Donnerstag: Python für Machine Learning

8 Python für Machine Learning
- Profiling
- Numpy und SciPy
- Die Numpy Package
- Die SciPy-Library
- scikit-learn

Python für Machine Learning

8 Python für Machine Learning
- Profiling
- Numpy und SciPy
- Die Numpy Package
- Die SciPy-Library
- scikit-learn

Profiling - Warum?[1]

- Viele Ursachen sind möglich, wenn ein Programm langsam läuft.
- Zeitaufwand minimieren durch Optimierung an der richtigen Stelle!
- Beim Profiling misst man den Ressourcenverbrauch eines Programms (Berechnungszeit, RAM).
- Faustregel: Erst profilen, dann optimieren.
  - Wie lange läuft das Programm?
  - Wie oft wird ein Funktion/Zeile ausgeführt?
  - Wieviel Speicher benötigt eine Datenstruktur?
- Optimierung an der falschen Stelle $\rightarrow$ großer Aufwand, kaum Verbesserungen

[1] Slides basieren auf Gorelick, Micha und Ozsvald, Ian: *High performance Python*. O'Reilly, 2014.

Profiling - Warum?[1]

- Viele Ursachen sind möglich, wenn ein Programm langsam läuft.
- Zeitaufwand minimieren durch Optimierung an der richtigen Stelle!
- Beim Profiling misst man den Ressourcenverbrauch eines Programms (Berechnungszeit, RAM).
- Faustregel: Erst profilen, dann optimieren.
  - Wie lange läuft das Programm?
  - Wie oft wird ein Funktion/Zeile ausgeführt?
  - Wieviel Speicher benötigt eine Datenstruktur?
- Optimierung an der falschen Stelle $\rightarrow$ großer Aufwand, kaum Verbesserungen

[1] Slides basieren auf Gorelick, Micha und Ozsvald, Ian: *High performance Python*. O'Reilly, 2014.

Profiling - Warum?[1]

- Viele Ursachen sind möglich, wenn ein Programm langsam läuft.
- Zeitaufwand minimieren durch Optimierung an der richtigen Stelle!
- Beim Profiling misst man den Ressourcenverbrauch eines Programms (Berechnungszeit, RAM).
- Faustregel: Erst profilen, dann optimieren.
  - Wie lange läuft das Programm?
  - Wie oft wird ein Funktion/Zeile ausgeführt?
  - Wieviel Speicher benötigt eine Datenstruktur?
- Optimierung an der falschen Stelle $\rightarrow$ großer Aufwand, kaum Verbesserungen

[1] Slides basieren auf Gorelick, Micha und Ozsvald, Ian: *High performance Python*. O'Reilly, 2014.

Profiling - Warum?[1]

- Viele Ursachen sind möglich, wenn ein Programm langsam läuft.
- Zeitaufwand minimieren durch Optimierung an der richtigen Stelle!
- Beim Profiling misst man den Ressourcenverbrauch eines Programms (Berechnungszeit, RAM).
- Faustregel: Erst profilen, dann optimieren.
  - Wie lange läuft das Programm?
  - Wie oft wird ein Funktion/Zeile ausgeführt?
  - Wieviel Speicher benötigt eine Datenstruktur?
- Optimierung an der falschen Stelle $\rightarrow$ großer Aufwand, kaum Verbesserungen

[1] Slides basieren auf Gorelick, Micha und Ozsvald, Ian: *High performance Python*. O'Reilly, 2014.

Profiling - Warum?[1]

- Viele Ursachen sind möglich, wenn ein Programm langsam läuft.
- Zeitaufwand minimieren durch Optimierung an der richtigen Stelle!
- Beim Profiling misst man den Ressourcenverbrauch eines Programms (Berechnungszeit, RAM).
- Faustregel: Erst profilen, dann optimieren.
  - Wie lange läuft das Programm?
  - Wie oft wird ein Funktion/Zeile ausgeführt?
  - Wieviel Speicher benötigt eine Datenstruktur?
- Optimierung an der falschen Stelle $\rightarrow$ großer Aufwand, kaum Verbesserungen

[1] Slides basieren auf Gorelick, Micha und Ozsvald, Ian: *High performance Python*. O'Reilly, 2014.

Profiling - Warum?[1]

- Viele Ursachen sind möglich, wenn ein Programm langsam läuft.
- Zeitaufwand minimieren durch Optimierung an der richtigen Stelle!
- Beim Profiling misst man den Ressourcenverbrauch eines Programms (Berechnungszeit, RAM).
- Faustregel: Erst profilen, dann optimieren.
  - Wie lange läuft das Programm?
  - Wie oft wird ein Funktion/Zeile ausgeführt?
  - Wieviel Speicher benötigt eine Datenstruktur?
- Optimierung an der falschen Stelle $\rightarrow$ großer Aufwand, kaum Verbesserungen

[1] Slides basieren auf Gorelick, Micha und Ozsvald, Ian: *High performance Python*. O'Reilly, 2014.

