

# Ricerca in documenti di testo

Informatica@SEFA 2018/2019 - Lezione 20

Massimo Lauria <massimo.lauria@uniroma1.it>  
<http://massimolauria.net/courses/infosefa2018/>

Venerdì, 30 Novembre 2018

# Piano della lezione: piccolo motore di ricerca

Date

- una **collezione** di documenti
- una lista di parole di **ricerca**

ordinare i documenti per **rilevanza**

# Operazioni su stringhe

# Elaborazione di stringhe

Effettuare ricerche in testi è necessario

- normalizzare i dati
- effettuare analisi su di essi

questi passi richiedono l'elaborazione di stringhe di testo.

# Maiuscolo e minuscolo

- `upper()` tutte le lettere in maiuscolo
- `lower()` tutte le lettere in minuscolo
- `capitalize()` prima lettera maiuscola

```
s = 'Buona giornata. Il Numero 12345 ha vinto la lotteria!'    1
print(s.lower())                                              2
print(s.upper())                                              3
print(s.capitalize())                                         4
```

```
buona giornata. il numero 12345 ha vinto la lotteria!
BUONA GIORNATA. IL NUMERO 12345 HA VINTO LA LOTTERIA!
Buona giornata. il numero 12345 ha vinto la lotteria!
```

# Quante volte appare una sottostringa?

```
a = "Ma che bella giornata." 1
print(a.count('e'))           2
print(a.count('z'))           3
print(a.count('giornata'))    4
```

2  
0  
1

```
s = 'Giorno dopo giorno.' 1
print(s.count('giorno'))    2
s = s.lower()                # testo normalizzato 3
print(s.count('giorno'))    4
```

1  
2

# Conteggio su intervalli

`count` ha due argomenti opzionali che permettono di effettuare il conteggio su un intervallo della stringa.

```
a = 'aiaiaiaiai' 1
print(a.count('i', 3)) 2
print(a.count('i', 3, 9)) # fino alla posizione 9 esclusa 3
4
5
```

```
4
3
```

# Esempio: conteggio delle vocali

```
def conta_vocali(s): 1
    '''Conta le vocali non accentate in s.''' 2
    # Per contare anche le vocali maiuscole 3
    s = s.lower() 4
    count = 0 5
    for v in 'aeiou': 6
        count += s.count(v) 7
    return count 8

print(conta_vocali("che bello andare a spasso")) 9
print(conta_vocali("Nn c sn vcl")) 10
print(conta_vocali("Nn c sn vcl")) 11
```

9  
0



# Sovrapposizioni e count

Purtroppo `count` non permette di contare le occorrenze che si sovrappongono.

La stringa `'abababa'` contiene tre volte la stringa `'aba'`, ma contiene solo due occorrenze non sovrapposte.

```
testo="abababa"  
print(testo.count('aba'))
```

1  
2

2

# Find: cerca una sottostringa

`stringa.find(sottostringa)` restituisce

- la posizione della prima occorrenza di sottostringa
- -1 se sottostringa non c'è.

```
s = 'che bello andare a spasso'      1
print(s.find('bello'))                2

# cerca a partire dalla posizione 4   3
print(s.find('e', 4))                 4

# cerca tra le posizioni 10 e 20 esclusa 5
print(s.find('bello', 10, 20))        6
                                     7
                                     8
```

```
4
5
-1
```

# Contare le occorrenze con sovrapposizione

```
def conta_occorrenze(testo,pattern):           1
    start=0                                    2
    end  =len(testo)-len(pattern)+1            3
    count=0                                    4
    while start < end:                          5
        start = testo.find(pattern,start)+1    6
        if start==0:                            7
            break                               8
        count = count + 1                      9
    return count                               10
                                              11
print(conta_occorrenze('ababa abaaba','aba')) 12
print(conta_occorrenze('acaciacacia acacia aca','acacia')) 13
print(conta_occorrenze('alabama','a'))         14
```

4  
3  
4

# Scomposizione del testo

Per fare analisi di testi può aver senso dividere in righe, parole, ecc...

