Lab 1: Introduction to scikit-learn (Part 1)

Foundations of Machine Learning

2015

The goal of this lab is to introduce **scikit-learn**, a Python scientific toolbox for machine learning. scikit-learn is widely used in the machine learning community and is based on SciPy, a well-established Python ecosystem for mathematics, science, and engineering. scikit-learn is an open source project.

This tutorial is based on resources from http://scikit-learn.org and the scikit-learn tutorial at Scipy 2013 by Gaël Varoquaux, Jake VanderPlas and Olivier Grisel (https://www.youtube.com/watch?v=r4bRUvvlaBw and https://github.com/jakevdp/sklearn_scipy2013).

In this first part, we'll set up our scientific Python working environment and get familiar with data representation in scikit-learn.

1 Setup

This tutorial uses Python 2.6 or 2.7, and requires recent versions of:

- **NumPy** (version 1.5+), the fundamental package for numerical computing in Python (http://www.numpy.org/);
- **SciPy** (version 0.10+) (http://www.scipy.org/), a Python ecosystem for mathematics, science and engineering;
- **matplotlib** (version 1.1+), a Python 2D plotting library (http://matplotlib.org/);
- **IPython** (version 0.13.1+), for interactive computing (http://ipython.org/);
- **scikit-learn** (version 0.16).

The notebook functionality of IPython allows you to create, manipulate and save notebooks (keeping track of the commands you typed and the results you obtained, together with your comments and observations) within your web browser and can be a very good way to produce technical reports on your work.

The easiest way to install these requirements is to use a packaged distribution, such as Anaconda CE, a free package provided by Continuum Analytics: http://continuum.io/downloads.html.

Task 1. Set up your scikit-learn working environment, for example by installing Anaconda. You should now be able to open an IPython session and import numpy, scipy, matplotlib and sklearn without raising any error message.

```
cazencott@falstaff ~ $ ipython
Python 2.7.6 (default, Jun 22 2015, 17:58:13)
Type "copyright", "credits" or "license" for more information.

IPython 1.2.1 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: import numpy

In [2]: import scipy

In [3]: import matplotlib

In [4]: import sklearn

In [5]:
```

2 IPython

IPython is a powerful interactive Python interpreter. In particular, it offers the following functionalities:

- **Tab-completion.** Autocomplete the name of an object after you've started typing it. To explore the structure of an object, and in particular to get a list of its attributes, simply type `object_name.<TAB>`.
- **Inline help.** Within IPython, you can get help about a particular module or function by typing `help(module_name)`. You can also get detailed information about a particular object by typing `object_name?`.
- **IPython magic functions.** Magics are predefined functions, which are called by prepending their name with "%". For example,
  - `%history` prints out all commands typed into the session;
  - `%paste` pastes and run what is copied to your clipboard;
  - `%run script.py` allows you to run the script `script.py` within IPython;
  - `%save` saves a set of lines to a file.

  You can find a list of magics here: http://ipython.org/ipython-doc/stable/interactive/magics.html.
- **The IPython Notebook.** An interactive computational environment in which you can combine code execution, rich text, math formulas, plots and more. You use it through your web browser and generate IPython notebooks files in the `.ipynb` format. They can readily be shared between IPython notebook users, or converted

to a variety of formats (including HTML).
To set up an IPython notebook, start here: http://ipython.org/ipython-doc/stable/notebook/notebook.html. You will find more information about the IPython Notebook here: http://ipython.org/notebook.html.

For more tips on using IPython, you can refer to

– Introducing IPython: http://ipython.org/ipython-doc/stable/interactive/tutorial.html
– IPython Tips & Tricks: http://ipython.org/ipython-doc/stable/interactive/tips.html#tips

Task 2. Play around with IPython, trying out the above commands. Set up an IPython notebook for this tutorial.

3 NumPy arrays

NumPy arrays are a fundamental structure for scientific computing. NumPy arrays are homogeneous (i.e. only all objects it contains have the same type) multi-dimensional arrays, which we'll use among other things to represent vectors and matrices.