Profiling - Warum?[1]

- Viele Ursachen sind möglich, wenn ein Programm langsam läuft.
- Zeitaufwand minimieren durch Optimierung an der richtigen Stelle!
- Beim Profiling misst man den Ressourcenverbrauch eines Programms (Berechnungszeit, RAM).
- Faustregel: Erst profilen, dann optimieren.
  - Wie lange läuft das Programm?
  - Wie oft wird ein Funktion/Zeile ausgeführt?
  - Wieviel Speicher benötigt eine Datenstruktur?
- Optimierung an der falschen Stelle $\rightarrow$ großer Aufwand, kaum Verbesserungen

[1] Slides basieren auf Gorelick, Micha und Ozsvald, Ian: *High performance Python*. O'Reilly, 2014.

Profiling - Warum?[1]

- Viele Ursachen sind möglich, wenn ein Programm langsam läuft.
- Zeitaufwand minimieren durch Optimierung an der richtigen Stelle!
- Beim Profiling misst man den Ressourcenverbrauch eines Programms (Berechnungszeit, RAM).
- Faustregel: Erst profilen, dann optimieren.
  - Wie lange läuft das Programm?
  - Wie oft wird ein Funktion/Zeile ausgeführt?
  - Wieviel Speicher benötigt eine Datenstruktur?
- Optimierung an der falschen Stelle $\rightarrow$ großer Aufwand, kaum Verbesserungen

---

[1] Slides basieren auf Gorelick, Micha und Ozsvald, Ian: *High performance Python*. O'Reilly, 2014.

Modul `timeit`

http://docs.python.org/library/timeit.html

- Modul, um kleine Pythoncodestücke zu timen.
- Handhabung im Interpreter:

```
>>> import timeit
>>> setup = "l = range(10000)" # parameters
>>> stmt = "len(l)" # statement to evaluate
>>> timeit.timeit(stmt=stmt, setup=setup, number=10000)
0.0019829273223876953
```

- `stmt`: Ausdruck, dessen benötigte Zeit gemessen werden soll
- `setup`: Anweisungen, die (einmal) vor der Ausführung von `stmt` durchgeführt werden sollen
- `number`: Anzahl der Ausführungen von `stmt` (Default 1000000)

Rückgabe: Benötigte Zeit in Sekunden

Modul cProfile

https://docs.python.org/2/library/profile.html

- `timeit` ist sehr umständlich, wenn man Funktionen oder Programme evaluieren will.
- `cProfile` ist ein in der Standardbibliothek eingebautes Werkzeug, das misst, wie oft und wie lange Teile des Programms ausgeführt wurden.
- Warnung: Profiling verlängert die Laufzeit
- Profiler auf ein Skript anwenden:

```
$ python -m cProfile myProgram.py
```

Beispiel: Tiny Search Engine

Idee

- Implementierung einer Mini-Suchmaschine
- Die Suchanfrage (*query*) und die Dokumente (*documents*) sind Worthäufigkeitsvektoren.
- Ähnlichkeit zwischen Suchanfrage und Dokument: Kosinus

Tiny Search Engine: Pseudodaten

Anstatt echte Dokumente einzulesen, werden für Suchanfrage und Dokumente zufällig “Termhäufigkeiten” zwischen 0 und 100 gezogen:

```
    v = []
    for i in range(l):
        freq = random.randint(0, 100)
        v.append(freq)
    return v
```

Tiny Search Engine: Kosinusähnlichkeit

$$\cos(a, b) = \frac{a \cdot b}{\|a\|\|b\|} = \frac{a \cdot b}{\sqrt{\sum_i a_i^2}\sqrt{\sum_i b_i^2}}$$

Einfache Python-Implementierung

```
def cosine(a, b):
    assert(len(a)==len(b))
    ab_sum = 0
    a_sq_sum = 0
    b_sq_sum = 0
    for i in range(len(a)):
        ab_sum += a[i] * b[i]
        a_sq_sum += a[i] * a[i]
        b_sq_sum += b[i] * b[i]
    total = ab_sum / (math.sqrt(a_sq_sum) * math.sqrt(b_sq_sum))
    return total
```

Tiny Search Engine: run

```
    random.seed()
    query = pseudo_data(vocab_size)
    maxsim = 0
    best = []
    for i in range(ndocs):
        doc = pseudo_data(vocab_size)
        cos = cs.cosine(query, doc)
        if cos > maxsim:
            maxsim = cos
            best = doc

    return (maxsim, best)
```

cProfile Aufruf

```
$ python -m cProfile -s cumtime \
tiny_search_engine.py --vocab-size 10000 \
--num-docs 1000
```

`-s`: Sortierung der Ausgabe (hier nach kumulativer Laufzeit)

Ausgegebene Statistiken (in Spalten)

ncalls	Anzahl der Aufrufe
tottime	Gesamtlaufzeit dieser Funktion (ohne Subfunktionen)
percall	tottime / ncalls
cumtime	Gesamtlaufzeit dieser Funktion (mit Subfunktionen)
percall	cumtime / ncalls

cProfile Aufruf

```
$ python -m cProfile -s cumtime \
tiny_search_engine.py --vocab-size 10000 \
--num-docs 1000
```

`-s`: Sortierung der Ausgabe (hier nach kumulativer Laufzeit)

