CS3920/CS5920 Lab Worksheet 1: Introduction to Python and Jupyter Notebooks

Volodymyr Vovk

October 1, 2022

This lab worksheet is particularly important in that all future lab sessions depend on it. It should be completed during the lab session of 3 October and your independent study time. The result of your work should be a Jupyter notebook; it should contain all code given in the main part of the worksheet, the code you write for all exercises, and all necessary explanations. There is no need to have your Jupyter notebook checked, but its material may be used in Quiz 1.

The topics that are briefly covered in this worksheet are:

- Installing and starting the key elements of the Python ecosystem, including Jupyter Notebooks.
- Basic commands.
- Experiments with the `iris` dataset.

For further details of this worksheet, see [1, Chapter 1] and [2].

If you are getting an error message you do not understand, it is often useful to type it (perhaps surrounded by quotation marks) in Google, and you are likely to find an accessible explanation (for example, in `stackoverflow.com`).

1 Starting a Jupyter Notebook

You can download Anaconda to your laptop from

https://www.anaconda.com/

(scroll down and choose "Individual Edition"), and then you can start it by choosing the subitem `Jupyter Notebook` of the item `Anaconda...` in the start menu.

Alternatively, you can run Jupyter Notebook on the departmental system as follows:

- Use `NoMachine` to log in to `teaching` (or `teaching.cim.rhul.ac.uk`; you can search for `NoMachine` using the Start button).

- Start Jupyter Notebook by typing `jupyter notebook` at the command prompt. (To get a command prompt you may need to launch the Terminal Emulator.)

Once you are in Jupyter Notebook, create a new folder (by clicking on New close to the top right corner and then on Folder). Rename the folder that you have created to something less bland than `Untitled`, such as `CS5920_Labs`. Enter your new folder. Create a new Jupyter notebook (by clicking on New close to the top right corner and then on Python 3). Rename it to, e.g., `Lab1`.

Now you can see a box labelled `In[1]`, and the system is ready for your instructions.

2 NumPy and matplotlib

Using `NumPy`:

```
In[1]:
  import numpy as np
  x = np.array([[1, 2, 3], [4, 5, 6]])
  x
Out[1]:
  array([[1, 2, 3],
         [4, 5, 6]])
```

You should type your commands into the `In[...]:` box and press "Run" in the tool bar just below the menu. The system will execute them and, if needed, provide an answer in the corresponding `Out[...]:` box.

Try `matplotlib`:

```
In[2]:
  %matplotlib inline
  import matplotlib.pyplot as plt
  # Generate a sequence of integers
  x = np.arange(20)
  # create a second array using sinus
  y = np.sin(x)
  # The plot function makes a line chart of one array against another
  plt.plot(x,y,marker="x")
```

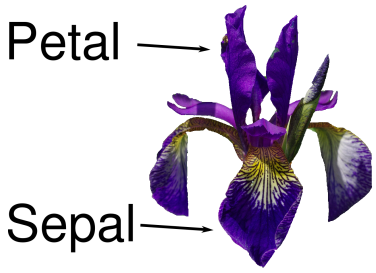
This plots `y` versus `x` (where `y` is the results of applying sin to each element of `x`).

3 Experiments with iris

In this part we will go through a simple machine learning application and create our first model.

A hobby botanist would like to tell the species of iris flowers that she found. She has a training set of labelled flowers. The features are the length and width

of the petals, and the length and width of the sepal, all measured in centimeters. This is what a typical iris may look like:



There are three possible labels (species): Setosa, Versicolor, or Virginica.

The `iris` dataset is a classical dataset in machine learning and statistics, collected by Ronald A. Fisher. It is included in `scikit-learn` in the dataset module. Type

```
In[3]:
  from sklearn.datasets import load_iris
  iris = load_iris()
```

You can't see anything but the system has loaded `iris` and now you can use it.

A nice feature of Jupyter notebooks is that you can mix code and text. You can switch "Code" in the tool bar to "Markdown" and add some comment (such as "iris is ready to use").

The `iris` object that is returned by `load_iris` is a `Bunch` object, which is very similar to a dictionary in Python. (Please read about dictionaries in Python by clicking here, or going to

`https://docs.python.org/3/tutorial/datastructures.html#dictionaries`

Have a brief look now and read more at home. But you should never need creating a `Bunch` object yourself!) It contains keys and their corresponding values:

```
In[4]:
  iris.keys()
Out[4]:
  dict_keys(['target_names','feature_names','DESCR','data','target'])
```

The value of the key `DESCR` is a description of the dataset.

```
In[5]:
  print(iris['DESCR'])
Out[5]:
  Iris Plants Database
  ====================
  Notes
  ----
```

```
  Data Set Characteristics:
  :Number of Instances: 150 (50 in each of three classes)
  :Number of Attributes: 4 numeric, predictive att
  ...
```

Have a quick look at the description (which is fairly long).

