Section 9: Introduction to NumPy and SciPy

Linxing Preston Jiang

Allen School of Computer Science & Engineering, University of Washington

May 24, 2018
Motivation

We have learned all basic data structures...do we need more?
A question

You have a matrix like this:

\[
\begin{bmatrix}
1 & 2 & 3 & 4 \\
4 & 5 & 6 & 7 \\
7 & 8 & 9 & 10
\end{bmatrix}
\]

and you want to sum up numbers by each column. How do you write code for it?
In Python

A solution using native list

```python
sums = []
for col_idx in range(len(matrix[0])):
    sum = 0
    for row_idx in range(len(matrix)):
        sum += matrix[row_idx][col_idx]
    sums.append(sum)
print sums
```
In Python

a solution if using numpy arrays

```
print matrix.sum(axis=1)
```
Another comparison

Sum benchmark: summing over a list

define N as 10000000

define numpy_array as arange(N)
define python_list as range(N)

print "### python list ###"
define start as time.time()
define sum as 0
for i in python_list:
  sum += i
print "average is: ", float(sum) / N
print "used time: ", time.time() - start

print "### numpy array ###"
define start as time.time()
print "average is: ", numpy_array.mean()
print "used time: ", time.time() - start
First, import

```python
import numpy

OR

import numpy as np (assuming this from now on)
```
Most important module of NumPy: Arrays

- just like lists, but could only contain **same type** of objects!
Most important module of NumPy: Arrays

- just like lists, but could only contain **same type** of objects!
- creation: `a = np.array([1, 4, 5, 6], float)`
Most important module of NumPy: Arrays

- just like lists, but could only contain same type of objects!
- creation: `a = np.array([1, 4, 5, 6], float)`
- You can use the same indexing:
Most important module of NumPy: Arrays

- just like lists, but could only contain **same type** of objects!
- creation: `a = np.array([1, 4, 5, 6], float)`
- You can use the same indexing:
  - `a[:2]`
Most important module of NumPy: Arrays

- just like lists, but could only contain **same type** of objects!
- creation: `a = np.array([1, 4, 5, 6], float)`
- You can use the same indexing:
  - `a[:2]`
  - `a[1]`
Most important module of NumPy: Arrays

- just like lists, but could only contain **same type** of objects!
- creation: `a = np.array([1, 4, 5, 6], float)`
- You can use the same indexing:
  - `a[:2]`
  - `a[1]`
  - `a[1:]`
Most important module of NumPy: Arrays

- just like lists, but could only contain **same type** of objects!
- creation: `a = np.array([1, 4, 5, 6], float)`
- You can use the same indexing:
  - `a[:2]`
  - `a[1]`
  - `a[1:]`
- arrays can easily be multidimensional: `a = np.array([[[1, 2, 3],
    [4, 5, 6]]], float)`
Arrays shapes

```
a.shape == (3, 4)
```

```
[ 1  2  3  4 ]
[ 5  6  7  8 ]
[ 9 10 11 12 ]
```

```
a.sum(axis=0)?
a.sum(axis=1)?
```
Arrays shapes

\[ a.\text{shape} == (3, 4) \]

\[
\begin{bmatrix}
1 & 2 & 3 & 4 \\
5 & 6 & 7 & 8 \\
9 & 10 & 11 & 12
\end{bmatrix}
\]

axis=0

\[ a.\text{sum}(\text{axis}=0) \]
\[ a.\text{sum}(\text{axis}=1) \]
Arrays shapes

\[ a.\text{shape} == (3, 4) \]

\[
\begin{bmatrix}
1 & 2 & 3 & 4 \\
5 & 6 & 7 & 8 \\
9 & 10 & 11 & 12
\end{bmatrix}
\]

axis=0

axis=1

\[ a.\text{sum}(\text{axis}=0)? \]
\[ a.\text{sum}(\text{axis}=1)? \]
Arrays, reshape

```python
a = a.reshape((4, 3))
```

$$
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9 \\
10 & 11 & 12
\end{bmatrix}
$$
Arrays, reshape

```
a = a.reshape((-1, 6))
```

```
1  2  3  4  5  6
7  8  9 10 11 12
```
Other ways to create arrays

- `a = np.zeros(5)`
  
  `[0., 0., 0., 0., 0.]`
Other ways to create arrays

- a = np.zeros(5)
  
  [0., 0., 0., 0., 0.]