Ausgegebene Statistiken (in Spalten)

ncalls	Anzahl der Aufrufe
tottime	Gesamtlaufzeit dieser Funktion (ohne Subfunktionen)
percall	tottime / ncalls
cumtime	Gesamtlaufzeit dieser Funktion (mit Subfunktionen)
percall	cumtime / ncalls

cProfile Ausgabe

```
         40092169 function calls (40091230 primitive calls) in 19.384 seconds

   Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000   19.385   19.385 tiny_search_engine.py:3(<module>)
        1    0.000    0.000   19.282   19.282 tiny_search_engine.py:109(main)
        1    0.016    0.016   19.281   19.281 tiny_search_engine.py:44(run)
     1001    3.987    0.004   17.195    0.017 tiny_search_engine.py:33(pseudo_data)
 10010000    3.300    0.000   12.412    0.000 random.py:238(randint)
 10010000    8.321    0.000    9.111    0.000 random.py:175(randrange)
     1000    2.012    0.002    2.070    0.002 cosinus.py:8(cosine)
 10010000    0.791    0.000    0.791    0.000 {method 'random' of '_random.Random' objects}
 10015034    0.743    0.000    0.743    0.000 {method 'append' of 'list' objects}
     2113    0.111    0.000    0.111    0.000 {range}
```

...

Visualisierung der Profiler-Ausabe

1. Profiler-Ausgabe in Datei schreiben:

```
$ python -m cProfile -o profile.stats
tiny_search_engine.py -v 10000 -d 1000
```

2. gprof2dot: Python-Skript, um Profiler-Ausgabe in dot-Format zu übertragen.[2]

```
$ gprof2dot profile.stats -f pstats > profile.dot
```

3. dot: Tool zum Zeichnen von gerichteten Graphen

```
$ dot profile.dot -Tpng -o profile.png
```

Andere Pakete für Visualisierung: z.B. `snakeviz`, `pycallgraph`

---

[2] siehe `https://github.com/jrfonseca/gprof2dot`. Mehr zum dot-Format: `http://www.graphviz.org/doc/info/lang.html`

Visualisierung der Profiler-Ausabe

1. Profiler-Ausgabe in Datei schreiben:

```
$ python -m cProfile -o profile.stats
tiny_search_engine.py -v 10000 -d 1000
```

2. `gprof2dot`: Python-Skript, um Profiler-Ausgabe in dot-Format zu übertragen.[2]

```
$ gprof2dot profile.stats -f pstats > profile.dot
```

3. `dot`: Tool zum Zeichnen von gerichteten Graphen

```
$ dot profile.dot -Tpng -o profile.png
```

Andere Pakete für Visualisierung: z.B. `snakeviz`, `pycallgraph`

[2] siehe `https://github.com/jrfonseca/gprof2dot`. Mehr zum dot-Format: `http://www.graphviz.org/doc/info/lang.html`

Visualisierung der Profiler-Ausabe

1. Profiler-Ausgabe in Datei schreiben:

```
$ python -m cProfile -o profile.stats
tiny_search_engine.py -v 10000 -d 1000
```

2. `gprof2dot`: Python-Skript, um Profiler-Ausgabe in dot-Format zu übertragen.[2]

```
$ gprof2dot profile.stats -f pstats > profile.dot
```

3. `dot`: Tool zum Zeichnen von gerichteten Graphen

```
$ dot profile.dot -Tpng -o profile.png
```

Andere Pakete für Visualisierung: z.B. `snakeviz`, `pycallgraph`

[2] siehe `https://github.com/jrfonseca/gprof2dot`. Mehr zum dot-Format: `http://www.graphviz.org/doc/info/lang.html`

Visualisierung der Profiler-Ausabe

1. Profiler-Ausgabe in Datei schreiben:

```
$ python -m cProfile -o profile.stats
tiny_search_engine.py -v 10000 -d 1000
```

2. `gprof2dot`: Python-Skript, um Profiler-Ausgabe in dot-Format zu übertragen.[2]

```
$ gprof2dot profile.stats -f pstats > profile.dot
```

3. `dot`: Tool zum Zeichnen von gerichteten Graphen

```
$ dot profile.dot -Tpng -o profile.png
```

Andere Pakete für Visualisierung: z.B. `snakeviz`, `pycallgraph`

---

[2] siehe `https://github.com/jrfonseca/gprof2dot`. Mehr zum dot-Format: `http://www.graphviz.org/doc/info/lang.html`

