which have the hash symbol # because they are just comments to remind us what we are doing. Typing is better, but copy-paste if you are lazy: anyway be sure to read each command carefully to sufficiently understand it.

# import the library modules

import sys

import scipy

import numpy

import matplotlib

import pandas

import sklearn # scikit-learn

import seaborn

# Check the version of some libraries

print('Python: {}'.format(sys.version))

print('scipy: {}'.format(scipy.\_\_version\_\_))

print('numpy: {}'.format(numpy.\_\_version\_\_))

print('matplotlib: {}'.format(matplotlib.\_\_version\_\_))

print('pandas: {}'.format(pandas.\_\_version\_\_))

print('sklearn: {}'.format(sklearn.\_\_version\_\_))

print('seaborn: {}'.format(seaborn.\_\_version\_\_))

This was my output:

>>> print('Python: {}'.format(sys.version))

Python: 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit (AMD64)]

>>> print('scipy: {}'.format(scipy.\_\_version\_\_))

scipy: 1.7.1

>>> print('numpy: {}'.format(numpy.\_\_version\_\_))

numpy: 1.21.2

>>> print('matplotlib: {}'.format(matplotlib.\_\_version\_\_))

matplotlib: 3.4.3

>>> print('pandas: {}'.format(pandas.\_\_version\_\_))

pandas: 1.3.3

>>> print('sklearn: {}'.format(sklearn.\_\_version\_\_))

sklearn: 1.0

>>> print('seaborn: {}'.format(seaborn.\_\_version\_\_))

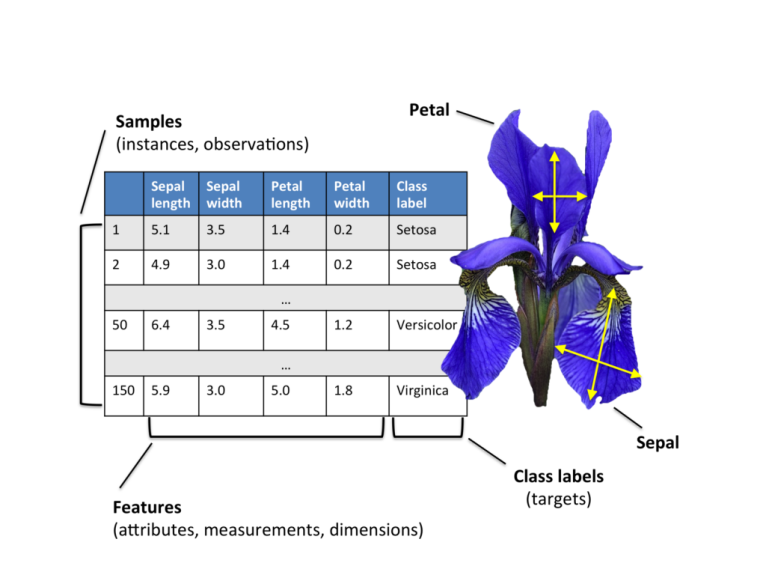
seaborn: 0.11.2

>>>

For our first test, we are going to use the iris flowers dataset (also called Fisher's Iris data set), https://en.wikipedia.org/wiki/Iris\_flower\_data\_set.

This dataset is famous because it is used as the “hello world” dataset in machine learning and statistics by pretty much everyone.

The dataset contains 150 observations of iris flowers. There are four columns of measurements of the flowers (petal and sepal length and width) in centimeters. The fifth column is the species of the flower observed. All observed flowers belong to one of three species (setosa, versicolor or virginica).



(from https://www.intelligenzaartificialeitalia.net/post/applicazione-svm-per-la-classificazione-automatica-su-iris-dataset-esempi-pratici-machine-learning)

First, let’s import all of the modules, functions and objects we are going to use in this tutorial.

# Load library functions

from pandas import read\_csv # pandas manage "dataframes"

from pandas.plotting import scatter\_matrix

from matplotlib import pyplot # I had already imported matplotlib

from matplotlib.pyplot import scatter # or simply use pyplot.scatter or matplotlib.pyplot.scatter

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import StratifiedKFold

from sklearn.metrics import classification\_report

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import accuracy\_score

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis

from sklearn.naive\_bayes import GaussianNB

from sklearn.svm import SVC

We can load the data directly from a Machine Learning repository. The file is in csv (comma-separated values) format.

We are using pandas to load the data.[[9]](#footnote-9) Pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool. We will also use pandas next to explore the data both with descriptive statistics and data visualization.

Note that we are specifying the names of each column when loading the data, with the names argument, because there is no header in the csv file. This will help later when we explore the data. The column names, respectively 'sepal-length', 'sepal-width', 'petal-length', 'petal-width', and 'class', are available on the site where the dataset was downloaded from.

Note: if the file contains a header row (suppose the first row), then you should explicitly pass header=0 to override the column names.

# Load dataset

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"

names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']

dataset = read\_csv(url, names=names)

If you prefer, you can download the iris.csv file into your working directory and load it using the same method, changing URL to the local file name.

Interlude: list directory, change directory.

import os

os.listdir()

os.chdir('newdir')

Interlude: strings. Strings in python are surrounded by either single quotation marks, or double quotation marks: 'hello' is the same as "hello". Python strings are immutable: if you want to change a string, consider passing through a list or choose other peculiar ways:[[10]](#footnote-10)

text = 'abcdefg'

new = list(text)

new[6] = 'W'

''.join(new)

or

text = 'abcdefg'

text = text[:1] + 'Z' + text[2:]

etc.

Interlude: lists and tuples.[[11]](#footnote-11) list and tuple are classes of data structures similar to arrays of other languages, but more flexible. Both lists and tuples are used to store multiple items (objects or values, of arbitrary and even inhomogeneous type) in one variable and are defined by enclosing a comma-separated sequence of objects into respectively square brackets and parentheses. Lists and tuples are two of four built-in Python data types used to store collections of data, the other two are Set, and Dictionary, all with different qualities and usage.

Lists (square brackets): thislist = ["apple", "banana", "cherry"]

Tuples (parentheses): mytuple = ('hello', 5, 3.4)

Lists and tuples need not be homogeneous: mylist = ['hello', 5, 3.4]

Both lists and tuples are ordered and allow duplicate values. The only important difference between lists and tuples is that the former are changeable while the latter are immutable (we cannot modify them).

>>> thislist = ["apple", "banana", "cherry"]

>>> mylist = ['hello', 5, 3.4]

>>> thistuple = ("apple", "banana", "cherry")

>>> mytuple = ('hello', 5, 3.4)

>>> mylist[1] = 'mickey mouse'

>>> mylist

['hello', 'mickey mouse', 3.4]

>>> mytuple[1] = 'mickey mouse'

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: 'tuple' object does not support item assignment

A (small) difference is that [123] is a list containing just one element, while (123) is number 123, so to have a tuple of 1 element we have a strange syntax with a comma:

>>> m = (123) # I wish a tuple with just one element, but this is not:

>>> type(m)

<class 'int'>

>>> n = [123] # on the other hand, a list with only one element is easy :-)

>>> type(n)

<class 'list'>

>>> m = (123,) # ok this is the trick for a tuple with one element

>>> type(m)

<class 'tuple'>

Here are some examples of lists and list processing. Except for mutability and for the use of parentheses instead of brackets, what we say will be valid for tuples too. By the way, as there is a strong similarity between lists and strings, what we say here will be valid for strings too, but only considering that Python strings are immutable!

>>> a = ['foo', 'bar', 55, numpy.nan, False, 55] # inhomogeneous, repeated objects

>>> print(a) # or simply: a

['foo', 'bar', 55, nan, False, 55]

Lists are ordered:

>>> [1, 2, 3, 4] == [4, 1, 3, 2]

False

A list can even contain functions, classes, modules... List length is arbitrary (within memory limits).

List items are indexed, the first item has index [0], the second item has index [1] etc.

>>> a[1]

'bar'

Negative indices count from the end of the list:

>>> a[-1]

55

Slicing:

if *a* is a list, the expression a[m:n] returns the portion of a from index m to, but not including, index n;

a[:n] starts from the beginning and is equivalent to a[0:n];

a[2:] goes to the end of the list and means a[2:len(a)]

>>> a

['foo', 'bar', 55, nan, False, 55]

>>> a[2:5]

[55, nan, False]

>>> a[:5]

['foo', 'bar', 55, nan, False]

>>> a[2:]

[55, nan, False, 55]

>>> len(a)

6

You can specify a stride (i.e. a step in slicing), either positive or negative, as the third value in a slicing expression:

>>> a[0:6:2]

['foo', 55, False]

>>> a[1:6:2]

['bar', nan, 55]

>>> a[6:0:-2]

[55, nan, 'bar']

>>> a[::-1] # returns the reversed list!

[55, False, nan, 55, 'bar', 'foo']

>>> a[:] # returns a copy of the a list

CAVEAT: for a string s, the command s[:] is a reference to THE SAME string!

List of lists:

>>> w = [['a', 'b', 'c'], ['1', '2', '3']]

>>> len(w) # 'number of rows'

2

>>> len(w[0]) # 'number of columns'

3

For other details on lists (and tuples) go through the web page suggested in the note.

We can see from the import statements, that read\_csv() is imported from pandas, so the dataset variable is a pandas dataframe:

type(dataset)

<class 'pandas.core.frame.DataFrame'>

After loading, we can take a look at the variable by simply naming the pandas dataframe or by the print command:

>>> dataset

sepal-length sepal-width petal-length petal-width class

0 5.1 3.5 1.4 0.2 Iris-setosa

1 4.9 3.0 1.4 0.2 Iris-setosa

2 4.7 3.2 1.3 0.2 Iris-setosa

3 4.6 3.1 1.5 0.2 Iris-setosa

4 5.0 3.6 1.4 0.2 Iris-setosa

.. ... ... ... ... ...