- a = np.arange(0, 10, 2)
  
  [0, 2, 4, 6, 8]
Other ways to create arrays

- `a = np.zeros(5)`
  
  \[0., 0., 0., 0., 0.\]

- `a = np.arange(0, 10, 2)`
  
  \[0, 2, 4, 6, 8\]

- `a = np.full((2, 2), 2)`
  
  \[
  \begin{bmatrix}
  2 & 2 \\
  2 & 2 \\
  \end{bmatrix}
  \]
Array ↔ list conversions

**list to array**

```python
lst = [1, 2, 3]
a = np.asarray(lst)
```

**array to list**

```python
a = np.array([1, 2, 3], int)
lst = a.tolist()
```
Useful operations

- `sum`, `mean`
- `np.var`, `np.std`
- `max`, `min`, `argmax`, `argmin`
- `zeros_like()`, `ones_like()`
- `concatenate`
Again, just like matplotlib, read the docs!!!

https://docs.scipy.org/doc/numpy/
Try it!

Practice

You are given a matrix with each row as a vector. Find the index of the row which has the smallest $L_2$ norm.

As a reference, for any vector $\vec{v}$, its $L_2$ norm is defined as:

$$||\vec{v}||_2 = \sqrt{\sum_{k=1}^{n} v_k^2}$$

example

```python
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
return 0
```
A solution with just Python

import math

def l2_norm(lst):
    sum = 0
    for i in lst:
        sum += i ** 2
    return math.sqrt(sum)

smallest = None
idx = None
for i in range(len(matrix)):
    l2 = l2_norm(matrix[i])
    if smallest is None or l2 < smallest:
        idx = i
        smallest = l2
print "index: ", i
import numpy as np
from numpy.linalg import norm

class Matrix:
    def __init__(self, matrix):
        self.matrix = np.asarray(matrix)

    def get_all_norms(self, axis=1):
        all_norms = norm(self.matrix, axis=axis)
        return all_norms;

matrix = Matrix(matrix)
all_norms = matrix.get_all_norms()
print "index: ", all_norms.argmin()
Final word on NumPy: Vectorization

Do not waste NumPy’s awesome performance by writing for loops on them!

```python
a = np.arange(10000).reshape((-1, 2))
# square entries
for i in range(len(a)):
    for j in range(len(a[i])):
        a[i][j] = a[i][j] ** 2
```
Final word on NumPy: Vectorization

Do not waste NumPy’s awesome performance by writing for loops on them!

for loop

```python
a = np.arange(10000).reshape((-1, 2))
# square entries
for i in range(len(a)):
    for j in range(len(a[i])):
        a[i][j] = a[i][j] ** 2
```

vectorization code

```python
a = np.arange(10000).reshape((-1, 2))
# square entries
a = a * a
```
Now let’s switch to SciPy!

- `scipy.cluster`
- `scipy.constants`
- `scipy.fftpack`
- `...`
- `scipy.signal`
- `scipy.stats`

Vector quantization / Kmeans

- Physics/Math constants
- Fourier Transform
- Signal Processing
- Statistics
SciPy is built on NumPy

- You need to know how to deal with NumPy arrays to be comfortable with SciPy functions.
- Depending on your need, you can almost find anything in it!
- Commonly used by me: stats, optimize, signal
In SciPy: define objective function, and the constraints

```python
def objective(x):
    return 0.5 * (x ** 2)

def constraint(x):
    # unlike definition (<=0), scipy constraints are >= 0
    return x + 10
```
x0 = 0
cons = {'type': 'ineq', 'fun': constraint}
# minimize
minimize(objective, x0, method="SLSQP",
        constraints=cons)

Would still work on non-convex constraints such as $||\vec{v}||_2 = 0$
Hypothesis testing: \( p \) values

```python
from scipy.stats import ttest_ind
import numpy as np

# two independent random variables
X = np.random.rand(10, 1)
Y = np.random.rand(10, 1)
# T test (two tailed p value)
t, p = ttest_ind(X, Y)
```