Callgraph

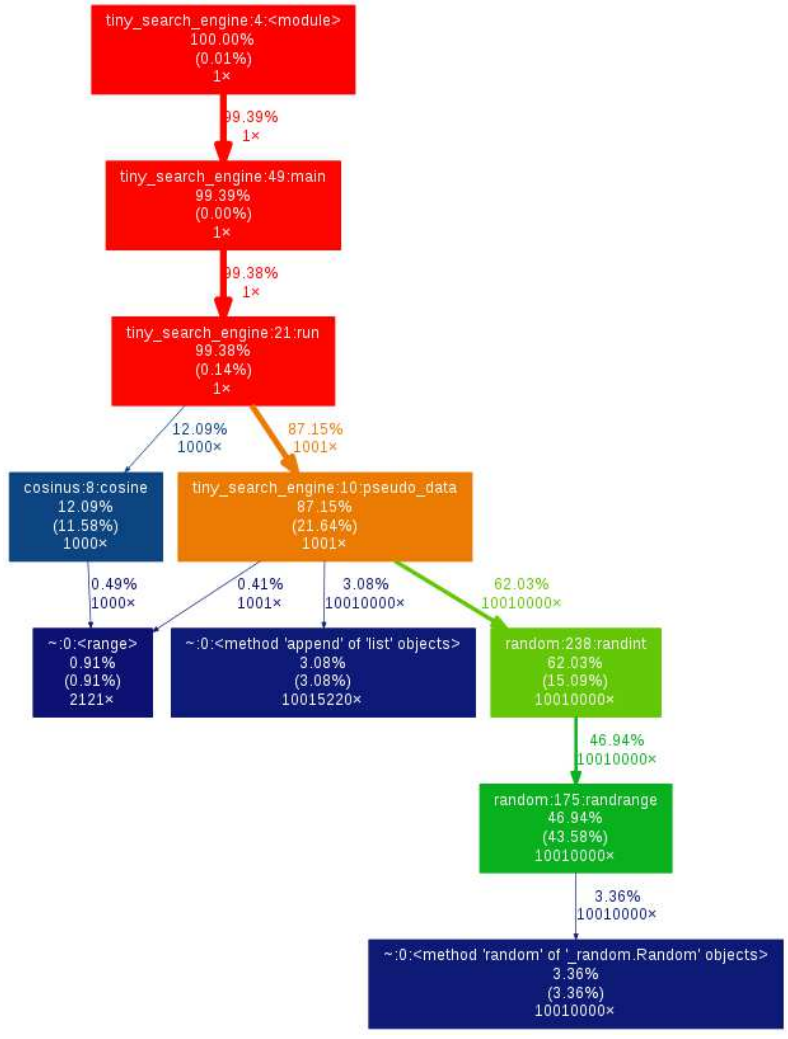


Abbildung: Callgraph für `tiny_search_engine.py`

Zeilenweises Profiling

- `cProfile` gibt Anzahl der Funktionsaufrufe und Laufzeit jeder Funktion an.
- Keine Auskunft darüber, welche Zeile wieviel Zeit benötigt.
- Lösung: `line_profiler` ermöglicht zeilenweises Profil.
- Skript zum Aufruf von `line_profiler`: `kernprof.py`
- Autor: Robert Kern, URL: `https://github.com/rkern/line_profiler`
- Installation: `sudo pip install line_profiler`

Zeilenweises Profiling

- `cProfile` gibt Anzahl der Funktionsaufrufe und Laufzeit jeder Funktion an.
- Keine Auskunft darüber, welche Zeile wieviel Zeit benötigt.
- Lösung: `line_profiler` ermöglicht zeilenweises Profil.
- Skript zum Aufruf von `line_profiler`: `kernprof.py`
- Autor: Robert Kern, URL: `https://github.com/rkern/line_profiler`
- Installation: `sudo pip install line_profiler`

Zeilenweises Profiling

- `cProfile` gibt Anzahl der Funktionsaufrufe und Laufzeit jeder Funktion an.
- Keine Auskunft darüber, welche Zeile wieviel Zeit benötigt.
- Lösung: `line_profiler` ermöglicht zeilenweises Profil.
- Skript zum Aufruf von `line_profiler`: `kernprof.py`
- Autor: Robert Kern, URL: `https://github.com/rkern/line_profiler`
- Installation: `sudo pip install line_profiler`

Zeilenweises Profiling

- `cProfile` gibt Anzahl der Funktionsaufrufe und Laufzeit jeder Funktion an.
- Keine Auskunft darüber, welche Zeile wieviel Zeit benötigt.
- Lösung: `line_profiler` ermöglicht zeilenweises Profil.
- Skript zum Aufruf von `line_profiler`: `kernprof.py`
- Autor: Robert Kern, URL: `https://github.com/rkern/line_profiler`
- Installation: `sudo pip install line_profiler`

Zeilenweises Profiling

- `cProfile` gibt Anzahl der Funktionsaufrufe und Laufzeit jeder Funktion an.
- Keine Auskunft darüber, welche Zeile wieviel Zeit benötigt.
- Lösung: `line_profiler` ermöglicht zeilenweises Profil.
- Skript zum Aufruf von `line_profiler`: `kernprof.py`
- Autor: Robert Kern, URL: `https://github.com/rkern/line_profiler`
- Installation: `sudo pip install line_profiler`

Zeilenweises Profiling

- `cProfile` gibt Anzahl der Funktionsaufrufe und Laufzeit jeder Funktion an.
- Keine Auskunft darüber, welche Zeile wieviel Zeit benötigt.
- Lösung: `line_profiler` ermöglicht zeilenweises Profil.
- Skript zum Aufruf von `line_profiler`: `kernprof.py`
- Autor: Robert Kern, URL: `https://github.com/rkern/line_profiler`
- Installation: **`sudo pip install line_profiler`**

Aufruf

Auf der Kommandozeilen:

```
$ kernprof -l -v tiny_search_engine.py
```

`-l` Aktiviert den Line Profiler.

`-v` Ausgabe wird angezeigt.

