EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Philosophische Fakultät
Seminar für Sprachwissenschaft

Data Science for Linguists

Session 2: NumPy

Johannes Dellert

3 November, 2023

Table of Contents

NumPy: Purpose

- main uses of NumPy for data analyis:
  - fast array-based operations for data munging and cleaning, subsetting and filtering
  - common array algorithms like sorting, unique, set operations
  - efficient descriptive statistics and data aggregation/summarisation
  - data alignment and relational data manipulations for merging heterogeneous datasets
  - conditional logic as array expressions instead of loops with if-then-else branches
  - group-wise data manipulations (aggregation, transformation, function application)
  - backbone of most other packages we are going to see in this course

Basic Usage

- convention for import (this is what we will assume throughout this course):
  `import numpy as np`

Table of Contents

NumPy Arrays: Creation

- the NumPy array object is much quicker and less memory-consuming than nested lists in core Python because it places the values next to each other in memory, allowing the underlying computations without to occur without type checking and other types of overhead
- **`np.arrays(data)`** is the main constructor, it infers the database from the structure of data
- onedimensional arrays (= row vectors) can be constructed directly from any sequence:
  `vec = np.array([1.5, -0.1, 3, 0, -3, 6.5])`

- twodimensional arrays can be created directly from nested lists of numbers (of same length):
  `arr = np.array([[1.5, -0.1, 3], [0, -3, 6.5]])`

- `data.ndim`, `data.shape`, `data.dtype`
- `np.zeros(10)` and `np.zeros((3, 6))`, `np.empty()` returns uninitialized memory [saves time if all will be filled anyway, but no guarantee of any default value]
- `np.arange(15)` as the numpy equivalent of `range(15)`
- `np.ones()`, `np.full()` with fill value, `np.zeros_like()` [copy of same dimensions with zeros in every entry], `np.identity()` for identity matrices

NumPy Arrays: Data Types

- possibility to specify the exact datatype,
  e.g. `np.float32` or `np.bool` for lower precision or binary matrix
- casting between types using **`arr.astype`**`(np.float64)`,
  e.g. string to float [creates new array!]
- fixed-length stringtype `numpy.string_` with truncation properties

NumPy Arrays: Vectorisation and Broadcasting

- vectorisation for element-wise application (can be done between equal-sized arrays): `arr + arr`, `(arr * arr)`
- propagating a scalar argument to each element in an array, e.g. `arr * 10`, `(1 / arr)`
- more general **broadcasting** (evaluating operations between differently sized arrays) is complicated, but powerful

NumPy Arrays: Indexing and Slicing

- slicing as view on the original array (unlike Python, were slicing means copying; `arr[x:y].copy()` for the original Python behaviour)
- broadcast when assigning scalar value to a slice (instead of TypeError)
- comma-separated list of indices to select individual elements in multidimensional array (`arr[x, y]`)
- subarray extraction like `arr3d[3]` or `arr3d[1, 0]`
  (again, views of the original array; not possible in nested lists)
- slicing along axes, e.g. `arr2d[:2]` “select first two rows”
- multiple slices `arr2d[:2, 1:]`
- lowerdimensional slice `arr2d[1, :2]`
- `arr[:, :1]` for slicing the first column
- meaning of shape `(2,)` vs `(1,2)`, i.e. `arr[1, :2]` vs `arr[1:2, :2]`

NumPy Arrays: Boolean Indexing and Filtering

- comparison between arrays (`arr2 > arr`) yields Boolean arrays
- assume we have a vector `names` of names for each row in `data`
- **Boolean indexing**: `names == "Bob"`, `data[names == "Bob"]`, `!=` or `~( == )` [~for inverting a Boolean array], &, | [NOT and/or]
- `data[data < 0] = 0` to set all negative values to 0
- **fancy indexing**: indexing using integer arrays, e.g. `arr[[4, 3, 0, 6]]`
  - multiple index array for selection corresponding to each tuple of indices
  - `arr[l1][:,l2]` for selecting subset of rows and columns

NumPy Arrays: Sorting and Searching

- `arr.sort()` sorts in place
- `arr.sort(axis=0)` sorts values within each column
- `arr.sort(axis=1)` sorts values across each row
- `arr.argsort()` returns the indices that would sort an array (important for sorting by a certain column)
- `np.sort(arr)` returns sorted copy of the array (like built-in `sorted()`)
- `np.unique(arr)`
- `np.in1d(values, arr)` returns Boolean vector whether each value is in `arr`
- `np.intersect1d(x, y)`: sorted, common elements
- `np.union1d(x, y)`: sorted union of elements

Table of Contents

Pseudorandom Numbers with `np.random`

- more efficient generation of whole arrays of sample values from many kinds of distributions, e.g. `np.random.standard_normal(size=(4,4))`
- `np.random.permutation()`
- `np.random.shuffle()`
- `np.random.uniform()`
- `np.random.integers()`
- `np.random.binomial()`, `np.random.normal()`, `np.random.beta()`

Random Seeds

- a pseudorandom number generator with a seed will always generate the same sequence of numbers (good practice for reproducibility!)
- `rng = np.random.default_rng(seed=12345)`
- `rng.standard_normal()` etc.

