# CS3920/CS5920 Lab Worksheet 3: In support of Assignment 1

Vladimir Vovk

October 10, 2021

This lab worksheet should be completed during the lab session of 11 October and your independent study time.

The topics that are briefly covered in this worksheet are:

- Reading `NumPy` arrays using `genfromtxt`.
- Functions in Python.
- A pitfall of assignments in Python.
- Finding the rank of a number in a list, using vector operations and a loop.
- Timing in Python.
- Computing the minimum of an array.
- Using `print()` for debugging.

For further details of the functions used in this worksheet, see [1, Chapter 1] and [3]. For details of the function `genfromtxt` see `https://docs.scipy.org/doc/numpy/user/basics.io.html`.

## 1 Reading `NumPy` arrays

Read about the `ionosphere` data set in the assignment sheet. Have a look at the file `ionosphere.txt` at the course's Moodle page. `genfromtxt` is a very versatile command for reading text files, including fairly complicated ones such as `ionosphere.txt`.

```
In[1]:
  import numpy as np
  X = np.genfromtxt("ionosphere.txt", delimiter=",")
  X[:3]
Out[1]:
  [the first 3 rows]
```

We had to say explicitly that the delimiter in this file is the comma (while in Lab 1 we used the default delimiter, a white space). Whereas the features look fine, the labels are floating point numbers rather than integers.

Let us load the features (`X`) and the labels (`y`) separately. From `Out[1]`, we can see that the number of features is 34. Therefore, the following code extracts only the samples.

```
In[2]:
  X = np.genfromtxt("ionosphere.txt", delimiter=",",
    usecols=np.arange(34))
  X[:3]
Out[2]:
  [the first 3 rows]
```

The option `usecols` chooses the columns of the data array that we wish to extract. As always in Python, the columns are numbered starting from 0. Finally, we can extract the labels:

```
In[3]:
  y = np.genfromtxt("ionosphere.txt", delimiter=",",
    usecols=34, dtype='int')
  y[:3]
Out[3]:
  [the first 3 labels]
```

Here we say explicitly that we want the labels to be of the data type `int` (i.e., we want them to be integer numbers.

## 2 Functions in Python

Sometimes we need to perform similar operations many times (such as computing distances when implementing Nearest Neighbours), in which case user-defined *functions* are convenient. (Python has lots of predefined functions as well, of course.) This is an example of a function:

```
In[4]:
  def cube(x):
    """this is my first function in this course; it cubes a number"""
    return x**3
  cube(3)
Out[4]:
  27
```

Make sure to pay attention to the indentation; the first three lines after `In[4]` are the definition of the function `cube` (including its description in the second line), and the fourth line after `In[4]` (i.e., the line immediately before `Out[4]`) is the call to the function (returning `27`). The user of your function can consult its description:

```
In[5]:
```

```
  help(cube)

  Help on function cube in module __main__:

  cube(x)
    this is my first function in this course; it cubes a number
```

It is possible to write a function that returns more than one value:

```
In[6]:
  def powers(x):
    """this function computes both square and cube"""
    return x**2, x**3
  powers(3)
Out[6]:
  (9, 27)
```

If you need only one of the values, use indexing:

```
In[7]:
  powers(3)[1]
Out[7]:
  27
```

# 3 Assignment in Python

There is a scary feature of Python that you will sooner or later come across. Let's perform a few trivial assignments.

```
In[8]:
  a = np.array([1,2,3])
  a
Out[8]:
  array([1, 2, 3])
In[9]:
  b = a
  b
Out[9]
  array([1, 2, 3])
In[10]:
  a[1] = 100
  a
Out[10]:
  array([  1, 100,   3])
```

So far, everything is as expected.

```
In[11]:
  b
Out[11]:
  array([  1, 100,   3])
```

But we did not change `b`!

The assignment `b = a` does not produce a copy of `a`, it just gives `a` an additional name, `b`. Both `a` and `b` refer to the same array.