145 6.7 3.0 5.2 2.3 Iris-virginica

146 6.3 2.5 5.0 1.9 Iris-virginica

147 6.5 3.0 5.2 2.0 Iris-virginica

148 6.2 3.4 5.4 2.3 Iris-virginica

149 5.9 3.0 5.1 1.8 Iris-virginica

[150 rows x 5 columns]

print(dataset.head(20)) allows to see the first 20 lines, with tail we see the last 20 lines.

Moreover, dataset['sepal-length'] or dataset['class'] show the columns with those names.

We also get some interesting info by the vars function:

>>> vars(dataset)

{'\_is\_copy': None, '\_mgr': BlockManager

Items: Index(['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class'], dtype='object')

Axis 1: RangeIndex(start=0, stop=150, step=1)

NumericBlock: slice(0, 4, 1), 4 x 150, dtype: float64

ObjectBlock: slice(4, 5, 1), 1 x 150, dtype: object, '\_item\_cache': {'class': 0 Iris-setosa

1 Iris-setosa

2 Iris-setosa

3 Iris-setosa

4 Iris-setosa

...

145 Iris-virginica

146 Iris-virginica

147 Iris-virginica

148 Iris-virginica

149 Iris-virginica

Name: class, Length: 150, dtype: object, 'sepal-length': 0 5.1

1 4.9

2 4.7

3 4.6

4 5.0

...

145 6.7

146 6.3

147 6.5

148 6.2

149 5.9

Name: sepal-length, Length: 150, dtype: float64}, '\_attrs': {}, '\_flags': <Flags(allows\_duplicate\_labels=True)>, 'plot': <pandas.plotting.\_core.PlotAccessor object at 0x00FCAF70>}

>

The column names are listed in “Items”. How can I get a list of the used labels (i.e. classes)?

>>> numpy.unique(dataset['class'])

array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)

If I wish to avoid to prefix the function with the module it is in, simply:

>>> from numpy import unique

then

>>> unique(dataset['class'])

Now, what if I want to get the dataset lines which belong to class 'Iris-setosa'? Before all consider this Boolean expression:

>>> dataset['class'] == 'Iris-setosa'

which gives a Series containing only Boolean values:

0 True

1 True

2 True

3 True

4 True

...

145 False

146 False

147 False

148 False

149 False

Name: class, Length: 150, dtype: bool

>>> type(dataset['class'] == 'Iris-setosa')

<class 'pandas.core.series.Series'>

so we can use this series to “slice” the dataset, extracting the lines of class Iris-setosa:

>>> dataset[dataset['class'] == 'Iris-setosa']

sepal-length sepal-width petal-length petal-width class

0 5.1 3.5 1.4 0.2 Iris-setosa

1 4.9 3.0 1.4 0.2 Iris-setosa

2 4.7 3.2 1.3 0.2 Iris-setosa

3 4.6 3.1 1.5 0.2 Iris-setosa

<omissis>

47 4.6 3.2 1.4 0.2 Iris-setosa

48 5.3 3.7 1.5 0.2 Iris-setosa

49 5.0 3.3 1.4 0.2 Iris-setosa

Finally, if I want only one feature, e.g. petal-width, of the lines belonging to class Iris-setosa, I can ask for:

>>> dataset['petal-width'][dataset['class'] == 'Iris-setosa']

or (order does not count in this case):

>>> dataset[dataset['class'] == 'Iris-setosa']['petal-width']

0 0.2

1 0.2

2 0.2

3 0.2

<omissis>

47 0.2

48 0.2

49 0.2

Name: petal-width, dtype: float64

This will be useful when plotting the data.

Remark that as it is, this would not be a binary classification problem but a multiclass one (three classes).

Also remark that the label (or target) variable is a string; more often it is a numeric or Boolean label, e.g. 0/1.

In binary problems it is customary to consider one class as “positive” and the other as “negative” (e.g. to distinguish a tumor- from a non-tumor lesion). In that case the 0 label is generally assigned to the negative case, and the 1 label to the positive case.

And now, before machine learning, here is some exploratory data analysis.

Before all, what is the shape of our dataframe?

print(dataset.shape)

gives a tuple: (number of lines, number of columns), i.e. the dataset dimensions.

>>> print(dataset.shape)

(150, 5)

Then, a statistical summary of the dataframe content:

>>> print(dataset.describe())

sepal-length sepal-width petal-length petal-width

count 150.000000 150.000000 150.000000 150.000000

mean 5.843333 3.054000 3.758667 1.198667

std 0.828066 0.433594 1.764420 0.763161

min 4.300000 2.000000 1.000000 0.100000

25% 5.100000 2.800000 1.600000 0.300000

50% 5.800000 3.000000 4.350000 1.300000

75% 6.400000 3.300000 5.100000 1.800000

max 7.900000 4.400000 6.900000 2.500000

>>>

(we get a per-column description).

Class Distribution:

>>> print(dataset.groupby('class').size())

class

Iris-setosa 50

Iris-versicolor 50

Iris-virginica 50

dtype: int64

Again a statistical summary, but per class

>>> print(dataset.groupby('class').describe())

sepal-length sepal-width ... petal-length petal-width

count mean std min 25% 50% 75% max count mean ... 75% max count mean std min 25% 50% 75% max

class ...

Iris-setosa 50.0 5.006 0.352490 4.3 4.800 5.0 5.2 5.8 50.0 3.418 ... 1.575 1.9 50.0 0.244 0.107210 0.1 0.2 0.2 0.3 0.6

Iris-versicolor 50.0 5.936 0.516171 4.9 5.600 5.9 6.3 7.0 50.0 2.770 ... 4.600 5.1 50.0 1.326 0.197753 1.0 1.2 1.3 1.5 1.8

Iris-virginica 50.0 6.588 0.635880 4.9 6.225 6.5 6.9 7.9 50.0 2.974 ... 5.875 6.9 50.0 2.026 0.274650 1.4 1.8 2.0 2.3 2.5

[3 rows x 32 columns]

The same, but setting full dataframe print:

>>> pandas.set\_option('display.max\_columns', None)

>>> print(dataset.groupby('class').describe())

sepal-length \

count mean std min 25% 50% 75% max

class

Iris-setosa 50.0 5.006 0.352490 4.3 4.800 5.0 5.2 5.8

Iris-versicolor 50.0 5.936 0.516171 4.9 5.600 5.9 6.3 7.0

Iris-virginica 50.0 6.588 0.635880 4.9 6.225 6.5 6.9 7.9

sepal-width \

count mean std min 25% 50% 75% max

class

Iris-setosa 50.0 3.418 0.381024 2.3 3.125 3.4 3.675 4.4

Iris-versicolor 50.0 2.770 0.313798 2.0 2.525 2.8 3.000 3.4

Iris-virginica 50.0 2.974 0.322497 2.2 2.800 3.0 3.175 3.8

petal-length \

count mean std min 25% 50% 75% max

class

Iris-setosa 50.0 1.464 0.173511 1.0 1.4 1.50 1.575 1.9

Iris-versicolor 50.0 4.260 0.469911 3.0 4.0 4.35 4.600 5.1

Iris-virginica 50.0 5.552 0.551895 4.5 5.1 5.55 5.875 6.9

petal-width

count mean std min 25% 50% 75% max

class

Iris-setosa 50.0 0.244 0.107210 0.1 0.2 0.2 0.3 0.6

Iris-versicolor 50.0 1.326 0.197753 1.0 1.2 1.3 1.5 1.8

Iris-virginica 50.0 2.026 0.274650 1.4 1.8 2.0 2.3 2.5

>>> pandas.reset\_option('display.max\_rows')

Interlude: full dataframe print. If we want to print full, non-truncated, dataframe content with print(a) or by naming a, for a dataframe with many lines, you can give:[[12]](#footnote-12)

pandas.set\_option('display.max\_rows', None)

followed, after printing, by:

pandas.reset\_option('display.max\_ columns')

Similarly, if the dataframe has too many columns but python is only printing part of them, use:

pandas.set\_option('display.max\_rows', None)

and

pandas.reset\_option('display.max\_columns')

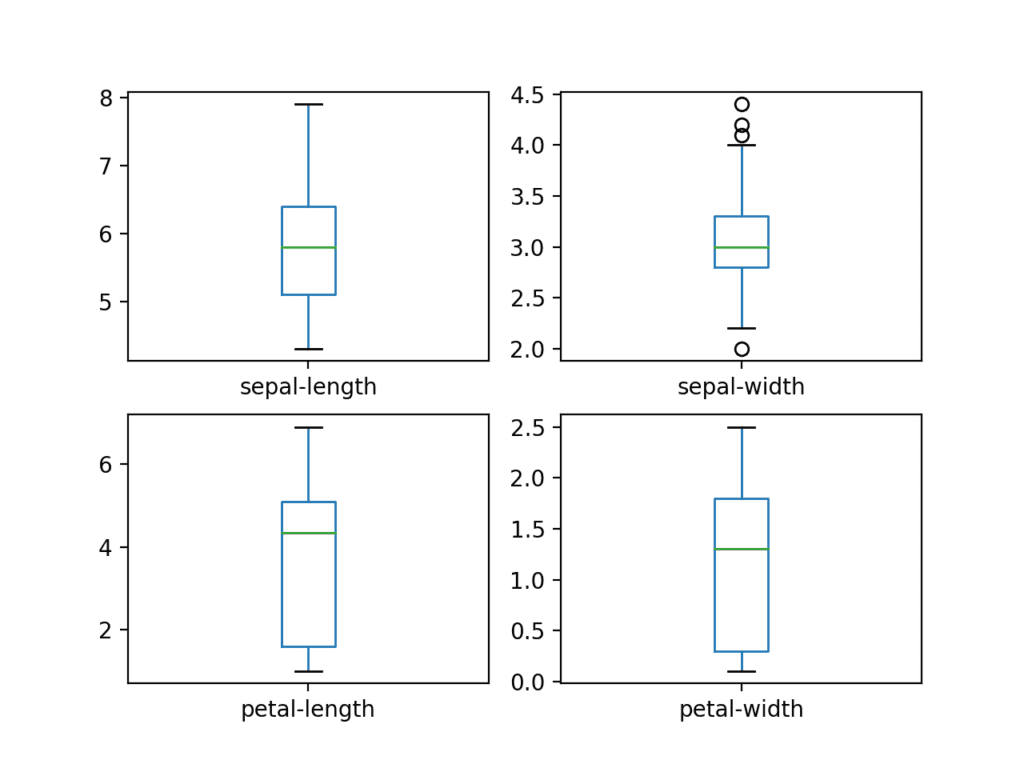
Data Visualization: Univariate plots to better understand each (numeric) attribute (feature)

# box and whisker plots

dataset.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False)

pyplot.show()

Compare the box plots with the dataset.describe() output.



