Bootcamp Python

Day03
NumPy

Bootcamp Python

Day03 - NumPy

Today you will learn how to use the Python library that will allow you to manipulate multidimensional arrays (vectors, matrices, tensors...) and perform complex mathematical operations on them.

Notions of the day

NumPy array, slicing, stacking, dimensions, broadcasting, normalization, etc...

General rules

- Use the NumPy Library: use NumPy's built-in functions as much as possible. Here you will be given no credit for reinventing the wheel.
- The version of Python to use is 3.7, you can check the version of Python with the following command: `python -V`
- The norm: during this bootcamp you will follow the PEP 8 standards. You can install pycodestyle which is a tool to check your Python code.
- The function eval is never allowed.
- The exercises are ordered from the easiest to the hardest.
- Your exercises are going to be evaluated by someone else, so make sure that your variable names and function names are appropriate and civil.
- Your manual is the internet.
- You can also ask questions in the dedicated channel in the 42 AI Slack: 42-ai.slack.com.
- If you find any issue or mistakes in the subject please create an issue on our dedicated repository on Github.

Helper

For this day you will use the image provided in the `resources` folder

Ensure that you have the right Python version.

```
> which python
/goinfre/miniconda/bin/python
> python -V
Python 3.7.*
> which pip
/goinfre/miniconda/bin/pip
```

Exercise 00 - NumPyCreator

Exercise 01 - ImageProcessor

Exercise 02 - ScrapBooker

Exercise 03 - ColorFilter

Exercise 04 - K-means Clustering

Exercise 00 - NumPyCreator

Turn-in directory:	ex00
Files to turn in:	NumPyCreator.py
Allowed libraries:	NumPy
Remarks:	n/a

Write a class named `NumPyCreator`, that implements all of the following methods.
Each method receives as an argument a different type of data structure and transforms it into a NumPy array:

- `from_list(lst)` : takes in a list and returns its corresponding NumPy array.
- `from_tuple(tpl)` : takes in a tuple and returns its corresponding NumPy array.
- `from_iterable(itr)` : takes in an iterable and returns an array which contains all of its elements.
- `from_shape(shape, value)` : returns an array filled with the same value.
  The first argument is a tuple which specifies the shape of the array, and the second argument specifies the value of all the elements. This value must be 0 by default.
- `random(shape)` : returns an array filled with random values.
  It takes as an argument a tuple which specifies the shape of the array.
- `identity(n)` : returns an array representing the identity matrix of size n.

BONUS: Add to those methods an optional argument which specifies the datatype (dtype) of the array (e.g. to represent its elements as integers, floats, ...)

NB: All those methods can be implemented in one line. You only need to find the right NumPy functions.

```
>>> from NumPyCreator import NumPyCreator
>>> npc = NumPyCreator()

>>> npc.from_list([[1,2,3],[6,3,4]])
array([[1, 2, 3],
       [6, 3, 4]])

>>> npc.from_tuple(("a", "b", "c"))
array(['a', 'b', 'c'])

>>> npc.from_iterable(range(5))
array([0, 1, 2, 3, 4])

>>> shape=(3,5)
>>> npc.from_shape(shape)
array([[0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0]])

>>> npc.random(shape)
array([[0.57055863, 0.23519999, 0.56209311, 0.79231567, 0.213768  ],
      [0.39608366, 0.18632147, 0.80054602, 0.44905766, 0.81313615],
      [0.79585328, 0.00660962, 0.92910958, 0.9905421 , 0.05244791]])

>>> npc.identity(4)
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

Exercise 01 - ImageProcessor

Turn-in directory:	ex01
Files to turn in:	ImageProcessor.py
Forbidden functions:	None
Helpful libraries:	Matplotlib

Build a tool that will be helpful to load and display images in the upcoming exercises.

Write a class named `ImageProcessor` that implements the following methods:

- `load(path)` : opens the .png file specified by the `path` argument and returns an array with the RGB values of the image pixels It must display a message specifying the dimensions of the image (e.g. 340 x 500).
- `display(array)` : takes a NumPy array as an argument and displays the corresponding RGB image.

NB: You can use the library of your choice for this exercise, but converting the image to a NumPy array is mandatory. The goal of this exercise is to dispense with the technicality of loading and displaying images, so that you can focus on array manipulation in the upcoming exercises.