If you really want a copy of `a`, use, e.g.,

```
In[12]:
  b = np.array(a)
```

or the specialized copying method

```
In[13]:
  b = a.copy()
```

# 4 Rank of a number

Suppose we have a sequence of real numbers and would like to find the rank of its last element (this is a common task in conformal prediction). First we discuss further methods of creating `NumPy` arrays.

```
In[14]:
  a = np.array([1,2,3])
  b = np.array([5,6])
  np.concatenate((a,b))

  array([1, 2, 3, 5, 6])
```

Now let us create a much bigger sequence:

```
In[15]:
  my_array = np.concatenate((np.arange(10**6),np.array([4])))
```

**Exercise 1.** *Describe briefly the array* `my_array` *in English.*

In general, the operation of concatenation is very powerful; in particular, we can concatenate not just vectors but also matrices. (Have a quick look at the description of the function `concatenate` in [2].)

**Exercise 2.** *Why did we repeat opening and closing parentheses in* `In[14]` *and* `In[15]`*? (We have* `((...))` *there.)*

Now let us try a "vectorized" way of computing the rank, i.e., a way not using loops (such as `for` loops).

```
In[16]:
  import time
  start = time.time()
  print(sum(4 >= my_array))
  print(time.time() - start,"seconds")
```

```
  6
  3.672 seconds
```

Here we are using the `time` module, which contains the very useful function `time` returning the current time in seconds. The computation takes time 3.672, in seconds.

**Exercise 3.** *Why do you think the result in* `In[16]` *is 6?*

Next let's compute the rank using a loop.

```
In[17]:
  start = time.time()
  count = 0
  for n in range(my_array.size):
    if 4 >= my_array[n]:
      count = count + 1
  print(count)
  print(time.time() - start,"seconds")

  6
  0.579 seconds
```

We get the same result, 6, but interestingly, we get it faster: in less than a second.

# 5 Computing minimum

The Nearest Neighbour method depends on the ability to compute the minimum of an array of real numbers, and in evaluating conformal predictors you may need to compute the largest p-value. This is a standard way of computing the minimum of an array in a loop:

```
In[18]:
  start = time.time()
  current_minimum = math.inf
  for n in range(my_array.size):
    if current_minimum > my_array[n]:
      current_minimum = my_array[n]
  print(current_minimum)
  print(time.time() - start,"seconds")

  0
  0.339 seconds
```

The variable `current_minimum` keeps the smallest element of the array found so far. We initialize it to $\infty$ and then move along the array keeping track of the current record. In the end it will hold the overall minimum.

**Exercise 4.** *Modify the code to also find the position of the minimum (i.e., the index of the smallest element of the array; you may assume that there is a unique smallest element).*

The `NumPy` library also provides functions for computing the maximum of an array and its position: read about `max` and `argmax` in [2]. The corresponding functions for minimum are `min` and `argmin`.

# 6 Debugging

Your code may sometimes produce unexpected results. An invaluable way of debugging it is to inspect key variables after the execution of your code. If interesting values were overwritten by other variables, add additional `print` statements (for example, in a loop) and see if you can explain the results that you see.

# 7 Other exercises

5. Write a function for computing the squared Euclidean norm of a vector represented as a `NumPy` array. You may use `for` loops. Your function should take two inputs, the array and its length.

6. Modify your function in such a way that it takes only one input, the array. If you do not remember a `NumPy` function computing the length of an array, Google for it.

7. Modify your function in such a way that it computes the Euclidean norm of a vector. If needed, Google for the function computing the square root.

If you have completed all exercises before the end of the lab session, you may leave early. But remember that this lab session is a valuable opportunity to ask questions related (but not too directly) to Assignment 1. Therefore, you might prefer to work on Assignment 1, in case you come across any problems.

# References


[1] Andreas C. Müller and Sarah Guido. *An introduction to machine learning with Python.* O'Reilly, Beijing, 2017.

[2] `NumPy` manual. `https://docs.scipy.org/doc/numpy/`, 2008–2021.

[3] `scikit-learn` tutorials. `http://scikit-learn.org/stable/tutorial/`, 2007–2021.