Some keys and mouse clicks/drag&drop change the appearance.

You have to close the figure to get control of the shell again. Graphs can be saved by the disk icon: different formats are available with different resolutions...

On the other hand, in order to have high-quality images, you can[[13]](#footnote-13) use a python command such as:

pyplot.savefig("filename.svg")

pyplot.savefig('filename.pdf')

pyplot.savefig('filename.png', dpi=300)

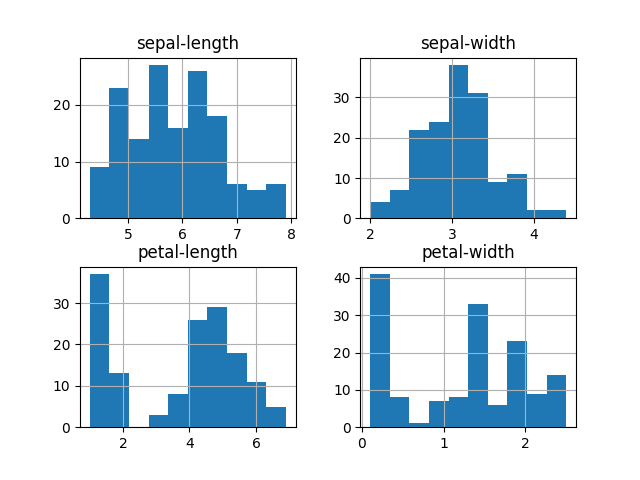
inserting the command before show().

We can also create a histogram of each input variable to get an idea of the distribution.

# histograms

dataset.hist()

pyplot.show()



It looks like two of the input variables have a Gaussian-like distribution. This is useful to note as we can use algorithms that can exploit this assumption.

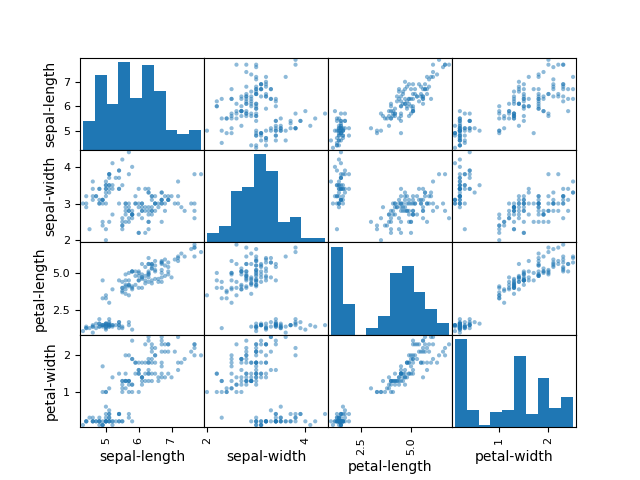
Data Visualization: Multivariate plots to better understand the relationships between attributes

Scatterplots of all pairs of attributes (without class distinction). This can be helpful to spot structured relationships between input variables. We can use pyplot graphic functions which come from matplotlib.[[14]](#footnote-14)

# scatter plot matrix

scatter\_matrix(dataset)

pyplot.show()



Note the diagonal grouping of some pairs of attributes. This suggests a high correlation between those features.

Now let us examine the data after splitting by class. The first command shows the scatter plot of petal length and petal width for setose (in red), the second command does the same for versicolor (in blue). So we have two scatter plots in the same figure.

scatter(dataset[dataset['class'] == 'Iris-setosa']['petal-width'], dataset[dataset['class'] ==

'Iris-setosa']['petal-length'], marker='D', color='red', label='setosa') # label, goes to the legend

scatter(dataset[dataset['class'] == 'Iris-versicolor']['petal-width'], dataset[dataset['class'] ==

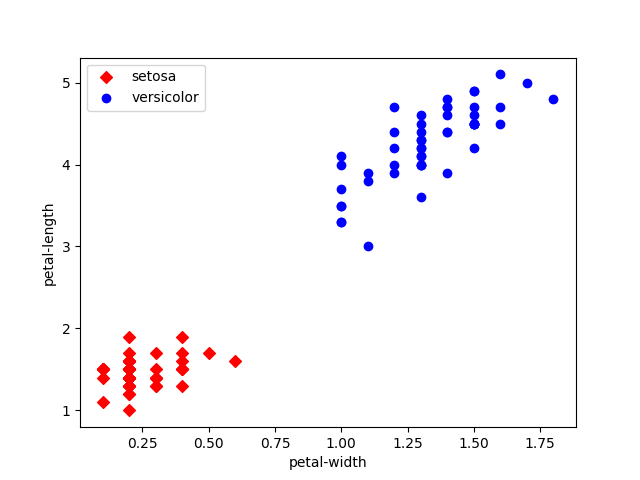
'Iris-versicolor']['petal-length'], marker='o', color='blue', label='versicolor')

pyplot.xlabel('petal-width')

pyplot.ylabel('petal-length')

pyplot.legend() # I did not say: from matplotlib.pyplot import legend, so I need to fully qualify,

pyplot.show() # otherwise legend() would do; the same for show(), xlabel()...



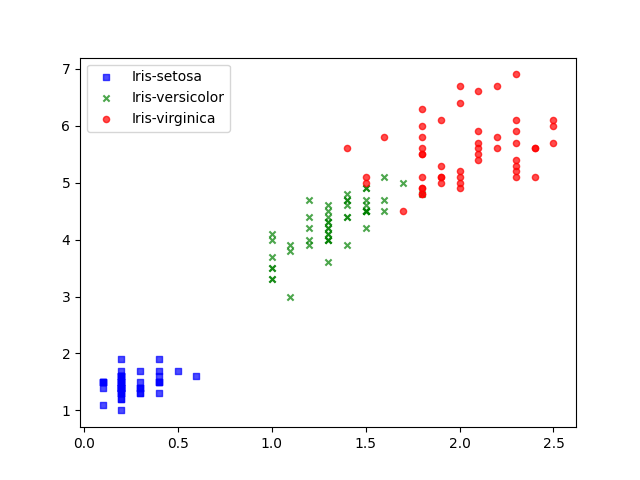
The above scatter plot could be achieved in one line by using the category\_scatter function from mlxtend python package authored by Dr. Sebastian Raschka. Here is the command (for the three classes):

import mlxtend

from mlxtend.plotting import category\_scatter

category\_scatter(x='petal-width', y='petal-length', label\_col='class', data=dataset, legend\_loc='upper left')

pyplot.show()



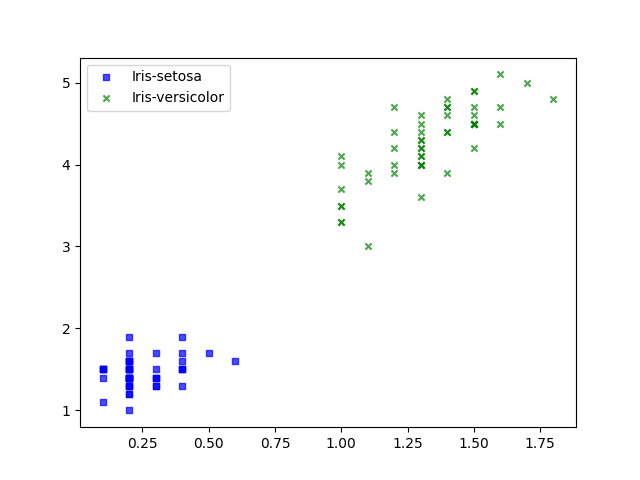
Or, for two classes:

category\_scatter(x='petal-width', y='petal-length', label\_col='class',

data=dataset[(dataset['class'] == 'Iris-setosa') | (dataset['class'] == 'Iris-versicolor')],

legend\_loc='upper left')

pyplot.show()



Interlude: figures.

Instead of creating a figure and displaying it with a global pyplot.show(), we can create and save it to an object and invoke its show() method. In this way we maintain control of the shell even if the figure is still open. Nonetheless, sometimes this gives strange behaviour...

fig = category\_scatter(x='petal-width', y='petal-length', label\_col='class',

data=dataset[(dataset['class'] == 'Iris-setosa') | (dataset['class'] == 'Iris-versicolor')],

legend\_loc='upper left')

fig.show()

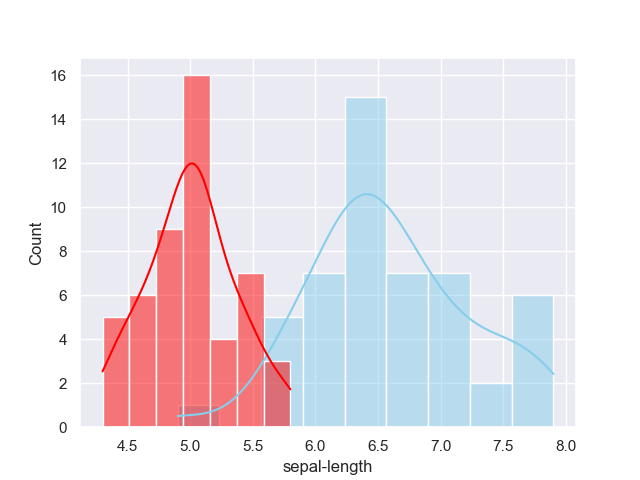
Another option to get some nice graphics is to use seaborn.[[15]](#footnote-15)

seaborn.histplot(data=dataset[dataset['class']=='Iris-virginica'], x="sepal-length", color="skyblue", label="Virginica", kde=True)

seaborn.histplot(data=dataset[dataset['class']=='Iris-setosa'], x="sepal-length", color="red", label="Setosa", kde=True)

pyplot.xlabel('sepal-length')