Default Ausgabe wird in (Binär-)datei `<Scriptname>.py.lprof` geschrieben.

Line Profiler anwenden

- Um den `line_profiler` auf eine Funktion anzuwenden, müssen wir ihm das erst mitteilen
- Dazu "dekoriert" man die Funktion mit `@profile`. Beispiel:

```
@profile
def pseudo_data(l):
    v = []
    for i in range(l):
        v.append(random.randint(0,100))
    return v
```

Decorators

- Ein Decorator ist ein aufrufbares Python-Objekt, das auf eine Funktion, Methode oder Klassendefinition angewendet wird.
- Der Decorator wird angewendet, indem man `@decoratorname` vor die Definition schreibt.
- In diesem Fall wird also der Decorator `profile` auf zu profilierende Funktionen angewendet.
- Das so veränderte Skript lässt sich nicht mehr normal ausführen!

line_profiler Ausgabe

Profiler Ausgabe für die Funktion pseudo_data

```
$ kernprof -l -v tiny_search_engine.py -v 1000 -d 1000
```

```
Timer unit: 1e-06 s

Total time: 3.52772 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( l):
    34      1001          456      0.5      0.0      v = []
    35   1002001       316844      0.3      9.0      for i in range(l):
    36   1001000      3210057      3.2     91.0          v.append(random.randint(0,100))
    37      1001          368      0.4      0.0      return v
```

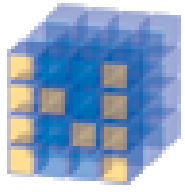
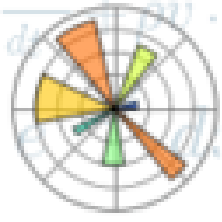
Übung 13

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

8 Python für Machine Learning
- Profiling
- Numpy und SciPy
- Die Numpy Package
- Die SciPy-Library
- scikit-learn

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.random_integers(1, 10, 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

Python für Machine Learning

Python für Machine Learning

Was?

- *SciKits* („SciPy Toolkits") sind Add-ons für SciPy
- Für uns interessant: `scikit-learn` - Toolkit für maschinelles Lernen, basierend auf SciPy
- Ausführliche Dokumentation: `http://scikit-learn.org/stable/user_guide.html`

Warum?

- Klare, gut dokumentierte API
- Schnell (verwendet NumPy Strukturen, kompilierten Code, Schnittstellen zu C-Bibliotheken für bestimmte Lernverfahren)
- Robust
- Relativ einfach zu bedienen

Python für Machine Learning

Einige Aufgabentypen I

Klassifikation

Elemente (*items*) sollen anhand ihrer Merkmale (*features*) bestimmten Klassen (*classes*, *labels*) zugeordnet werden. Es ist im Voraus bekannt, welche (und wie viele) Klassen es gibt.

Clustering

Elemente sollen anhand ihrer Merkmale gruppiert werden, so dass ähnliche Dinge in derselben Gruppe sind.

Einige Aufgabentypen II

Structured Prediction

Komplexen Strukturen (z.B. Sätzen) sollen andere komplexe Strukturen (z.B. Parse-Bäume) zugeordnet werden.

Regression

Für reellwertige Eingaben (z.B. Größe einer Wohnung in Quadratmetern) sollen reellwertige Vorhersagen gemacht werden (z.B. potentieller Kaufpreis).

Laufendes Beispiel: Klassifikation

Was sind die Elemente?

- Elemente sind die zu klassifierenden Objekte.
- Im Prinzip alles möglich.
- Wichtig: Nützliche und generalisierbare Merkmale für die Elemente extrahieren!

Was sind die Elemente?

- Elemente sind die zu klassifierenden Objekte.
- Im Prinzip alles möglich.
- Wichtig: Nützliche und generalisierbare Merkmale für die Elemente extrahieren!

Was sind die Elemente?

- Elemente sind die zu klassifierenden Objekte.
- Im Prinzip alles möglich.
- Wichtig: Nützliche und generalisierbare Merkmale für die Elemente extrahieren!

Was sind die Klassen?

- Elemente werden Klassen zugeordnet.
- Am Ende soll jedes Element zu genau einer Klasse gehören[4]
- Einfachste Klassen: “ja” und “nein” (binäre Klassifikation)

[4] Ausnahme: *multi-label classification*

Was sind die Klassen?

- Elemente werden Klassen zugeordnet.
- Am Ende soll jedes Element zu genau einer Klasse gehören[4]
- Einfachste Klassen: “ja” und “nein” (binäre Klassifikation)

[4] Ausnahme: *multi-label classification*

Was sind die Klassen?

- Elemente werden Klassen zugeordnet.
- Am Ende soll jedes Element zu genau einer Klasse gehören[4]
- Einfachste Klassen: “ja” und “nein” (binäre Klassifikation)

[4] Ausnahme: *multi-label classification*

Elemente und Klassen

Beispiele

Problem	Elemente	Klassen
POS-Tagging	Wörter	POS-Tags
Disambiguierung	Wörter im Kontext	Synsets
Anaphernresolution	Anapher/Antezedenten-Paare	Ja/Nein

Workflow

Gegeben *Trainingsdaten*, also mit Klassen annotierten Elementen, führen wir folgende Arbeitsschritte durch:

- *Feature-Extraktion*: Für jedes Element extrahieren wir Merkmalsausprägungen (*feature values*) aus den Daten
- *Training*: Ein Algorithmus "lernt" anhand der extrahierten Merkmale der annotierten Daten und der annotierten Klassen ein Modell.
- *Testing/Validierung*: Das trainierte Modell wird auf ungesehene, annotierte Beispiele angewendet, um die Performance des Modells zu testen.
- Dann kann das Modell oder die Feature-Extraktion eventuell verbessert werden.

