3. Deep learning tools

3.4 SciPy

Manel Martínez-Ramón
Meenu Ajith
Aswathy Rajendra Kurup

Introduction

- Built on top of NumPy.
- Optimized to process multidimensional arrays.
- A tool to work with
  - images and signal processing
  - large data sets.
- This section contains some examples of the capabilities of the library.

Data input and output

- SciPi is used to work with several data types as .mat, .wav and others, with specific commands for each data type.
- In this example we create, save and load a .mat file, that can be also read in MatLab.

```python
import scipy.io as sio
import numpy as np

arr = np.array([1,2,3,4])  #create an array
#save the array by the name 'arr_samp'
sio.savemat('sample_data.mat', {'arr_samp': arr})
#load the array into the variable from mat file
sample_arr = sio.loadmat('sample_data.mat')

print(sample_arr['arr_samp'])
```

```
[[1 2 3 4]]
```

Clustering methods

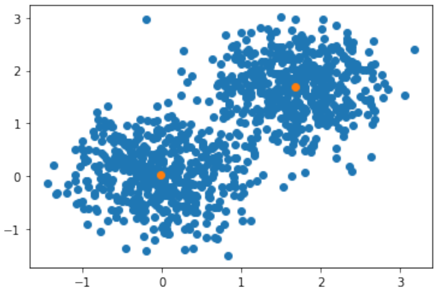
- Clustering is a procedure that is intended to split a set of data in different points with the purpose of obtaining an interpretation of the data structure.
- A popular rather limited clustering method is K-means, and SciPy has it implemented

```python
from scipy.cluster.vq import kmeans, vq, whiten #importing the packages required for Kmeans clustering
import numpy as np
import matplotlib.pyplot as plt
data1 = np.random.randn(500,2) # Random 2D data
data2 = np.random.randn(500,2) + np.array([3,3])
data = np.vstack((data1,data2)) # stacking the two data
whit = whiten(data) # whiten the data
[center,_] = kmeans(whit, 2) # Apply Kmeans clustering
```

Clustering methods

- This is the representation

```
out = vq(data, center) #assigns labels to data
plt.scatter(whit[:,0], whit[:,1]) #plot the scatterplots
plt.scatter(center[:,0], center[:,1]) # plot the centroids
plt.show()
```



Constants

- *scipy.constants* gives access to many constants.
- They can be used in mathematical expressions.

```python
import scipy.constants
from scipy.constants import find

print(find('light')) # find all constants with keyword 'light'
print(scipy.constants.physical_constants['speed of light in
    vacuum'])
print(scipy.constants.pi)
```

```
['speed of light in vacuum']
(299792458.0, 'm s^-1', 0.0)
3.141592653589793
```

Linear algebra

- *scipy.linalg* performs linear algebra operations in Python. *F*aster than BLAS and LAPACK libraries.

```python
from scipy import linalg
import numpy as np

A = np.array([[1,2],[3,2]]) # create a square matrix

print(linalg.det(A))     # compute the determinant of a
print(linalg.inv(A))        # compute the inverse of the matrix
```

```
-4.0
[[-0.5   0.5 ]
 [ 0.75 -0.25]]
```

Linear algebra

```
val, vect = linalg.eig(A)  # compute the svd of a
print('\neigenvalues =')
print(val)
print('\neigenvectors =')
print(vect)
b = np.array([2,4])        #  2D vector
print(linalg.solve(a,b))   # Solution of equation Ax=b
```

```
eigenvalues =
[-1.+0.j  4.+0.j]

eigenvectors =
[[-0.70710678 -0.5547002 ]
 [ 0.70710678 -0.83205029]]

solution of equation Ax=b:
[1.  0.5]
```

Numeric integrals

- The *integrate* package is used to compute numeric integrals with various methods.
- Example:

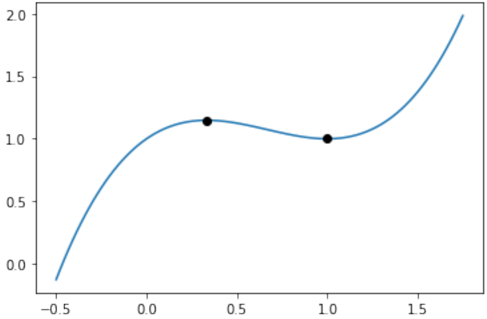
$$\int_{y=0}^{\frac{1}{2}} \int_{x=0}^{1-2y} xy dx dy = \frac{1}{96} \approx 0.0104167$$

```python
from scipy import integrate
def f(x, y):
    return x*y
def bounds_y():
    return [0, 0.5]
def bounds_x(y):
    return [0, 1-2*y]

integrate.nquad(f, [bounds_x, bounds_y])
```

```
(0.010416666666666668, 4.101620128472366e-16)
```

Optimization

- Minimizing or maximizing a function with constrained and unconstrained minimization problems.
- The package has the most common optimization approaches such as least squares (*least_ squares()*) and curve fitting techniques (*curve_fit()*).



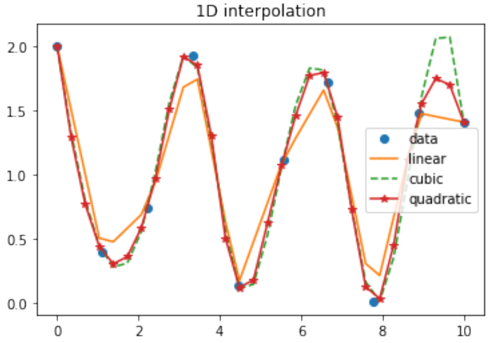
$$f(x) = x^3 - 2x^2 + x$$
$$f'(x) = 3x^2 - 4x + 1$$
$$x_0 = \frac{1}{3},\ 1$$

```python
from scipy import optimize
import numpy as np

def func(x):
        return  x**3 -  2*x**2 + x  + 1  # The function

optimize.minimize_scalar(func) # Find the minimum
```

```
     fun: 1.0
    nfev: 12
     nit: 8
 success: True
       x: 1.0
```

Other stuff

- The package *interpolate* is also useful. It can interpolate using several methods.



- The package *scipy.ndimage* can does image processing operations:display, geometric transformations, filtering, edge detection...
- Library *MISC* is used as a sample to test operations.