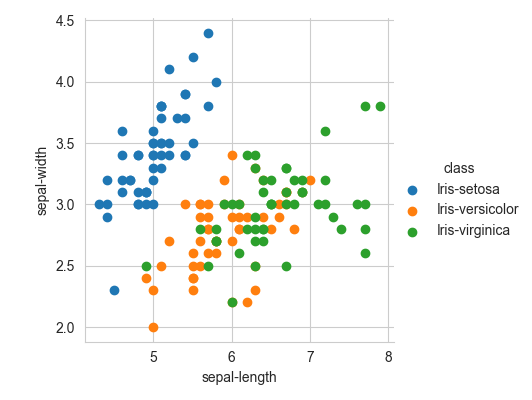
pyplot.show()



import seaborn

seaborn.set\_style("whitegrid")

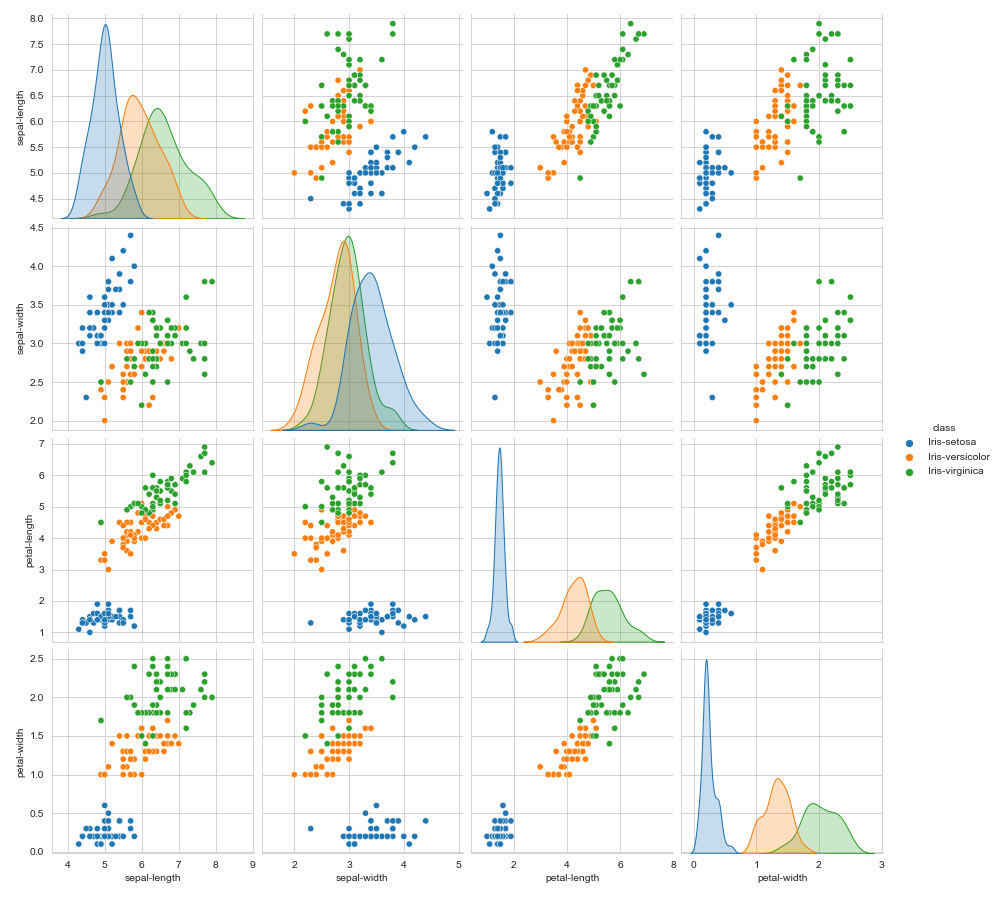
seaborn.FacetGrid(dataset,hue="class",height=4).map(pyplot.scatter,"sepal-length","sepal-width").add\_legend()  
pyplot.show()



seaborn.set\_style("whitegrid");  
seaborn.pairplot(dataset,hue="class",height=2); # height is the height in inches of each facet

pyplot.savefig('img.png', dpi=75) # save

pyplot.show()



From the above plots we can conclude that petal length and petal width are two features which can separate the data very well.

After data visualization, let us finally start doing some machine learning.

Summing up (in case we exited python for any reason, and we got in again) here are the essential commands:

import sys

import scipy

import numpy

import matplotlib

import pandas

import sklearn

import mlxtend

import seaborn

from pandas import read\_csv

from pandas.plotting import scatter\_matrix

from matplotlib import pyplot # I had already imported matplotlib

from matplotlib.pyplot import scatter # or simply use pyplot.scatter or matplotlib.pyplot.scatter

from mlxtend.plotting import category\_scatter

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import StratifiedKFold

from sklearn.metrics import classification\_report

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import accuracy\_score

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis

from sklearn.naive\_bayes import GaussianNB

from sklearn.svm import SVC

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"

names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']

dataset = read\_csv(url, names=names)

(By the way, everyone can decide for him/herself how to organize imports: for example, you can say:

import sklearn

from sklearn.model\_selection import train\_test\_split

and then use the train\_test\_split() function, or you can only give the first of the two commands, and then use:

sklearn.model\_selection.train\_test\_split()

The two ways are perfectly equivalent, as we can see with:

>>> sklearn.model\_selection.train\_test\_split

<function train\_test\_split at 0x114449B8>

>>> train\_test\_split

<function train\_test\_split at 0x114449B8>

>>>

Moreover, people often like to shorten library names like this:

import numpy as np

so that objects and functions can be called in a shortened way, e.g. np.nan or np.unique() instead of numpy.nan etc. Another frequent example is import pandas as pd.

Shortcuts should be avoided if there is some risk of confusion.

Interlude: indentation.[[16]](#footnote-16) Python indentation is a way of telling python that a group of statements belongs to a particular block of code. A block is a combination of equally indented consecutive statements. Other languages such as C, C++, Java use braces { } to define a block of code. Python uses indentation. Whitespace is used for indentation in Python. All statements with the same distance to the right belong to the same block of code. If a block has to be more deeply nested, it is simply indented further to the right.

Examples:

# Python program showing indentation

site = 'gfg'

if site == 'gfg':

print('Logging on to geeksforgeeks...')

else:

print('retype the URL.')

print('All set !')

..........

j = 1

while(j<= 5):

print(j)

j = j + 1

..........

Remark: “bulk” copy/paste from a file to the command line is not always perfect: it is sometimes necessary to insert a blank line just after an indented block.

Interlude: the for loop. Tutorial from the web pages in the notes.[[17]](#footnote-17)

Dataset preparation for binary classification

To simplify things, let us reduce the three-class problem to a binary one.

In what follows, after data splitting into training and validation sets, we shall train a classifier and calculate ROC (Receiver Operating Characteristics) curves so to assess quality by the area under the ROC curve (AUC, or ROC-AUC), and by a confusion matrix obtained at a specific threshold.

Information about the ROC curve and its calculation in python and sklearn can be easily found.[[18]](#footnote-18)

Many other interesting pages in the Web contain discussions about the nature and the problems related to ROC curves.[[19]](#footnote-19)

Let’s start: reduce the problem to a binary one, by only using two of the classes, in particular 'Iris-virginica' and 'Iris-setosa'.

a = dataset[(dataset['class'] == 'Iris-virginica') | (dataset['class'] == 'Iris-setosa')].copy()

Note: in the preceding command, the copy() function is necessary to avoid the SettingWithCopy warning:[[20]](#footnote-20)

Note: I cannot use the syntax dataset.class (attribute access, with the dot) but I must use dataset['class'] (indexing with square brackets) because ‘class’ is a python keyword, so the former command would give syntax error![[21]](#footnote-21)

Now let’s take a look at variable a. In particular its shape, and what happened to the first column of the dataset. Numbers between 49 and 100 excluded were cut out (they were versicolor iris) but the column was not “renumbered” from 0 to 99! This happens because the first axis in a dataframe is a label, not an index.

>>> a.shape

(100, 5)

>>> print(a)