Workflow

Gegeben *Trainingsdaten*, also mit Klassen annotierten Elementen, führen wir folgende Arbeitsschritte durch:

- *Feature-Extraktion*: Für jedes Element extrahieren wir Merkmalsausprägungen (*feature values*) aus den Daten
- *Training*: Ein Algorithmus "lernt" anhand der extrahierten Merkmale der annotierten Daten und der annotierten Klassen ein Modell.
- *Testing/Validierung*: Das trainierte Modell wird auf ungesehene, annotierte Beispiele angewendet, um die Performance des Modells zu testen.
- Dann kann das Modell oder die Feature-Extraktion eventuell verbessert werden.

Workflow

Gegeben *Trainingsdaten*, also mit Klassen annotierten Elementen, führen wir folgende Arbeitsschritte durch:

- *Feature-Extraktion*: Für jedes Element extrahieren wir Merkmalsausprägungen (*feature values*) aus den Daten
- *Training*: Ein Algorithmus "lernt" anhand der extrahierten Merkmale der annotierten Daten und der annotierten Klassen ein Modell.
- *Testing/Validierung*: Das trainierte Modell wird auf ungesehene, annotierte Beispiele angewendet, um die Performance des Modells zu testen.
- Dann kann das Modell oder die Feature-Extraktion eventuell verbessert werden.

Workflow

Gegeben *Trainingsdaten*, also mit Klassen annotierten Elementen, führen wir folgende Arbeitsschritte durch:

- *Feature-Extraktion*: Für jedes Element extrahieren wir Merkmalsausprägungen (*feature values*) aus den Daten
- *Training*: Ein Algorithmus "lernt" anhand der extrahierten Merkmale der annotierten Daten und der annotierten Klassen ein Modell.
- *Testing/Validierung*: Das trainierte Modell wird auf ungesehene, annotierte Beispiele angewendet, um die Performance des Modells zu testen.
- Dann kann das Modell oder die Feature-Extraktion eventuell verbessert werden.

Workflow

Gegeben *Trainingsdaten*, also mit Klassen annotierten Elementen, führen wir folgende Arbeitsschritte durch:

- *Feature-Extraktion*: Für jedes Element extrahieren wir Merkmalsausprägungen (*feature values*) aus den Daten
- *Training*: Ein Algorithmus "lernt" anhand der extrahierten Merkmale der annotierten Daten und der annotierten Klassen ein Modell.
- *Testing/Validierung*: Das trainierte Modell wird auf ungesehene, annotierte Beispiele angewendet, um die Performance des Modells zu testen.
- Dann kann das Modell oder die Feature-Extraktion eventuell verbessert werden.

Feature-Extraktion

Wichtiger Teil der Arbeit:

- Entscheiden, welche Merkmale verwendet werden sollen
- Code schreiben, der die Merkmale aus den Daten extrahiert

Trainings- und Testdaten

- Für faire und nützliche Ergebnisse darf nicht auf denselben Daten trainiert und getestet werden!
- Prozentuale Aufteilung: 30% der Daten werden zufällig ausgewählt und als Testdaten beiseite gelassen (während des Trainings)
- Cross Validation: Mehrere Trainingsläufe, wobei jeweils ein anderer Teil der Daten als Testset ausgelassen wird. Besonders sinnvoll bei wenig Daten.

Python für Machine Learning

scikit-learn

scikit-learn bietet...

- Überwachte Lernalgorithmen
- Unüberwachte Lernalgorithmen
- Module zur Vektorisierung und Filterung von Daten
- Module zur Validierung und Evaluierung

Verwendung

```
>>> import sklearn
>>> from sklearn import ...
>>> help(sklearn) # Überblick über Pakete
```

Wichtigste Funktionen von Lernalgorithmen

`fit(X[,y])` Lerne von Daten

`predict(X)` Wende Modell auf (Test-)daten an

`transform(X)` Transformiere Daten (z.B. Feature Selection)

Die Funktionen können auch kombiniert werden (z.B. `fit_predict()` oder `fit_transform()`).

Überwachte Methoden I

- Gegeben Beobachtungen X und Klassen Y
- Lerne $f : X \to Y$ so dass eine Zielfunktion minimiert (oder maximiert) wird (`fit()`-Methode)
- Bestimme $y = f(x)$ für ungesehene x (`predict()`-Methode)

`http://scikit-learn.org/stable/supervised_learning.html`

Überwachte Methoden II

Zwei Kategorien

Klassifikation Y besteht aus endlich vielen Elementen (z.B. $Y = \{0, 1\}$)
`sklearn.linear_model.LogisticRegression`,
`sklearn.tree.DecisionTreeClassifier`, ...