Task 3. Try out the following NumPy commands.

```
# import numpy
import numpy as np

# create a random array
X = np.random.random((3, 5))

# The dimensions of an array are accessible via:
X.shape

# The total number of elements of an array is accesible via:
X.size

# You can of course print the array
print X
```

Accessing elements, rows and columns of arrays is easy. Remember, in Python indexing starts at 0:

```
# get a single element
print X[0, 0]

# get a row
print X[1, :]
```

```
print X[1]

# get a column
print X[:, 1]
```

Transposing an array:

```
X.T
```

Transforming a row vector into a column vector:

```
# get a row of X
Y = X[1]
print Y
print Y.shape

# transpose it into a column
Ycol = Y[:, np.newaxis]
print Ycol
print Ycol.shape
```

You can apply a same transformation to all items in the array:

```
# multiply all items in X by 2
2*X

# add 1 to all items in X
X+1

# apply some classical mathematical functions
exp(X)
```

Matrices products:

```
# elementwise product
X*X

# dot product
X.dot(X.T)
```

For more on arrays, you can refer to http://docs.scipy.org/doc/numpy/reference/arrays.html. For more about NumPy, you can refer to:

- The Tentative NumPy Tutorial at http://wiki.scipy.org/Tentative_NumPy_Tutorial;
- The NumPy documentation at http://docs.scipy.org/doc/numpy/index.html.

Sparse matrices It often happens that the data we manipulate contains a lot of zeros. In this case, storing all the zeros is inefficient, and it is much more efficient to use data structures meant for sparse matrices. This can be done with the `scipy.sparse` submodule, which allows to store sparse matrices efficiently, and implements many interesting functions (linear algebra, sparse solvers, graph algorithms, etc.).

4 matplotlib

Visualization is an important part of machine learning. Plotting your data will allow you to have a better feel for it (how are the features distributed, are there outliers, etc.). Plotting measures of performance (whether ROC curves or single-valued performance measures, *with error bars*) allows you to rapidly compare methods.

matplotlib is a very flexible data visualization package, partially inspired by MATLAB. If you are interested in data visualization, know that more and more data scientists move to using Seaborn (http://stanford.edu/~mwaskom/software/seaborn/) or Bokeh (http://bokeh.pydata.org/en/latest/).

If you are using an IPython notebook, turn on the "inline" mode of matplotlib so that figures are displayed inside your notebook, by typing `%pylab inline`. In IPython (without a notebook), use `%matplotlib` to ensure your plots are displayed.

Task 4. Go over the following basics of matplotlib.

Set up matplotlib:

```
# import matplotlib
%matplotlib
import matplotlib.pyplot as plt

# alternatively, in an IPython notebook:
%matplotlib inline
import matplotlib.pyplot as plt
```

Plotting a line:

```
# create an array of 100 equally-spaced points between 0 and 10
x = np.linspace(0, 10, 100)

# plot a sinusoid
plt.plot(x, np.sin(x))
```

Scatterplots:

```
# create 500 points with random (x, y) coordinates
x = np.random.normal(size=500)
y = np.random.normal(size=500)

# plot them
plt.scatter(x, y)
```

Showing images saved as arrays:

```
# create an image
x = np.linspace(1, 12, 100)
y = x[:, np.newaxis]
im = y * np.sin(x) * np.cos(y)
```

```
print im.shape

# show it (the origin is, by default, at the top-left corner!)
plt.imshow(im)

# Contour plot - note that origin here is at the bottom-left by default!
plt.contour(im)

# 3D plotting
from mpl_toolkits.mplot3d import Axes3D
ax = plt.axes(projection='3d')
# y.ravel() is the 1D array version of y
print y.shape
print y.ravel().shape
# create a grid from x, y
xgrid, ygrid = np.meshgrid(x, y.ravel())
# cmap: specify the color map (color scheme, if you wish)
# cstride, rstride: set the stride used to sample
# the input data to generate the graph.
ax.plot_surface(xgrid, ygrid, im, cmap=plt.cm.jet, cstride=2,
                rstride=2, linewidth=0)
plt.show()
```

Many more types of plots and functionalities to label axes, display legends, etc. are available. The matplotlib gallery (http://matplotlib.org/gallery.html) is a good place to start to get an idea of what is possible and how to do it.

5 Representation of data in scikit-learn

Machine learning is about creating models from data: for that reason, we'll start by discussing how data can be represented in order to be understood by the computer. Along with this, we'll build on our matplotlib examples from the previous section and show some examples of how to visualize data.

Most machine learning algorithms implemented in scikit-learn expect data to be stored in a two-dimensional array, of shape `(n_samples,n_features)`. The arrays can be either NumPy arrays, or in some cases scipy.sparse matrices. The number of features must be fixed in advance. However it can be very high dimensional (e.g. millions of features) with most of them being zeros for a given sample. This is a case where scipy.sparse matrices can be useful, in that they are much more memory-efficient than NumPy arrays.