sepal-length sepal-width petal-length petal-width class

0 5.1 3.5 1.4 0.2 Iris-setosa

1 4.9 3.0 1.4 0.2 Iris-setosa

2 4.7 3.2 1.3 0.2 Iris-setosa

3 4.6 3.1 1.5 0.2 Iris-setosa

4 5.0 3.6 1.4 0.2 Iris-setosa

5 5.4 3.9 1.7 0.4 Iris-setosa

6 4.6 3.4 1.4 0.3 Iris-setosa

7 5.0 3.4 1.5 0.2 Iris-setosa

8 4.4 2.9 1.4 0.2 Iris-setosa

9 4.9 3.1 1.5 0.1 Iris-setosa

10 5.4 3.7 1.5 0.2 Iris-setosa

11 4.8 3.4 1.6 0.2 Iris-setosa

12 4.8 3.0 1.4 0.1 Iris-setosa

13 4.3 3.0 1.1 0.1 Iris-setosa

14 5.8 4.0 1.2 0.2 Iris-setosa

15 5.7 4.4 1.5 0.4 Iris-setosa

16 5.4 3.9 1.3 0.4 Iris-setosa

17 5.1 3.5 1.4 0.3 Iris-setosa

18 5.7 3.8 1.7 0.3 Iris-setosa

19 5.1 3.8 1.5 0.3 Iris-setosa

<omissis>

41 4.5 2.3 1.3 0.3 Iris-setosa

42 4.4 3.2 1.3 0.2 Iris-setosa

43 5.0 3.5 1.6 0.6 Iris-setosa

44 5.1 3.8 1.9 0.4 Iris-setosa

45 4.8 3.0 1.4 0.3 Iris-setosa

46 5.1 3.8 1.6 0.2 Iris-setosa

47 4.6 3.2 1.4 0.2 Iris-setosa

48 5.3 3.7 1.5 0.2 Iris-setosa

49 5.0 3.3 1.4 0.2 Iris-setosa

100 6.3 3.3 6.0 2.5 Iris-virginica

101 5.8 2.7 5.1 1.9 Iris-virginica

102 7.1 3.0 5.9 2.1 Iris-virginica

103 6.3 2.9 5.6 1.8 Iris-virginica

104 6.5 3.0 5.8 2.2 Iris-virginica

105 7.6 3.0 6.6 2.1 Iris-virginica

106 4.9 2.5 4.5 1.7 Iris-virginica

107 7.3 2.9 6.3 1.8 Iris-virginica

108 6.7 2.5 5.8 1.8 Iris-virginica

<omissis>

139 6.9 3.1 5.4 2.1 Iris-virginica

140 6.7 3.1 5.6 2.4 Iris-virginica

141 6.9 3.1 5.1 2.3 Iris-virginica

142 5.8 2.7 5.1 1.9 Iris-virginica

143 6.8 3.2 5.9 2.3 Iris-virginica

144 6.7 3.3 5.7 2.5 Iris-virginica

145 6.7 3.0 5.2 2.3 Iris-virginica

146 6.3 2.5 5.0 1.9 Iris-virginica

147 6.5 3.0 5.2 2.0 Iris-virginica

148 6.2 3.4 5.4 2.3 Iris-virginica

149 5.9 3.0 5.1 1.8 Iris-virginica

>>>

Actually, if you wish to select a particular line (let’s take the last one), you can use:[[22]](#footnote-22)

>>> a.iloc[99]

sepal-length 5.9

sepal-width 3.0

petal-length 5.1

petal-width 1.8

class Iris-virginica

Name: 149, dtype: object

with iloc which is primarily integer position based (from 0 to length-1 of the axis), or

>>> a.loc[149]

sepal-length 5.9

sepal-width 3.0

petal-length 5.1

petal-width 1.8

class Iris-virginica

Name: 149, dtype: object

with loc which is primarily label position based (this is not an index along the axis).

Both commands access the last line of the file: as the number 99, or as the one labelled as 149 (because with that label it was created).

Use .iloc or .loc with single brackets to pull out a Series, or with double brackets to extract a DataFrame:

>>> a.iloc[[99]]

sepal-length sepal-width petal-length petal-width class

149 5.9 3.0 5.1 1.8 Iris-virginica

>>> a.loc[[149]]

sepal-length sepal-width petal-length petal-width class

149 5.9 3.0 5.1 1.8 Iris-virginica

Finally, if you wish a range of rows, you must use the single brackets syntax and you always get dataframes. Nonetheless, note the difference:

>>> a.iloc[10:12]

sepal-length sepal-width petal-length petal-width class

10 5.4 3.7 1.5 0.2 Iris-setosa

11 4.8 3.4 1.6 0.2 Iris-setosa

>>> a.loc[10:12]

sepal-length sepal-width petal-length petal-width class

10 5.4 3.7 1.5 0.2 Iris-setosa

11 4.8 3.4 1.6 0.2 Iris-setosa

12 4.8 3.0 1.4 0.1 Iris-setosa

i.e. contrary to usual Python slices, both the start and the stop are included, when present in the index, for .loc!

Now convert[[23]](#footnote-23) ‘Iris-virginica’ and ‘Iris-setosa’ class names to respectively label 0 and label 1; this is not necessary, but some people are accustomed to it and allows to decide that one class is positive, the other one is negative, so that terms such as TP, FP, etc have a meaning.

Moreover it will be handy when comparing actual labels (classes) of validation datasets with scores as given by a classifier, because scores are usually numeric values between 0 and 1.

a.replace(to\_replace = ['Iris-virginica', 'Iris-setosa'], value = [0,1], inplace=True) # gives 0 / 1

Remark that this command:

a['class'] = a['class'] == 'Iris-virginica'

would give False / True in the class column, in case we like this way of naming the class.

Now, the class column of “a” contains 0’s and 1’s. Before feeding the dataset to the classifier, just take the numpy.array part and then split into X and Y, and then into training and validation.

Data splitting into train/validation(/test)

The simplest form of data splitting for model construction and validation is called hold-out. We subdivide the dataset into two parts, typically 80% for training and 20% for validation (by someone called “test”). We do not set apart data for true final testing.

We will use sklearn.model\_selection.train\_test\_split.[[24]](#footnote-24)

a = a.values # extract the numpy.ndarray containing the data

X = a[:,0:4] # features: slicing with 0:4, the selected interval is [0,4) (np.ndarray)

Y = a[:,4] # targets (classes, labels) (np.ndarray)

X\_train, X\_validation, Y\_train, Y\_validation = train\_test\_split(X, Y, test\_size=0.20, random\_state=1)

We used python slicing[[25]](#footnote-25) to select the columns in the NumPy array (the target classes into Y, the feature columns into X).

In train\_test\_split we set the random seed via the random\_state argument to a fixed number to ensure that each run on any algorithm is evaluated on the same splits of the training dataset (for repeatability).

Consider that in some situations (when a “sample”, for example a patient when studying a medical problem, corresponds to multiple vectors of features, we cannot simply use train\_test\_split() because we would possibly put vectors belonging to the same patient in both train and validation/test sets, with an obvious statistical bias (remember that in ML model creation the training set must be held completely separate from the validation and test datasets!).

and now:

>>> a.shape # a is a numpy.ndarray, it has a shape, just like pandas dataframes; lists and tuples have length: len([1,2,5])

(100, 5)

>>> X\_train.shape

(80, 4)

>>> Y\_train.shape

(80,)

>>> X\_validation.shape

(20, 4)

>>> Y\_validation.shape

(20,)

Remark that numpy array shapes are returned as python tuples which, unlike python lists, can't straightforwardly be written down with a single entry: (442) would just evaluate to the integer 422 , unlike [422]. The extra comma is just an aspect of python tuple syntax for single-element tuples to distinguish them from integers. Something more about tuples and lists, in an “interlude”.

Interlude: numpy ndarrays. We can easily verify that the dataset variable is a complex thing defined in pandas while a is a numpy n-dimensional array or numpy.ndarray:[[26]](#footnote-26)

type(dataset)

<class 'pandas.core.frame.DataFrame'>

type(a)

<class 'numpy.ndarray'>

NumPy main object is the homogeneous multidimensional array. It is a table of elements (usually numbers), all of the same type, indexed by a tuple of non-negative integers. In NumPy, dimensions are called axes.

For example, the coordinates of a point in 3D space, such as [1, 2, 1], have one axis. That axis has 3 elements in it, so we say it has a length of 3. In the example pictured below, the array has 2 axes. The first axis has a length of 2 (rows), the second axis has a length of 3 (columns).

[[1., 0., 0.],

[0., 1., 2.]]

Numpy.ndarray objects are created with functions such as array(), zeros() or empty() (or directly from the ndarray class):

ar = [1,2,3] # create a list (also works with a tuple)

ar = numpy.array(ar) # convert to nparray

ar

array([1, 2, 3])

type(ar)

<class 'numpy.ndarray'>

ar[1]

2

ar[1:2]

array([2])

ar[1:3] # the rightmost value of the range is not included

array([2, 3])

We now have training data in the X\_train and Y\_train variables for training the models, and X\_validation and Y\_validation sets to verify the ongoing optimization.

Define the classifier.[[27]](#footnote-27)

# Define a MultiLayer Perceptron (The defaut hidden\_layer\_sizes is (100,) which means only 1 hidden layer with 100 neurons)

from sklearn.neural\_network import MLPClassifier

from sklearn.metrics import roc\_curve, roc\_auc\_score, average\_precision\_score

#myMLP = MLPClassifier(hidden\_layer\_sizes=(), max\_iter=5000, activation = 'logistic', solver='sgd')

#myMLP = MLPClassifier(hidden\_layer\_sizes=(), max\_iter=5000, activation = 'relu', solver='adam')

myMLP = MLPClassifier(hidden\_layer\_sizes=(3), max\_iter=300, activation = 'relu', solver='adam')

#myMLP = MLPClassifier(hidden\_layer\_sizes=(10,), max\_iter=300, activation = 'relu', solver='adam', random\_state=1)

#myMLP = MLPClassifier(hidden\_layer\_sizes=(3,), max\_iter=1000, solver='sgd', tol=1e-4,

# random\_state=1,learning\_rate\_init=1, learning\_rate='adaptive') # alpha=1e-4,

myMLP.verbose = 1

# Train

myMLP.fit(X\_train, Y\_train)

# attributes of the trained classifier

print('current loss computed with the loss function: ', myMLP.loss\_)

print('coefs: ', myMLP.coefs\_)

print('intercepts: ', myMLP.intercepts\_)

print('number of iterations of the solver: ', myMLP.n\_iter\_)

print('num of layers: ', myMLP.n\_layers\_)

print('Num of o/p: ', myMLP.n\_outputs\_)

# Calculate predictions (classes, after thresholding!!)

Y\_predicted = myMLP.predict(X\_validation) # Y\_predicted.size is 20

#Comparing the predictions (with threshold=0.5) against the actual observations in Y\_validation

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(Y\_predicted, Y\_validation)

#Printing the accuracy

print("Accuracy of MLPClassifier with threshold set to 0.5: ", cm.trace()/cm.sum())

# ROC curves for each binary classification

Y\_probab = myMLP.predict\_proba(X\_validation) # this is a (30,2) ndarray

fpr2, tpr2, threshold = roc\_curve(Y\_validation, Y\_probab[:,1])

# Calculate the area under the roc curve

print("ROC-AUC of MLPClassifier: ",roc\_auc\_score(Y\_validation, Y\_probab[:,1]))

# Finally plot the ROC curve

pyplot.plot(fpr2, tpr2, color = "red")

pyplot.xlim([-0.01,1.01])

pyplot.ylim([-0.01,1.01])

pyplot.xlabel("FPR")

pyplot.ylabel("TPR")

pyplot.show()

We take the second column of Y\_probab because the score needed to compute the ROC is the probability of the positive class, which is the second element of predict\_proba output.