```
>>> from ImageProcessor import ImageProcessor
>>> imp = ImageProcessor()
>>> arr = imp.load("../resources/42AI.png")
Loading image of dimensions 200 x 200
>>> arr
array([[[0.03529412, 0.12156863, 0.3137255 ],
        [0.03921569, 0.1254902 , 0.31764707],
        [0.04313726, 0.12941177, 0.3254902 ],
        ...,
        [0.02745098, 0.07450981, 0.22745098],
        [0.02745098, 0.07450981, 0.22745098],
        [0.02352941, 0.07058824, 0.22352941]],

       [[0.03921569, 0.11764706, 0.30588236],
        [0.03529412, 0.11764706, 0.30980393],
        [0.03921569, 0.12156863, 0.30980393],
        ...,
        [0.02352941, 0.07450981, 0.22745098],
        [0.02352941, 0.07450981, 0.22745098],
        [0.02352941, 0.07450981, 0.22745098]],

       [[0.03137255, 0.10980392, 0.2901961 ],
        [0.03137255, 0.11372549, 0.29803923],
        [0.03529412, 0.11764706, 0.30588236],
        ...,
        [0.02745098, 0.07450981, 0.23137255],
        [0.02352941, 0.07450981, 0.22745098],
        [0.02352941, 0.07450981, 0.22745098]],

       ...,

       [[0.03137255, 0.07450981, 0.21960784],
        [0.03137255, 0.07058824, 0.21568628],
        [0.03137255, 0.07058824, 0.21960784],
        ...,
        [0.03921569, 0.10980392, 0.2784314 ],
        [0.03921569, 0.10980392, 0.27450982],
        [0.03921569, 0.10980392, 0.27450982]],
```

```
       [[0.03137255, 0.07058824, 0.21960784],
        [0.03137255, 0.07058824, 0.21568628],
        [0.03137255, 0.07058824, 0.21568628],
        ...,
        [0.03921569, 0.10588235, 0.27058825],
        [0.03921569, 0.10588235, 0.27058825],
        [0.03921569, 0.10588235, 0.27058825]],

       [[0.03137255, 0.07058824, 0.21960784],
        [0.03137255, 0.07058824, 0.21176471],
        [0.03137255, 0.07058824, 0.21568628],
        ...,
        [0.03921569, 0.10588235, 0.26666668],
        [0.03921569, 0.10588235, 0.26666668],
        [0.03921569, 0.10588235, 0.26666668]]], dtype=float32)
>>> imp.display(arr)
```

Exercise 02 - ScrapBooker

Turn-in directory:	ex02
Files to turn in:	ScrapBooker.py
Allowed libraries:	NumPy
Notions:	Slicing

Write a class named `ScrapBooker` that implements the following methods.
All methods take in a NumPy array and return a new modified one.
We are assuming that all inputs are correct, i.e. you don't have to protect your functions against input errors.

- `crop(array, dimensions, position)` : crops the image as a rectangle with the given `dimensions` (meaning, the new height and width for the image), whose top left corner is given by the `position` argument. The position should be (0,0) by default. You have to consider it an error (and handle said error) if `dimensions` is larger than the current image size.
- `thin(array, n, axis)` : deletes every n-th pixel row along the specified axis (0 vertical, 1 horizontal), example below.
- `juxtapose(array, n, axis)` : juxtaposes `n` copies of the image along the specified axis (0 vertical, 1 horizontal).
- `mosaic(array, dimensions)` : makes a grid with multiple copies of the array. The `dimensions` argument specifies the dimensions (meaning the height and width) of the grid (e.g. 2x3).

NB: In this exercise, when specifying positions or dimensions, we will assume that the first coordinate is counted along the vertical axis starting from the TOP, and that the second coordinate is counted along the horizontal axis starting from the left. Indexing starts from 0.

e.g.:
(1,3)
. . . . .
. . . x.
. . . . .

example for thin:

```
perform thin with n=3 and axis=0:
ABCDEFGHIJQL        ABDEGHJQ
ABCDEFGHIJQL        ABDEGHJQ
ABCDEFGHIJQL        ABDEGHJQ
ABCDEFGHIJQL        ABDEGHJQ
ABCDEFGHIJQL        ABDEGHJQ
ABCDEFGHIJQL  ==>   ABDEGHJQ
ABCDEFGHIJQL        ABDEGHJQ
ABCDEFGHIJQL        ABDEGHJQ
ABCDEFGHIJQL        ABDEGHJQ
ABCDEFGHIJQL        ABDEGHJQ
ABCDEFGHIJQL        ABDEGHJQ

perform thin with n=4 and axis=1:
AAAAAAAAAAAA
BBBBBBBBBBBB        AAAAAAAAAAAA
CCCCCCCCCCCC        BBBBBBBBBBBB
DDDDDDDDDDDD        CCCCCCCCCCCC
EEEEEEEEEEEE        EEEEEEEEEEEE
FFFFFFFFFFFF  ==>   FFFFFFFFFFFF
GGGGGGGGGGGG        GGGGGGGGGGGG
HHHHHHHHHHHH        IIIIIIIIIIII
IIIIIIIIIIII        JJJJJJJJJJJJ
JJJJJJJJJJJJ        KKKKKKKKKKKK
```

```
KKKKKKKKKKK
LLLLLLLLLLL
```

Exercise 03 - ColorFilter

Turn-in directory:	ex03
Files to turn in:	ColorFilter.py
Forbidden functions:	See each method
Notions:	Broadcasting

Now you will build a tool that can apply a variety of color filters on images. For this exercise, the authorized functions and operators are specified for each methods. You are not allowed to use anything else.

Write a class named `ColorFilter` which implements the following methods:

- `invert(array)` : Takes a NumPy array of an image as an argument and returns an array with inverted color.
  Authorized functions: None
  Authorized operator: -
- `to_blue(array)` : Takes a NumPy array of an image as an argument and returns an array with a blue filter.
  Authorized functions: .zeros, .shape
  Authorized operator: None
- `to_green(array)` : Takes a NumPy array of an image as an argument and returns an array with a green filter.
  Authorized functions: None
  Authorized operator: *
- `to_red(array)` : Takes a NumPy array of an image as an argument and returns an array with a red filter.
  Authorized functions : to_green, to_blue
  Authorized operator: -, +
- `to_celluloid(array)` : Takes a NumPy array of an image as an argument, and returns an array with a celluloid shade filter. The celluloid filter must display at least four thresholds of shades. Be careful! You are not asked to apply black contour on the object here (you will have to, but later. . .), you only have to work on the shades of your images.
  Authorized functions: .arange, linspace

`Bonus`: add an argument to your method to let the user choose the number of thresholds.
Authorized functions: .vectorize, .arange
Authorized operator: None

- `to_grayscale(array, filter)` : Takes a NumPy array of an image as an argument and returns an array in grayscale. The method takes another argument to select between two possible grayscale filters. Each filter has specific authorized functions and operators.
  - 'mean' or 'm' : Takes a NumPy array of an image as an argument and returns an array in grayscale created from the mean of the RBG channels.
    Authorized functions: .sum, .shape, reshape, broadcast_to, as_type
    Authorized operator: /
  - 'weighted' or 'w' : Takes a NumPy array of an image as an argument and returns an array in weighted grayscale. This argument should be selected by default if not given.
    The usual weighted grayscale is calculated as : 0.299 * R_channel + 0.587 * G_channel + 0.114 * B_channel.
    Authorized functions: .sum, .shape, .tile
    Authorized operator: *