Table of Contents

Unary Universal Functions

- **universal functions** are fast element-wise array functions
- examples of useful unary universal functions:
  - `np.sqrt(arr)`
  - `np.exp(arr)`
  - `np.log(arr)`
  - `np.cos(arr)`

Binary Universal Functions

- examples of useful binary universal functions:
  - `np.add(arr1, arr2)`
  - `np.multiply(arr1, arr2)`
  - `np.greater(arr1, arr2)`
  - `np.maximum(arr1, arr2)`

Array-Oriented Programming

- evaluate function across regular grid of values, find minimum (grid search)
  - `np.arange(-5, 5, 0.01)`
  - `xs, ys, zs = np.meshgrid(points, points, points)`
  - `vals = np.sqrt(xs ** 2 + ys ** 2 + zs ** 2)`
  - `min = np.argmin(vals, keepdims=True)`
- `np.where(cond, xarr, yarr)`

Table of Contents

File Input and Output with Arrays

- NumPy has a special `.npy` file format for storing arrays in uncompressed archive
  - `np.save("some_array", arr)`
  - `np.load("some_array.npy")`
- there is also way of storing several arrays under ids in a common file (`.npz` format)
- to load in array data from a CSV file: np.genfromtxt("csv.txt", delimiter=",") with many options (skip header, only load certain columns, ... check online documentation!)

Table of Contents

Linear Algebra

- Matrix multiplication: `x.dot(y)` or `np.dot(x,y)` or `x @ y`
- transpose `mtx.T` (swaps dimensions)
- inverse `np.linalg.inv(mtx)`
- QR decomposition `np.linalg.qr(mtx)` (orthonormal times upper triangular matrix)
- trace `np.linalg.trace(mtx)` (sum of diagonal elements)
- determinant `np.linalg.det(mtx)` ("volume compression factor" of transformation)
- eigenvalues and eigenvectors `np.linalg.eig(mtx)` ("invariants" of transformation)
- `np.linalg.solve(amtx, bvect)` for solving set $Ax = b$ of linear equations

Table of Contents

Assignment 2: Tasks (Part 2)

1) Create a new Jupyter notebook and import `numpy`. Load the contents of the second, fourth, and sixth column in `Kuperman-BRM-data-2012.csv` into a NumPy array, not forgetting to skip the header (you cannot work with mixed data yet). Each row now contains age-of-acquisition data for one English word. The first column will be the number of participants with known age of acquisition for the word, the second column the average age of acquistion, and the third column will be a frequency count in the SUBTLEX-US corpus.
2) Normalize the last column in place in order to turn raw frequencies into percentages.
3) How many English words do children acquire in the first five years of their lives?
4) What percentage of tokens in the corpus would a seven-year-old child understand? (Hint: filter by age of acquisition, then add up)
5) How many of the top-5000 most frequent words (B2 level) can we expect a ten-year old native speaker to know? (Hint: use `argsort()` to sort by last column, select last 5000 entries, then filter by age)

Assignment 2: Tasks (Part 2)

6) By how many points does the percentage of understood tokens grow in every year? What ages are the peak years of increase in comprehension? (Hint: list comprehension, `np.diff()`)
7) By how many words does the lexicon of a child grow in every year of its life? Is there a peak and a tendency? (Hint: `np.where()` with two conditions in list comprehension)
8) How many words are learnt prematurely compared to a learning sequence which would optimise for the highest possible coverage early on (i.e. acquiring words by order of frequency)?
9) If the child randomly selected the same number of words it learnt (without any frequency effect), how much faster would we expect the coverage of tokens in the corpus to develop?

Preliminary Course Plan

1	**27/10**	**IPython and Jupyter**
2	**03/11**	**Introduction to NumPy**
3	10/11	Pandas and Data Frames
4	17/11	Data Cleaning and Preparation
5	24/11	Linguistic Preprocessing
6	01/12	Data Wrangling: Join, Combine, Reshape
7	08/12	Data Aggregation and Grouping
8	15/12	Visualisation with Seaborn
9	22/12	Modeling and Prediction
10	12/01	Classification
11	19/01	Clustering
12	26/01	Pattern Extraction and Density Estimation
13	02/02	Statistical Inference
14	09/02	Data Science Projects

Questions

Questions?

Comments?

Suggestions?