Some parameters[[28]](#footnote-28) of MLPClassifier() are:

hidden\_layer\_sizes : This parameter allows us to set the number of layers and the number of nodes we wish to have in the Neural Network Classifier. Each element in the tuple represents the number of nodes at the ith position where i is the index of the tuple. Thus the length of tuple denotes the total number of hidden layers in the network.

max\_iter: It denotes the number of epochs.

activation: The activation function for the hidden layers.

solver: This parameter specifies the algorithm for weight optimization across the nodes.

random\_state: The parameter allows to set a seed for reproducing the same results

In case you get this message:

"ConvergenceWarning: Stochastic Optimizer: Maximum iterations (1000) reached and the optimization hasn't converged yet. ConvergenceWarning)"

A convergence point is a machine learning model localized optimal state (a point of minimum in loss/error). It basically means that the variables (the weights, for a MLP) within the model have the best possible values (within a certain vicinity) in order to predict the target. Machine learning algorithms might not be able to find a convergence point. In this case the hyperparameters (number of hidden layers, number of neurons, the solver, the learning rate...) should be tuned (e.g. by increasing the number of epochs, by adding momentum, etc).

Now add feature standardization.[[29]](#footnote-29)

Standardize features by removing the mean and scaling to unit variance.

The standard score of a sample x is calculated as:

z = (x - u) / s

where u is the mean of the training samples or zero if with\_mean=False, and s is the standard deviation of the training samples or one if with\_std=False.

Centering and scaling happen independently on each feature by computing the relevant statistics on the samples in the training set. Mean and standard deviation are then stored to be used on validation using function transform() (which is necessary to avoid a statistical bias[[30]](#footnote-30))! Basically fit\_transform() is a fit(), i.e. the calculation of mean and standard deviation, followed by transform(), which applies the transformation.

Before altering X\_train and X\_validation, let us save a copy somewhere. We can either use some temporary variables:

X\_train\_2 = X\_train

X\_validation\_2 = X\_validation

or use the pickle() command,[[31]](#footnote-31) which saves to disk (“serialization”):

import pickle

with open('XYtrain.pickle', 'wb') as file\_handle: # python 2: 'w'

pickle.dump([X\_train, X\_validation], file\_handle, protocol=pickle.HIGHEST\_PROTOCOL)

where the protocol argument is there for optimization (file size) and in case you wish to save a single variable, you need no list. Check that you have saved a file!

Remember when necessary the dir() command to see the defined variables, and the os.dirlist() command to see the folder directory.

You can later reload by:

with open('XYtrain.pickle', 'rb') as file\_handle: # Python 2: remove 'rb'

X\_train, X\_validation = pickle.load(file\_handle)

What are the mean and std before rescaling?

>>> X\_train.mean(axis=0)

array([5.755 , 3.19625, 3.40875, 1.1025 ])

>>> X\_validation.mean(axis=0)

array([5.965, 3.195, 3.905, 1.265])

>>> X\_train.std(axis=0)

array([0.96370379, 0.37464775, 2.09100417, 0.9251993 ])

>>> X\_validation.std(axis=0)

array([0.82053336, 0.54357612, 2.00586016, 0.85864719])

Now, let us rescale!

from sklearn.preprocessing import StandardScaler

sc = StandardScaler() # instantiate an object of the StandardScaler class

type(sc)

vars(sc)

X\_train = sc.fit\_transform(X\_train)

vars(sc) # I can see that the mean and std have been computed; they are used in the next command

X\_validation = sc.transform(X\_validation)

Let us see the result:

>>> X\_train.mean(axis=0) # mean by rows

array([ 1.97480921e-15, -1.28369537e-16, 4.42701431e-16, -3.10862447e-16])

>>> X\_validation.mean(axis=0)

array([ 0.21790928, -0.00333647, 0.23732616, 0.17563783])

>>> X\_train.std(axis=0)

array([1., 1., 1., 1.])

>>> X\_validation.std(axis=0)

array([0.85143731, 1.4508992 , 0.95928081, 0.92806727])

Remark that the mean value and the std of the standardized validation vectors are not 0 and 1 respectively: this depends on the fact that the training subsample is not actually representative of the population.

As an alternative, it is possible to normalize the features to a range, say [0,1]. The library contains functions for this purpose.[[32]](#footnote-32)

Now let us put all the code (and more!) into a script and call it from python.

Before proceeding, if it was not done yet, install another classifier, XGBoost, in our python virtual environment. From the shell command line (not in python shell) give the following pip command:

pip install xgboost

and for Anaconda give

conda install xgboost

We shall talk about it later.

Now, create a file called callClassifier.py containing the following code:

# run this program from python with

# exec(open("callClassifier.py").read(), globals()) # python 3

# or from the os shell with

# python callClassifier.py

def callClassifier(whichCase = 0, whichModel = "MLP", standardizeData = False):

# various imports

import scipy

import numpy

import matplotlib

import pandas

import sklearn

import random # to build random colors for the plots

from pandas import read\_csv

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import classification\_report

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import accuracy\_score

from sklearn.neural\_network import MLPClassifier

from sklearn.metrics import roc\_curve, roc\_auc\_score, average\_precision\_score

from sklearn.svm import SVC

from xgboost import XGBClassifier

from matplotlib import pyplot

# Load dataset

if whichCase == 0:

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"

names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']

dataset = read\_csv(url, names=names)

dataset = dataset[(dataset['class'] == 'Iris-virginica') | (dataset['class'] == 'Iris-setosa')].copy()

dataset.replace(to\_replace = ['Iris-virginica', 'Iris-setosa'], value = [0,1], inplace=True) # gives 0 / 1

a = dataset.values # extract the numpy.ndarray containing the data

X = a[:,0:4] # features

Y = a[:,4] # targets (classes, labels)

elif whichCase == 1:

url = "archive-weight-height-sex--kaggle.csv"

dataset = read\_csv(url, header=0)

dataset.replace(to\_replace = ['Male', 'Female'], value = [0,1], inplace=True) # gives 0 / 1

a = dataset.values # extract the numpy.ndarray containing the data

X = a[:,1:3] # features

Y = a[:,0] # targets (classes, labels)

else:

print("Something is wrong, value of whichCase is not known\n")

return(-1)

X\_train, X\_validation, Y\_train, Y\_validation = train\_test\_split(X, Y, test\_size=0.20, random\_state=None) # an integer, or None

if standardizeData == 'True':

from sklearn.preprocessing import StandardScaler

sc = StandardScaler() # instantiate an object of the StandardScaler class

X\_train = sc.fit\_transform(X\_train) # train: standardize and get standardization parameters

X\_validation = sc.transform(X\_validation) # validation: just apply standardization parameters contains in sc

if whichModel == "MLP": # note that linear separation is enough (it is even better, in some cases, than having HLs)

# Define a MultiLayer Perceptron (The defaut hidden\_layer\_sizes is (100,) which means only 1 hidden layer with 100 neurons)

myModel = MLPClassifier(hidden\_layer\_sizes=(3), max\_iter=5000, activation = 'relu', solver='adam')

#myModel = MLPClassifier(hidden\_layer\_sizes=(), max\_iter=5000, activation = 'relu', solver='adam')

#myModel = MLPClassifier(hidden\_layer\_sizes=(3,), max\_iter=300, activation = 'relu', solver='adam')

#myModel = MLPClassifier(hidden\_layer\_sizes=(10,), max\_iter=300, activation = 'relu', solver='adam', random\_state=1)

#myModel = MLPClassifier(hidden\_layer\_sizes=(50,), max\_iter=10, alpha=1e-4,solver='sgd', tol=1e-4, random\_state=1,learning\_rate\_init=.1)

elif whichModel == "XGB":

myModel = XGBClassifier()

else:

print("Something is wrong, value of whichModel is not known\n")

return(-1)

myModel.verbose = 0 # 1

# Train

myModel.fit(X\_train, Y\_train)

# Calculate predictions

Y\_predicted = myModel.predict(X\_validation)

#Comparing the predictions against the actual observations in y\_val

cm = confusion\_matrix(Y\_predicted, Y\_validation)

#Printing the accuracy

print("Accuracy of Classifier with threshold set to 0.5: ", cm.trace()/cm.sum())

# ROC curves for each binary classification

Y\_probab = myModel.predict\_proba(X\_validation) # this is a (30,2) ndarray; 2 because it is a binary problem

fpr2, tpr2, threshold = roc\_curve(Y\_validation, Y\_probab[:,1])

# Calculate the area under the roc curve

auc = roc\_auc\_score(Y\_validation, Y\_probab[:,1])

print("ROC-AUC of Classifier: ", auc)

# Plot the ROC curve; note the varying color, the possibility to see everything on the same figure or not, the use of pause to avoid blocking, the axis limits

# https://stackoverflow.com/questions/28269157/plotting-in-a-non-blocking-way-with-matplotlib

# https://www.kite.com/python/answers/how-to-generate-a-random-color-for-a-matplotlib-plot-in-python#use-numpy-random-rand

r = random.random()

b = random.random()

g = random.random()

color = (r, g, b)

#pyplot.figure() # start a new fig https://stackoverflow.com/questions/6916978/how-do-i-tell-matplotlib-to-create-a-second-new-plot-then-later-plot-on-the-o

pyplot.plot(fpr2, tpr2, color = color) # "red"

pyplot.xlim([-0.01,1.01])

pyplot.ylim([-0.01,1.01])

pyplot.xlabel("FPR")

pyplot.ylabel("TPR")

#pyplot.draw()

pyplot.pause(0.001)

return auc

if \_\_name\_\_ == "\_\_main\_\_": # this is executed only if the script is called from the command line

import numpy

AUCs = []

for j in range(5): # 0 to n-1 (calculates the AUC n times)

# whichCase may be 0 or 1; whichModel may be "MLP" or "XGB"

AUCs.append(callClassifier(whichCase = 2, whichModel = "XGB", standardizeData = False))

print("Mean AUC is ", numpy.mean(AUCs))

print("AUC std is ", numpy.std(AUCs))

# pyplot.close('all') should close all the windows

The main code calls the callClassifier function a certain number of times and calculates the mean and std of the found AUCs. Remark that sometimes the MLP gives very low values of AUC (around 0.5), probably due to a false start (the initial work point of the network in terms of weights is random) or to badly configured hyperparameters.