Regression Y ist stetig (z.B. $Y = \mathbb{R}$ oder $Y = [0, 1]$)
`sklearn.linear_model.LinearRegression`,
`sklearn.svm.SVR`, ...

Überwachte Methoden: Beispiel

```
>>> from sklearn import linear_model
>>> import numpy
>>> model = linear_model.LogisticRegression()
# erstelle Beispieldaten
>>> X = numpy.array([[0, 1, 1],
...      [1, 0, 1]])
>>> y = [1, 0]
>>> model.fit(X, y)
>>> model.predict(X)
[1, 0]
```

Unüberwachte Methoden I

- Hier sind nur Beobachtungen X gegeben
- Diese sollen z.B. gruppiert werden

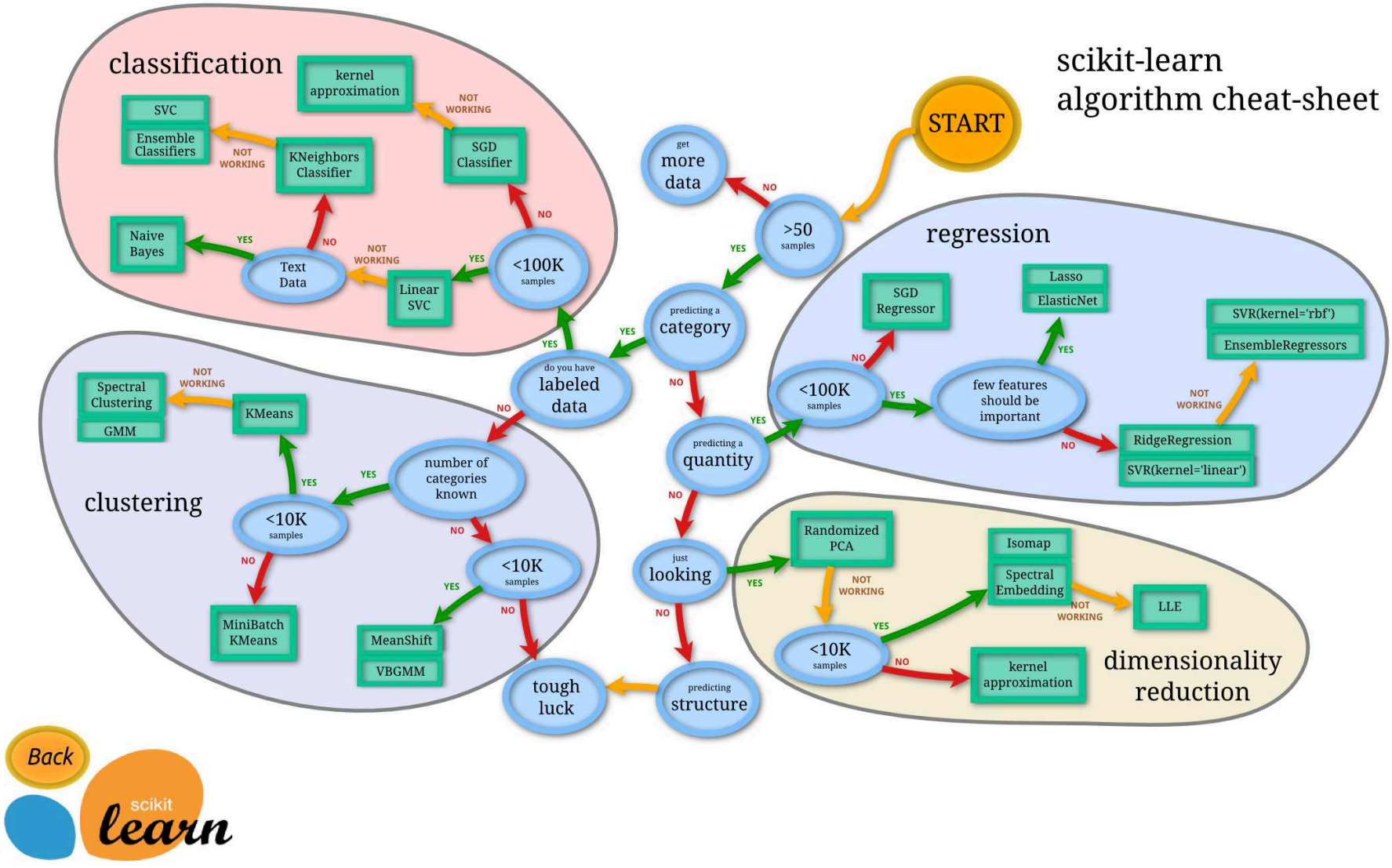
Unüberwachte Methoden II

Einige unüberwachte Lernverfahren

Clustering Daten werden in *Cluster* gruppiert
`sklearn.cluster`

Matrixzerlegung Oft: „Wichtigste" Dimensionen einer Beobachtungsmatrix finden
`sklearn.decomposition`

Welches Lernverfahren wann?