The value of the key `target_names` is an array of strings containing the labels (species of flower that we want to predict):

```
In[6]:
  iris['target_names']
Out[6]:
  array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

In many cases using an explicit `print` command leads to a nicer output containing less irrelevant technical information:

```
In[7]:
  print(iris['target_names'])
Out[7]:
  ['setosa' 'versicolor' 'virginica']
```

`print` also has plenty of optional arguments making the output even nicer; see [1] for numerous examples (at home). The value of `feature_names` is a list of strings giving the description of each feature:

```
In[8]:
  iris['feature_names']
Out[8]:
  ['sepal length (cm)',
   'sepal width (cm)',
   'petal length (cm)',
   'petal width (cm)']
```

The data itself is contained in the `target` and `data` fields, in a `NumPy` array:

```
In[9]:
  type(iris['data'])
Out[9]:
  numpy.ndarray
```

The rows in the data array correspond to samples (flowers), while the columns are their features:

```
In[10]:
  iris['data'].shape
Out[10]:
  (150, 4)
```

There are 150 samples and 4 features. Here are the first three samples:

```
In[11]:
```

```
  iris['data'][:3]
Out[11]:
  array([[5.1, 3.5, 1.4, 0.2],
         [4.9, 3. , 1.4, 0.2],
         [4.7, 3.2, 1.3, 0.2]])
```

We can see that all of the first three flowers have a petal width of 0.2 cm and that the first flower has the longest sepal, at 5.1 cm. The target array contains the species of each of the flowers that were measured, also as a `NumPy` array:

```
In[12]:
  type(iris['target'])
Out[12]:
  numpy.ndarray
```

It is a one-dimensional array, with one entry per flower:

```
In[13]:
  iris['target'].shape
Out[13]:
  (150,)
```

The species are encoded as integers from 0 to 2:

```
In[14]:
  iris['target']
Out[14]:
  array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2,
         2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
         2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
         2, 2, 2])
```

In this array, 0 means setosa, 1 means versicolor, and 2 means virginica.

4 Look at your data!

It is often a good idea to visualize your data, to see if the task is easily solvable without machine learning, or if the desired information might not be contained in the data. Computer screens have only two dimensions, which allows us to only plot two (or maybe three) features at a time.

In `scikit-learn`, the matrix of samples is usually denoted with a capital `X`, while the vector of labels is denoted by a lower-case `y`.

```
In[15]:
  X = iris['data']
  y = iris['target']
```

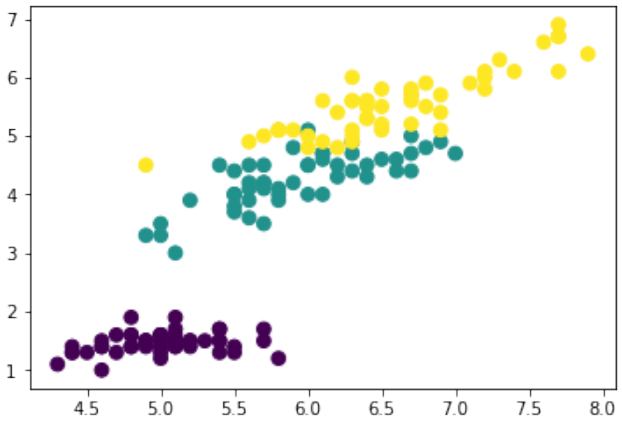
A possible guess is that features 0 and 2 are almost as informative as all four features (feature 0 serving as a proxy for the size of the sepal and feature 2 as a proxy for the size of the petal).

```
In[16]:
  plt.scatter(X[:,0], X[:,2], c=y, s=60)
Out[16]:
  [the scatter plot]
```

The command `scatter` produces the following scatter plot of the points

$$(\texttt{sepal length}, \texttt{petal length})$$

for all flowers in the dataset:



The colour of each point is its label: in `c=y`, `c` stands for "colour" and `y` is the vector of labels. The parameter `s` is the size of the marker. The classification task (especially for the bottom left species) is not hopeless.

We can also plot several scatter plots at once:

```
In[17]:
  fig, ax = plt.subplots(3, 3, figsize=(15, 15))
  plt.suptitle("iris_pairplot")

  for i in range(3):
    for j in range(3):
      ax[i,j].scatter(X[:,j], X[:,i+1], c=y, s=60)
      ax[i,j].set_xticks(())
      ax[i,j].set_yticks(())
      if i == 2:
        ax[i,j].set_xlabel(iris['feature_names'][j])
      if j == 0:
        ax[i,j].set_ylabel(iris['feature_names'][i+1])
      if j > i:
        ax[i,j].set_visible(False)
```

```
[the picture]
[your explanation (see the next section)]
```

5 Exercises

The following exercises involve the `iris` dataset.

1. Is the guess that features 0 and 2 are almost as informative as all four features correct? Or, at least, can we say that features 0 and 2 are as informative as any other pair of features? Draw scatter plots for other pairs of features, and write your conclusions in your Jupyter notebook.

2. This is a difficult exercise. What is going on in the third listing of Section 4 (the one with two nested `for` loops)? Add a brief explanation to the Jupyter notebook for this lab. If needed, use the `help` command (such as `help(plt.subplots)`).

3. If you have done the previous exercise, this exercise is optional for you. Try all code discussed in the lectures.

If you have completed all exercises before the end of the lab session, you may leave early.

References

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

[2] `scikit-learn` tutorials. `http://scikit-learn.org/stable/tutorial/`, 2007–2022.