Remark that another classifier was inserted in the code, XGBoost. The internal workings of XGBboost are difficult to explain in a small time, so please refer to Web sites[[33]](#footnote-33) and literature. Practical usage of XGBoost in the python environment is available online.[[34]](#footnote-34)

XGBoost is a very capable and fast classifier, which is able to digest data with missing or NaN values. To check this, open (locally) one of the two datasets and replace a value with NaN, in this way (see the file 5th line):

Gender,Height,Weight

Male,73.847017017515,241.893563180437

Male,68.7819040458903,162.310472521300

Male,74.1101053917849,212.7408555565

Male,71.7309784033377,nan

Male,69.8817958611153,206.349800623871

Male,67.2530156878065,152.212155757083

Then try to classify by the MLP. After receiving an error (except if in the meantime there was an update which fixes the problem...), try XGBoost. The choice of the classifier is done with one of the parameters of the function call.

Exercises

1. What if we try and distinguish setosa vs versicolor? Modify the callMLP() program and do it! Also add some graphing of the features (histograms of single features, and scatter plots of feature pairs, by classes). Also add a boxplot of the values of AUC.
2. What happens with the archive-weight-height-sex--kaggle.csv data? It relates the sex of a person to her/his height and weight (so, two features! The target class is sex).
   1. Check if the format is similar to the iris.csv dataset (it is similar but not identical): in this case, the csv file has a header line, so you have to drop the names=names argument and include the header=0 one:

Gender,Height,Weight

Male,73.847017017515,241.893563180437

Male,68.7819040458903,162.310472521300

..........

1. Add some graphing of the two features (two histograms and a scatter plot, by classes), and a final boxplot of the values of AUC.
2. Remark the change in the code where class strings are replaced by 0 and 1.

(by the way, this dataset is available on the kaggle web site but it is very strange, because features are listed with very high precision... this makes me think that it is a synthetic and not a real dataset...).

K-fold Cross Validation

K-fold Cross Validation is a training&validation technique in which the dataset (the feature matrix) is split into *k* parts; the model is then trained on a subset formed of *k*-1 parts and evaluated on the remaining one, then repeating the procedure by cycling on the validation part.

5-fold Cross Validation

The advantage of this scheme is that data are used in an efficient way, as each sample contributes both to training and to validation. This is particularly useful when the dataset available for learning is small-sized. In principle this method gives a better quality than simple hold-out because all the samples are used for training (though cyclically).

Quality assessment can be obtained in two different ways:

* by calculating quality figures at each iteration, then averaging (possible only when the validation part is large enough)
* by cumulating the score of all the validation parts, then calculating quality figures

In sklearn k-fold CV is obtained by several functions:[[35]](#footnote-35) KFold, StratifiedKFold, ShuffleSplit, StratifiedShuffleSplit.

The two methods with “Stratified” in their name use "stratified fold". It means each part preserves (if possible) the same percentage of samples of each class (label) as the original data (data stratification, i.e. distribution per classes, is respected). This is important when the classes are unbalanced.

In KFolds(), each validation set does not overlap, even with shuffle. With KFolds() and the shuffle = True option, the data is shuffled once at the start, and then divided into the number of desired splits. The validation data is one of the splits, the train data is the rest.

In ShuffleSplit(), the data is shuffled every time, and then split. This means the validation sets may overlap between the splits. Another use of this function, besides cross validation with shuffling, is simple hold-out, with k=2, as a substitute of train\_test\_split().

Anyway, always remember that when several vectors of features correspond to each single subject of the ML procedure (item, sample, patient...), a statistical bias must be avoided by creating the training and validation sets on a “per subject” basis, avoiding that a subject’s vectors are split between training and validation.

An example is a setting in which patients undergo repeated biopsy with several sampling sites, e.g. to detect a cancer: of course each patient has several feature vectors, each corresponding to a sampling site. In this case the mentioned functions must be adapted accordingly (see later for another comment o the subject).

In the following tests, we shall use StratifiedKFold.

Initialization commands:

import sys

import scipy

import numpy

import matplotlib

import pandas

import sklearn

import matplotlib

from pandas import read\_csv

from pandas.plotting import scatter\_matrix

from sklearn.neural\_network import MLPClassifier

from sklearn.metrics import roc\_curve, roc\_auc\_score

from matplotlib import pyplot

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"

names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']

dataset = read\_csv(url, names=names)

a = dataset[(dataset['class'] == 'Iris-virginica') | (dataset['class'] == 'Iris-setosa')].copy()

a.replace(to\_replace = ['Iris-virginica', 'Iris-setosa'], value = [0,1], inplace=True) # gives 0 / 1

a = a.values # extract the numpy.ndarray containing the data

X = a [:,0:4] # features: slicing with 0:4, the selected interval is [0,4) (np.ndarray)

Y = a [:,4] # targets (classes, labels) (np.ndarray)

myMLP = MLPClassifier(hidden\_layer\_sizes=(), max\_iter=500, activation = 'relu', solver='adam')

myMLP.verbose = 1

Now, the necessary import for this example is:

from sklearn.model\_selection import StratifiedKFold, cross\_val\_predict

Let us see how StratifiedKFold works.

Before all define some fake data:

fake\_X = numpy.array([

[1,2,3,4],

[11,12,13,14],

[21,22,23,24],

[31,32,33,34],

[41,42,43,44],

[51,52,53,54],

[61,62,63,64],

[71,72,73,74]

])

fake\_Y = numpy.array([0,0,0,0,1,1,1,1])

which represent respectively a matrix of feature vectors and the corresponding classes.

fake\_X.shape

(8, 4)

Now:

skfold = StratifiedKFold(n\_splits=4, shuffle=False)

for train, test in skfold.split(fake\_X, fake\_Y):

print('Train: %s | test: %s' % (train, test))

The result is in term of indices to fake\_X rows:

Train: [1 2 3 5 6 7] | test: [0 4]

Train: [0 2 3 4 6 7] | test: [1 5]

Train: [0 1 3 4 5 7] | test: [2 6]

Train: [0 1 2 4 5 6] | test: [3 7]

Again, if we are dealing with the multi-vector-per-subject case, these splits will be used to index subjects, not directly feature vectors, and each subject will call for its feature vectors (indirect indexing).

Let’s go back to the iris, where we had X and Y as the features and the target classes respectively (before holdout).

As already said, the quality of the trained classifier in case of k-fold CV can be measured in two main ways: either by averaging the quality figures obtained on the single splits,[[36]](#footnote-36) or by cumulating scores/probabilities/prediction of each validation sample and then calculating the quality figures on this overall set.

We wish to create a ROC curve from the cumulated results of training/validation on the various splits: to achieve this, we can use the cross\_val\_predict function with the method argument set to ‘proba’.[[37]](#footnote-37)

kfold = StratifiedKFold(n\_splits=5, random\_state=0, shuffle=True)

Y\_probab = cross\_val\_predict(myMLP, X, Y, cv=kfold, method='predict\_proba')

The Y\_probab variable is a two-column ndarray, in which the first column reports the probabilities of each validation vector to belong to class 0, the second column the same for class 1. We will use the second column, because it is high (about 1) when the sample probably belongs to class 1.

fpr2, tpr2, threshold = roc\_curve(Y, Y\_probab[:,1]) # Y are the original classes

Finally draw the ROC curve and calculate the AUC.

# Calculate the area under the roc curve

auc = roc\_auc\_score(Y, Y\_probab[:,1])

print("ROC-AUC of MLPClassifier: ", auc)

# Plot the ROC curve

pyplot.plot(fpr2, tpr2, color = "red")

pyplot.xlim([-0.01,1.01])

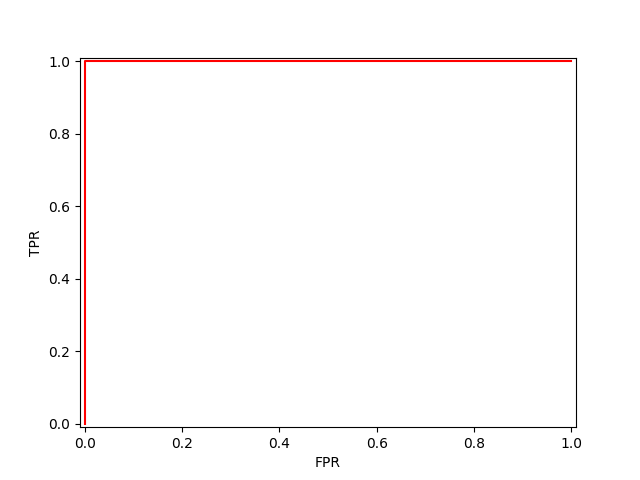
pyplot.ylim([-0.01,1.01])

pyplot.xlabel("FPR")

pyplot.ylabel("TPR")

pyplot.show()

With this scheme we are exploiting the whole dataset for training and for validation, moreover shuffling each time as to the starting point of the MLP. This may give us a more reliable estimation of classifier quality, and also allows us to train with more data.