```
>>> from ImageProcessor import ImageProcessor
>>> imp = ImageProcessor()
>>> arr = imp.load("../42AI.png")
Loading image of dimensions 200 x 200
>>> from ColorFilter import ColorFilter
```

```
>>> cf = ColorFilter()
>>> cf.invert(arr)
>>>
>>> cf.to_green(arr)
>>>
>>> cf.to_red(arr)
>>>
>>> cf.to_blue(arr)
>>>
>>> cf.to_celluloid(arr)
>>>
>>> cf.to_grayscale(arr, 'm')
>>>
>>> cf.to_grayscale(arr, 'weighted')
>>>
```

Examples

From this base image:

Figure 1: Elon Musk

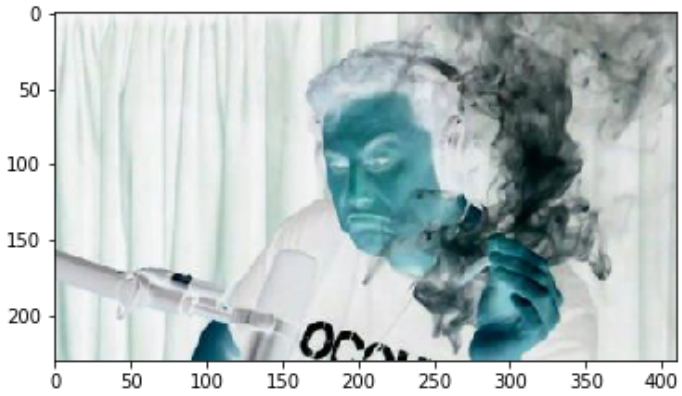

Figure 2: invert

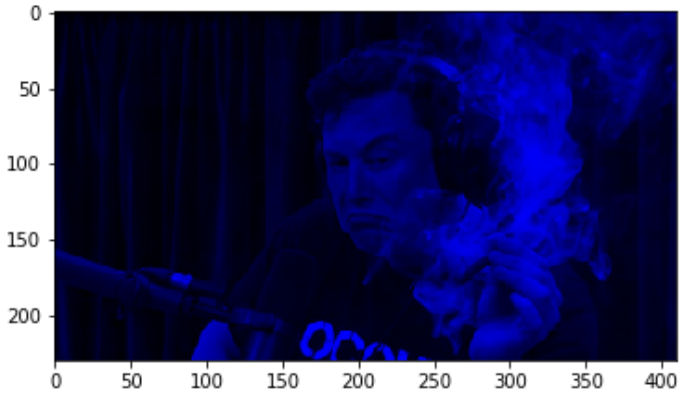

Figure 3: to_blue

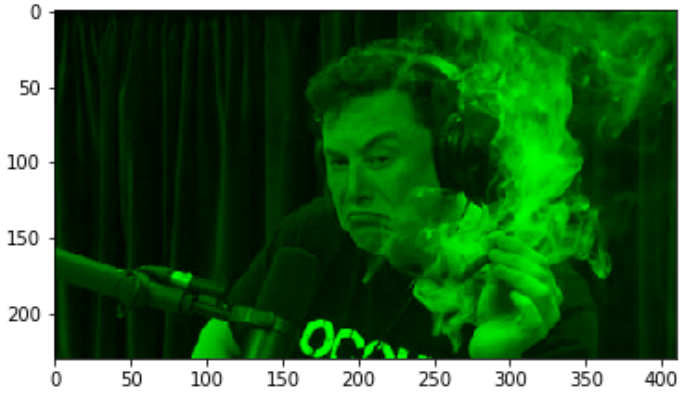

Figure 4: to_green

Figure 5: to_red

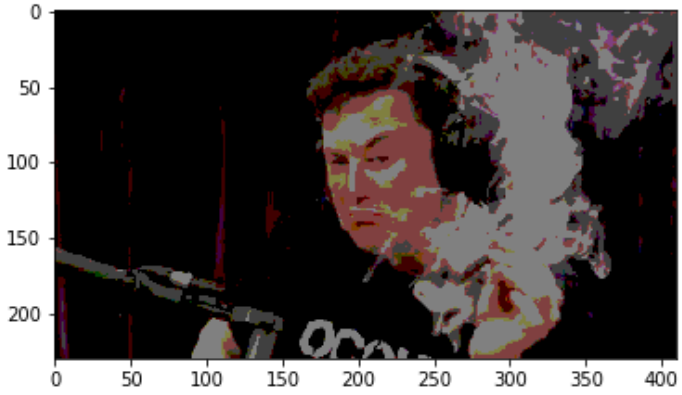

Figure 6: celluloid

Exercise 04 - K-means Clustering

Turn-in directory:	ex04
Files to turn in:	Kmeans.py
Forbidden functions:	None
Remarks:	n/a

ALERT! DATA CORRUPTED

Objective:

The solar system census dataset is corrupted! The citizens' homelands are missing!
You must implement the K-means clustering algorithm in order to recover the citizens' origins.

On this web-page you can find good explanations on how K-means is working:
Possibly the simplest way to explain K-Means algorithm

The missing part is how to compute the distance between 2 data points (cluster centroid or a row in the data). In our case the data we have to process is composed of 3 values (height, weight and bone_density). Thus, each data point is a vector of 3 values.
Now that we have mathematically defined our data points (vector of 3 values), it is then very easy to compute the distance between two points using vector properties. You can use L1 distance, L2 distance, cosine similarity, and so forth... Choosing the distance to use is called hyperparameter tuning. I would suggest you to try with the easiest setting (L1 distance) first.
What you will notice is that the final result of the "training"/"fitting" will depend a lot on the random initialization. Commonly, in machine-learning libraries, K-means is run multiple times (with different random initializations) and the best result is saved.

NB: To implement the fit function, keep in mind that a centroid can be considered as the gravity center of a set of points.

Instructions:

Create the class `KmeansClustering` with the following methods:

```
class KmeansClustering:
    def __init__(self, max_iter=20, ncentroid=5):
        self.ncentroid = ncentroid # number of centroids
        self.max_iter = max_iter # number of max iterations to update the centroids
        self.centroids = [] # values of the centroids

    def fit(self, X):
        """
        Run the K-means clustering algorithm.
        For the location of the initial centroids, random pick ncentroids from the dataset.
        Args:
          X: has to be an numpy.ndarray, a matrice of dimension m * n.
        Returns:
          None.
        Raises:
          This function should not raise any Exception.
        """

    def predict(self, X):
        """
        Predict from wich cluster each datapoint belongs to.
        Args:
          X: has to be an numpy.ndarray, a matrice of dimension m * n.
        Returns:
          the prediction has a numpy.ndarray, a vector of dimension m * 1.
        Raises:
```

```
        This function should not raise any Exception.
    """
```

Dataset:

The dataset, named **solar_system_census** can be found in the resources folder.
It is a part of the solar system census dataset, and contains biometric informations such as the height, weight, and bone density of solar system citizens.

As you should know solar citizens come from four registered areas: The flying cities of Venus, United Nations of Earth, Mars Republic, and the Asteroids' Belt colonies.

Unfortunately the data about the planets of origin was lost...
Use your K-means algorithm to recover it!
Once your clusters are found, try to find matches between clusters and the citizens' homelands.

Hints:

- People are slender on Venus than on Earth.
- People of the Martian Republic are taller than on Earth.
- Citizens of the Belt are the tallest of the solar system and have the lowest bone density due to the lack of gravity.

Example:

Here is an exemple of the algorithm in action:
K-means animation