Datenvorbereitung

Preprocessing Normalisierung, Skalierung, kategoriale Features
`sklearn.preprocessing`

Feature-Extraktion Features in korrektes Format für Learner bringen, einfache Text- und Bildfeatures extrahieren
`sklearn.feature_extraction`

`http://scikit-learn.org/stable/modules/preprocessing.html`
`http://scikit-learn.org/stable/modules/feature_extraction.html`

Datenvorbereitung: Beispiel

- `feature_extraction.text` enthält Tools für Umgang mit Textdaten

```
>>> from sklearn.feature_extraction  import text
>>> countvec = text.CountVectorizer()
>>> text = ['Erster Satz', 'Zweiter Satz']
# 'lernt' das Vokabular
>>> countvec.fit(text)
>>> countvec.get_feature_names()
[u'erster', u'satz', u'zweiter']
# 'lernt' das Vokabular und
# generiert Dokument-Term-Matrix
>>> vectors = countvec.fit_transform(text)
# (sparse) Dokument-Term-Matrix
>>> print vectors
  (0, 0)    1
  (0, 1)    1
  (1, 1)    1
  (1, 2)    1
```

Evaluierung

Cross Validation Train/Test Split, Cross Validation (`sklearn.cross_validation`)

Grid Search Hyperparameter einstellen (`sklearn.grid_search`)

Evaluationsmetriken Precision, Recall, F1-Score (`sklearn.metrics`)

`http://scikit-learn.org/stable/model_selection.html`

Evaluierung: Beispiel

```
>>> predicted =  model.predict(X)
>>> sklearn.metrics.accuracy_score(y, predicted)
1.0
```

Modell speichern

- trainiertes Modell (z.B. SVM-Objekt) für spätere Anwendung speichern
- Lösung: Serialisierung mit `pickle` (Python-Modul zur Objekt-Serialisierung)

Übung 15

Mittagspause

Donnerstag: Weka

9 Weka
- Weka

Weka

Intro

Weka ist ...

- Eine Sammlung von Algorithmen für Machine Learning und Data Mining
- In Java implementiert
- Hat eine GUI und eine API
- Lizensiert unter GNU GPL
- `http://www.cs.waikato.ac.nz/ml/weka/`

Datenformate I

CSV (Comma-Separated Values

- Ein Beispiel pro Zeile
- Merkmale werden durch Komma getrennt

Example

```
Darth, upper, ""
Vader, upper, Darth
war, lower, Vader
ein, lower, war
Lord, upper, ein
der, lower, Lord
Sith, upper, der
...
```

Datenformate II

ARFF - *Attribute Relation File Format*

Standardformat in Weka

Example

```
@RELATION darth-vader
@ATTRIBUTE token STRING
@ATTRIBUTE case {upper,lower}
@ATTRIBUTE previous STRING
@ATTRIBUTE class {name, other}
@DATA
"Darth", upper, "", name
"Vader", upper, "Darth", name
"war", lower, "Vader", other
"ein", lower, "war", other
...
```

Datenformate III

Syntax von ARFF

- `@RELATION name`
  definiert einen Namen für das Datenset
- `@ATTRIBUTE attribute TYPE`
  definiert ein Attribut namens "attribute" vom Typ TYPE

string	Zeichenketten
numeric, real, integer	Zahlen
{ nom1, nom2 }	Listen nominaler Werte
date	Datumsangaben (yyyy-MM-dd'T'HH:mm:ss)

- `@DATA`
  Hier stehen die einzelnen Elemente (in CSV-Format)

Datenformate IV

Beispiel nominaler Werte

- { red, green, blue }
- { gabi, paula, anna-katharina }
- { one, two, three }
- { true, false }

Konvertierung

Sind alle Zeichenketten in einem Datenset bekannt, können sie automatisch in nominale Werte konvertiert werden.

Datenformate V

Annotation, fehlende Werte, Sonderzeichen

- Klassen werden im Attribut *class* angegeben, normalerweise als letztes Attribut
- Fehlende Werte werden mit einem `?` gekennzeichnet.
- Kommentare beginnen mit '%'
- Sonderzeichen (z.B. '?',',', '%') müssen in Anführungszeichen stehen, wenn sie nicht in ihrer Sonderbedeutung vorkommen.

Weka Benutzeroberfläche

Weka GUI Chooser

- ausführen mit

```
:~$ java -jar /path/to/weka.jar
```

- **Explorer**: Daten importieren, bearbeiten und visualisieren
- **Experimenter**: Experimente mit unterschiedlichen Parametern durchführen
- **KnowledgeFlow**: Komponenten und Datenströme graphisch modellieren
- **Simple CLI**: Command Line Interface

CLI

- Jeder Klassifizierer ist in einer Java-Klasse implementiert
- Aufruf über die Kommandozeile möglich:[5]

```
:~$ java weka.classifiers.trees.J48 <parameter>
```

- Parameter: Manche Parameter werden von jedem Klassifizierer verwendet (zum Beispiel Angabe der Trainings- und Testdaten), manche Parameter sind spezifisch für bestimmte Klassifizierer
- Wird der Klassifizierer ohne Argumente gestartet, zeigt der help screen alle Parameter an

[5] J48: Weka-Implementierung eines Entscheidungsbaums

Weitere Informationen zu Weka

- Witten, Frank & Hall (2011): Data Mining. Morgan Kaufman.
  $\rightarrow$ UB
- Online-Kurse: `https://weka.waikato.ac`

Übung 16