>>> print("ROC-AUC of MLPClassifier: ", auc)

ROC-AUC of MLPClassifier: 0.916

This is the complete code:

import sys

import scipy

import numpy

import matplotlib

import pandas

import sklearn

import matplotlib

from pandas import read\_csv

from pandas.plotting import scatter\_matrix

from sklearn.neural\_network import MLPClassifier

from sklearn.metrics import roc\_curve, roc\_auc\_score

from matplotlib import pyplot

from sklearn.model\_selection import StratifiedKFold, cross\_val\_predict

# Load data

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"

names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']

dataset = read\_csv(url, names=names)

# Select only virginica and setose, then impose 0 and 1 as the class labels

a = dataset[(dataset['class'] == 'Iris-virginica') | (dataset['class'] == 'Iris-setosa')].copy()

a.replace(to\_replace = ['Iris-virginica', 'Iris-setosa'], value = [0,1], inplace=True) # gives 0 / 1

# Extract the numpy.ndarray which is in the pandas dataframe

a = a.values # extract the numpy.ndarray containing the data

# Separate features and class labels

X = a[:,0:4] # features: slicing with 0:4, the selected interval is [0,4) (np.ndarray)

Y = a[:,4] # targets (classes, labels) (np.ndarray)

# Define a MLP: LINEAR WORKS BEST IN THIS CASE!!!!!!!!

myMLP = MLPClassifier(hidden\_layer\_sizes=(), max\_iter=2000, activation = 'relu', solver='adam')

myMLP.verbose = 1

# Set k-fold cross validation

kfold = StratifiedKFold(n\_splits=5, random\_state=0, shuffle=True)

# Do your math!

Y\_probab = cross\_val\_predict(myMLP, X, Y, cv=kfold, method='predict\_proba')

# Calculate the points of the ROC curve

fpr2, tpr2, threshold = roc\_curve(Y, Y\_probab[:,1]) # Y are the original classes

# Calculate the area under the ROC curve

auc = roc\_auc\_score(Y, Y\_probab[:,1])

# Output

print("ROC-AUC of MLPClassifier: ", auc)

# Plot the ROC curve

pyplot.plot(fpr2, tpr2, color = "red")

pyplot.xlim([-0.01,1.01])

pyplot.ylim([-0.01,1.01])

pyplot.xlabel("FPR")

pyplot.ylabel("TPR")

pyplot.show()

Missing data and “data imputation”

Missing data[[38]](#footnote-38) arise in almost all serious statistical analyses. In datasets, missing values could be represented as ‘?’, ‘nan’, ’N/A’, blank cell, or sometimes ‘-999’, ’inf’, ‘-inf’.

Some approaches we can follow in case of missing data, are:

* **Complete Case Analysis**: ignore the missing samples and only consider observations where all variables are observed. This method should not be used unless the proportion of missing values is very small
* **Mean, median, mode imputation**: guess of a missing value is the mean, median, or mode (most frequently appeared value) of that variable.
* **Regression imputation**: mean, median or mode imputation only look at the distribution of the values of the variable with missing entries. If we know there is a correlation between the variables with the missing value, and other variables, we can often get better guesses by regressing the missing variable on other variables.
* **K-nearest neighbour (KNN) imputation**: Besides model-based imputation like regression imputation, neighbour-based imputation can also be used. K-nearest neighbour (KNN) imputation is an example of neighbour-based imputation. For a discrete variable, KNN imputer uses the most frequent value among the k nearest neighbours and, for a continuous variable, use the mean or mode.
* **Other imputation methods**.

Here is an example:

import numpy as np

from sklearn.impute import SimpleImputer

X\_train = [[12, np.nan, 34],

[10, 32, np.nan],

[np.nan, 11, 20]]

# Create our imputer to replace missing values with the mean e.g.

imp = SimpleImputer(missing\_values=np.nan, strategy='mean')

imp = imp.fit(X\_train)

# Impute our data, then train

X\_train\_imp = imp.transform(X\_train)

Check the result by showing the X\_train and X\_train\_imp variables.

1. Most of this tutorial comes from https://machinelearningmastery.com/machine-learning-in-python-step-by-step/ [↑](#footnote-ref-1)
2. https://scikit-learn.org/stable/ [↑](#footnote-ref-2)
3. https://machinelearningmastery.com/setup-python-environment-machine-learning-deep-learning-anaconda/ [↑](#footnote-ref-3)
4. https://towardsdatascience.com/virtual-environments-104c62d48c54, https://conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html [↑](#footnote-ref-4)
5. https://stackoverflow.com/questions/41535881/how-do-i-upgrade-to-python-3-6-with-conda, https://conda.io/projects/conda/en/latest/user-guide/tasks/manage-pkgs.html#package-update [↑](#footnote-ref-5)
6. https://www.scipy.org/install.html [↑](#footnote-ref-6)
7. https://stackoverflow.com/questions/1027714/how-to-execute-a-file-within-the-python-interpreter [↑](#footnote-ref-7)
8. [↑](#footnote-ref-8)
9. https://pandas.pydata.org/docs/reference/api/pandas.read\_csv.html [↑](#footnote-ref-9)
10. https://stackoverflow.com/questions/1228299/changing-one-character-in-a-string [↑](#footnote-ref-10)
11. https://realpython.com/python-lists-tuples/ [↑](#footnote-ref-11)
12. https://stackoverflow.com/questions/25351968/how-can-i-display-full-non-truncated-dataframe-information-in-html-when-conver [↑](#footnote-ref-12)
13. https://stackoverflow.com/questions/39870642/matplotlib-how-to-plot-a-high-resolution-graph [↑](#footnote-ref-13)
14. https://towardsdatascience.com/data-visualization-for-machine-learning-and-data-science-a45178970be7 [↑](#footnote-ref-14)
15. https://seaborn.pydata.org/index.html, https://towardsdatascience.com/data-visualization-for-machine-learning-and-data-science-a45178970be7, https://towardsdatascience.com/how-to-perform-exploratory-data-analysis-with-seaborn-97e3413e841d, https://seaborn.pydata.org/tutorial/distributions.html [↑](#footnote-ref-15)
16. https://www.geeksforgeeks.org/indentation-in-python/ [↑](#footnote-ref-16)
17. https://www.w3schools.com/python/python\_for\_loops.asp, https://pynative.com/python-range-function/, https://pynative.com/python-range-for-float-numbers/ [↑](#footnote-ref-17)
18. https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/ [↑](#footnote-ref-18)
19. https://stats.stackexchange.com/questions/266387/can-auc-roc-be-between-0-0-5 [↑](#footnote-ref-19)
20. https://www.dataquest.io/blog/settingwithcopywarning/, https://towardsdatascience.com/explaining-the-settingwithcopywarning-in-pandas-ebc19d799d25 [↑](#footnote-ref-20)
21. https://stackoverflow.com/questions/9746838/why-cant-attribute-names-be-python-keywords [↑](#footnote-ref-21)
22. https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html, https://stackoverflow.com/questions/46307490/how-can-i-extract-the-nth-row-of-a-pandas-data-frame-as-a-pandas-data-frame/46307819 [↑](#footnote-ref-22)
23. https://stackoverflow.com/questions/40901770/is-there-a-simple-way-to-change-a-column-of-yes-no-to-1-0-in-a-pandas-dataframe [↑](#footnote-ref-23)
24. https://scikit-learn.org/stable/modules/generated/sklearn.model\_selection.train\_test\_split.html [↑](#footnote-ref-24)
25. https://machinelearningmastery.com/index-slice-reshape-numpy-arrays-machine-learning-python/ [↑](#footnote-ref-25)
26. https://numpy.org/doc/stable/user/quickstart.html

    https://stackoverflow.com/questions/15879315/what-is-the-difference-between-ndarray-and-array-in-numpy

    https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html, https://numpy.org/doc/stable/reference/generated/numpy.array.html [↑](#footnote-ref-26)
27. https://analyticsindiamag.com/a-beginners-guide-to-scikit-learns-mlpclassifier/ [↑](#footnote-ref-27)
28. https://scikit-learn.org/stable/modules/generated/sklearn.neural\_network.MLPClassifier.html [↑](#footnote-ref-28)
29. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html [↑](#footnote-ref-29)
30. https://scikit-learn.org/stable/modules/preprocessing.html 6.3.1:“This class implements the Transformer API to compute the mean and standard deviation on a training set so as to be able to later re-apply the same transformation on the testing set.”. https://scikit-learn.org/stable/modules/cross\_validation.html#computing-cross-validated-metrics “Just as it is important to test a predictor on data held-out from training, preprocessing (such as standardization, feature selection, etc.) and similar data transformations similarly should be learnt from a training set and applied to held-out data for prediction”. [↑](#footnote-ref-30)
31. https://stackoverflow.com/questions/11218477/how-can-i-use-pickle-to-save-a-dict [↑](#footnote-ref-31)
32. Paragraph “6.3.1.1. Scaling features to a range” in https://scikit-learn.org/stable/modules/preprocessing.html [↑](#footnote-ref-32)
33. https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/ [↑](#footnote-ref-33)
34. https://machinelearningmastery.com/develop-first-xgboost-model-python-scikit-learn/ [↑](#footnote-ref-34)
35. https://scikit-learn.org/stable/modules/cross\_validation.html, https://stackoverflow.com/questions/45969390/difference-between-stratifiedkfold-and-stratifiedshufflesplit-in-sklearn, https://stackoverflow.com/questions/65318931/stratifiedkfold-vs-kfold-in-scikit-learn [↑](#footnote-ref-35)
36. https://scikit-learn.org/stable/modules/cross\_validation.html#computing-cross-validated-metrics, by cross\_val\_score and subsequent averaging of split scores [↑](#footnote-ref-36)
37. https://www.discoverbits.in/775/predict-probability-cross\_val\_predict-classification-problem and paragraph “3.1.1.2. Obtaining predictions by cross-validation” at https://scikit-learn.org/stable/modules/cross\_validation.html#computing-cross-validated-metrics [↑](#footnote-ref-37)
38. http://www.stat.columbia.edu/~gelman/arm/missing.pdf, https://www.kaggle.com/residentmario/simple-techniques-for-missing-data-imputation, https://medium.com/@Cambridge\_Spark/tutorial-introduction-to-missing-data-imputation-4912b51c34eb [↑](#footnote-ref-38)