5.1 Iris data

As an example of a simple dataset, we're going to take a look at the Iris data stored by scikit-learn. The data is a classical UCI dataset and consists of measurements of three different species of irises.

Iris setosa

Iris versicolor

Iris virginica

Question. If we want to design an algorithm to recognize iris species, what might the data be?

scikit-learn has a very straightforward set of data on these iris species. The data consist of the following:

- Features in the Iris dataset:
  - sepal length in cm
  - sepal width in cm
  - petal length in cm
  - petal width in cm
- Target classes to predict:
  - *Iris setosa*
  - *Iris versicolour*
  - *Iris virginica*

Task 5. Load the Iris data with scikit-learn as follows.

scikit-learn embeds a copy of the iris CSV file along with a helper function to load it into NumPy arrays:

```
# Load the iris data
from sklearn.datasets import load_iris
iris = load_iris()

# iris is a Bunch object
# see what's available in iris:
iris.keys()

# The features of each sample flower are stored in the data
#  attribute of the dataset:
n_samples, n_features = iris.data.shape
print n_samples
print n_features
print iris.data[0]
```

```
# The information about the class of each sample is stored in the target
# attribute of the dataset:
print iris.target.shape
print iris.target

# The names of the classes are stored in the last attribute,
# namely target_names:
print iris.target_names
```

This data is four dimensional, but we can visualize two of the dimensions at a time using a simple scatter-plot.

```
%matplotlib
x_index = 0
y_index = 1

# this formatter will label the colorbar with the correct target names.
# lambda i, *args: iris.target_names[int(i)] converts an index i into the
# corresponding label
formatter = plt.FuncFormatter(lambda i, *args: iris.target_names[int(i)])

plt.scatter(iris.data[:, x_index], iris.data[:, y_index], s=40,
            c=iris.target)
plt.colorbar(ticks=[0, 1, 2], format=formatter)
plt.show()
plt.xlabel(iris.feature_names[x_index])
plt.ylabel(iris.feature_names[y_index])
```

Task 6. In the above script, change x_index and y_index, and find (visually) a combination of two parameters which maximally separate the three classes.

5.2 Digits data

Let us now take a look at the Digits dataset. This data set, also very classical, contains images of hand-written digits, each labeled with the corresponding digit. This means there are 10 classes in this data. Each image has been compressed into an 8×8 matrix.

Task 7. Load the Digits data with scikit-learn as follows.

```
# Load the digits dataset
from sklearn.datasets import load_digits
digits = load_digits()

# Explore the data format
digits.keys()
```

```
print digits.data[0]
print digits.target

# Each data point is an image,
# and has for target the number it represents
print np.unique(digits.target)

# Data size
n_samples, n_features = digits.data.shape
print (n_samples, n_features)

# Image format
print digits.images.shape
```

What is the difference between `digits.data` and `digits.images`? Actually, `digits.data` is simply `digits.images` reshaped, with the 8×8 array flattened out in a 64-dimensional vector. These two representations can exist without needing to fully duplicate the data, and hence the memory overhead is very small. Indeed, the two arrays, although they have different shapes, point at the same memory block. To check this is true:

```
# check the two arrays contain identical data
print np.all(digits.images.reshape((1797, 64)) == digits.data)

# check the two arrays point to the same memory address
print digits.data.__array_interface__['data']
print digits.images.__array_interface__['data']
```

Let us visualize the data:

```
# set up the figure
fig = plt.figure(figsize=(6, 6)) # figure size in inches
fig.subplots_adjust(left=0, right=1, bottom=0, top=1, hspace=0.05,
                    wspace=0.05)

# plot the 49 first digits: each image is 8 by 8 pixels
for i in range(49):
  ax = fig.add_subplot(7, 7, i + 1, xticks=[], yticks=[])
  # ax.imshow(digits.images[i], cmap=plt.cm.Greys, interpolation='nearest')
  # ax.imshow(digits.images[i])
  ax.imshow(digits.images[i], cmap=plt.cm.Greys)

  # label the image with the target value
  ax.text(0, 6, str(digits.target[i]))
plt.show()
```

Task 6. Modify the above script to visualize all the "8" contained in the data. Use the `cmap` option of `imshow` to display the data with a colormap that is more appropriate for visualizing intensities.

Help on colormaps: http://matplotlib.org/users/colormaps.html.