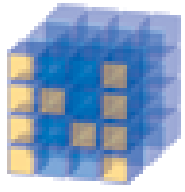
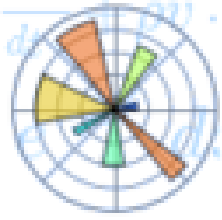
Python für Machine Learning

Scipy

- Sammlung von python-basierter Open Source Software für Mathematik und Naturwissenschaften
- Beinhaltet

NumPy	Numerik-Paket, insbesondere für Vektor- und Matrixrechnung
Scipy	Sammlung von Algorithmen und Werkzeugen
Matplotlib	Paket zum Plotten
IPython	Interaktive Shell und Browserbasierte Notebooks
Weitere	Sympy, Pandas

IP[y]:
IPython

http://www.scipy.org/

Warum Scipy?

- Stellt viele Funktionalitäten zur Verfügung, die Python so nicht anbietet (z.B. Operationen auf Vektoren/Matrizen, höhere Mathematik)
- Schnittstelle zu effizienten C/C++-Libraries
- Warum interessant für Computerlinguisten?
  - Oft Vektorrepräsentation benötigt (Feature-Vektor, Kontextvektor)
  - Große Datenmengen $\rightarrow$ Speicher- und Laufzeiteffizienz notwendig
  - Toolkit für maschinelles Lernen (`scikit-learn`) arbeitet mit SciPy-Funktionen und Datenstrukturen

Ziel für den nächsten Teil: mithilfe von Numpy/Scipy unsere Mini-Suchmaschine optimieren (Geschwindigkeit mit Profiler messen!)

Python für Machine Learning

NumPy[3]: `ndarrays`

- Wichtigste Daten-Container in Python: *list* und *dictionary*
- NumPy stellt eine weitere Datenstruktur zur Verfügung: den mehrdimensionalen Array `ndarray`
- Unterschied zur Liste: `list` kann verschiedene Arten von Einträgen haben, `ndarray` erlaubt nur Einträge vom selben Typ (integer, floats, strings)
- Beliebig viele Dimensionen möglich

[3] Dieser Teil basiert auf Bressert, Eli: *NumPy and SciPy*. O'Reilly, 2013.

NumPy: ndarray erzeugen

Array kann auf verschiedene Weisen erzeugt werden.

```
import numpy as np

# eindimensionaler Array
myarr1 = np.array([1, 2, 3, 4])

# zweidimensionaler Array
myarr2 = np.array([[1, 2, 3],
                   [4, 5, 6]])

# 3x4-Array, alle Eintraege sind 0
myarr3 = np.zeros((3, 4))

# Array mit 100K Elementen
myarr4 = np.arange(100000)

# Integer-Array
myarr5 = np.zeros(100, dtype=int)
```

Arrays: Indexing und Slicing

Beispiel (Liste)

```
mylist = [[1, 2], [3, 4]]
elem = mylist[0][1]
```

Beispiel (Array)

```
# Array aus Liste initialisieren
myarr = np.array(mylist)

elem = myarr[0, 1]
column1 = myarr[:,0]
row2 = myarr[1,:]
```

Operationen auf Arrays

Einfache Rechenoperationen werden elementweise ausgeführt.

```
>>> a = np.array([1, 2, 3, 4])
>>> b = np.array([5, 6, 7, 8])
>>> a * b
array([ 5, 12, 21, 32])
>>> b - a
array([4, 4, 4, 4])
>>> a ** 2
array([ 1,  4,  9, 16])
```

NumPy: Lineare Algebra I

Mit Arrays

```
>>> np.dot(a, b) # Skalarprodukt von a und b
70
>>> m = a.reshape((2, 2))
>>> m
array([[1, 2],
       [3, 4]])
>>> m.T  # m transponiert
array([[1, 3],
       [2, 4]])
```

NumPy: Lineare Algebra II

Matrizen

```
>>> m = np.matrix([[1, 2], [3, 4]])
>>> n = np.matrix([[5, 6], [7, 8]])
>>> m * n # Matrix-Multiplikation
matrix([[19, 22],
        [43, 50]])
```

- `matrix`-Objekte können nur 2 Dimensionen haben.
- Auf `matrix`-Objekt wird standardmäßig Matrixmultiplikation ausgeführt, nicht elementweise Multiplikation.

NumPy: Universal Functions (ufunc)

- ufuncs sind mathematische Funktionen, die elementweise auf Arrays angewendet werden:

```
np.add(a,b) # wird auch aufgerufen bei 'a + b'
np.exp(a)
np.sqrt(a)
np.log(a)
```

- Das normale math-Modul kann nicht auf ndarrays angewendet werden.

numpy.random

- Zufallsgenerator für ndarray-Objekte

```
# 2 x 3 Samples von Standardnormalverteilung
>>> np.random.randn(2, 3)
array([[-1.12695503,  0.21605989, -1.73561989],
       [ 0.38781853, -1.98914298, -0.55159214]])
# 10 Integers zwischen 1 und 10
>>> np.random.randint(1, 10 + 1, 10)
array([ 1,  8,  5,  5,  4,  4, 10,  3,  8,  8])
```

- Viele Wahrscheinlichkeitsverteilungen vorhanden (Multinomialverteilung, Dirichlet-Verteilung, Zipf-Verteilung, ...)

Dokumentation

- NumPy Tutorial:
  `http://wiki.scipy.org/Tentative_NumPy_Tutorial`
- NumPy Befehlsreferenz:
  `http://docs.scipy.org/doc/numpy/reference/`
- `help()`-Funktion und Doc-Strings

Laufzeitvergleich I

Generierung von Pseudodaten mit Python:

```
Total time: 3.73401 s
File: tiny_search_engine.py
Function: pseudo_data at line 32

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    32                                           @profile
    33                                           def pseudo_data(n):
    34      1001          433      0.4      0.0      v = []
    35   1002001       289524      0.3      7.8      for i in range(n):
    36   1001000      3083360      3.1     82.6          freq = random.randint(0, 100)
    37   1001000       360374      0.4      9.7          v.append(freq)
    38      1001          318      0.3      0.0      return v
```

Laufzeitvergleich II

Generierung von Pseudodaten mit NumPy:

```
def pseudo_data_numpy(n):
    v = np.random.random_integers(0, 100, n)
    return v
```

```
Total time: 0.131629 s
File: tiny_search_engine.py
Function: pseudo_data_numpy at line 40

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    40                                           @profile
    41                                           def pseudo_data_numpy(n):
    42      1001       131170    131.0     99.7      v = np.random.random_integers(0, 100, l)
    43      1001          459      0.5      0.3      return v
```

$\rightarrow$ 28 mal schneller!

Python für Machine Learning

Was bietet die SciPy-Library?

- Benutzt NumPy-Arrays für Standardaufgaben aus verschiedenen naturwissenschaftlichen Disziplinen
- Unter anderem:
  - Optimierung (`scipy.optimize`)
  - Statistik (`scipy.stats`)
  - Räumliche Strukturen und Distanzen (`scipy.spatial`)
  - Signal- und Bildverarbeitung (`scipy.signal`, `scipy.ndimage`)
  - Clustering (`scipy.cluster`)
  - Dünn besetzte (sparse) Matrizen (`scipy.sparse`)

Dokumentation

- Scipy Referenz
  `http://docs.scipy.org/doc/scipy/reference/`
- Buch: Bressert, Eli. SciPy and NumPy. O'Reilly, 2012.

Übung 14