`splitlines` taglia dove ci sono gli `'\n'`

```
# Scomposizione in righe      1
testo = '''Prima linea      2
seconda linea                3

e quarta linea.'''          4
print(testo.splitlines())    5
print(testo.splitlines(True)) # mantiene i '\n'      6
                                7
```

```
['Prima linea', 'seconda linea', '', 'e quarta linea.']
['Prima linea\n', 'seconda linea\n', '\n', 'e quarta linea.']
```

# Scomposizione per parole split

```
s = "Una frase d'esempio, non \n troppo lunga"      1
# il separatore di default è un gruppo di spazi massimale 2
print('Es1. ', s.split())                             3
# altri separatori                                     4
print('Es2. ', s.split(','))                           5
print('Es3. ', s.split('p'))                           6
print('Es4. ', s.split('pp'))                           7
# se usiamo ' ' come separatore                        8
print('Es5. ', s.split(' '))                           9
```

```
Es1.  ['Una', 'frase', "d'esempio,", 'non', 'troppo', 'lunga']
Es2.  ["Una frase d'esempio", ' non \n troppo lunga']
Es3.  ["Una frase d'esem", 'io, non \n tro', '', 'o lunga']
Es4.  ["Una frase d'esempio, non \n tro", 'o lunga']
Es5.  ['Una', 'frase', "d'esempio,", 'non', '', '\n', '', 'troppo', 'lunga']
```

# Nozione di “whitespace”

**Whitespace:** una sequenza non vuota di ' ', '\n', '\t'.

In certe applicazioni di elaborazione di testi ci interessa il testo effettivo e non il modo in cui la separazione tra parole viene rappresentata. Ad esempio se applichiamo `split()` a

```
queste  
sono  
quattro parole
```

```
queste sono quattro  
parole
```

otteniamo sempre

```
['queste', 'sono', 'quattro', 'parole'].
```

# Pulizia degli elementi del testo

Spesso durante la suddivisione di testi, o la lettura di un input, gli elementi testuali hanno degli spazi spuri prima e dopo il testo utile.

`strip` elimina lo spazio bianco (whitespace) prima e dopo il testo effettivo

```
s = ' \n spazio prima e dopo \n '      1
print(repr(s))                          2
print(repr(s.strip()))                  3
```

```
' \n spazio prima \n e dopo \n '
'spazio prima \n e dopo'
```

# Sostituzione di testo

Otteniamo una nuova stringa a partire dalla vecchia, sostituendo dei pattern di testo.

```
s = "Ciao Bruno come stai?" 1
2
print(s.replace('Bruno', 'Sara')) 3
print(s.replace('bruno', 'sara')) 4
print(s.replace('come ', '').replace('?', ' bene?')) 5
6
t = "Odio l'estate, amo l'estate." 7
print(t.replace("l'estate", "l'inverno")) 8
print(t.replace("l'estate", "la primavera")) 9
```

```
Ciao Sara come stai?
Ciao Bruno come stai?
Ciao Bruno stai bene?
Odio l'inverno, amo l'inverno.
Odio la primavera, amo la primavera.
```



# Metodo format per inserire dati nel testo

```
base='{ } per { } uguale { }'      # sostituiti in ordine      1
                                     2
print(base.format(5,3,5*3))         3
print(base.format(8,7,8*7))         4
                                     5
# Usare i nomi dei parametri esplicitamente      6
print('{nome} nato in {indirizzo} nel {anno}'.format(      7
    nome='Mario', anno='1974',indirizzo='Italia'))      8
                                     9
# usare i numeri dei parametri      10
print('{0} nato in {2} nel {1}, si nel {1}'.format(      11
    'Mario','1974','Italia'))      12
```

```
5 per 3 uguale 15
8 per 7 uguale 56
Mario nato in Italia nel 1974
Mario nato in Italia nel 1974, si nel 1974
```

# Elaborazione del testo

# Esempi di analisi di testi

Abbiamo abbastanza nozioni e strumenti per vedere qualche tipo di elaborazione su testi. Lo scopo è vedere esempi di

- elaborazione di stringhe
- estrazione di dati
- analisi di file

# Esempio: costruzione di una rubrica

Trasformiamo una rappresentazione testuale di una rubrica telefonica in un dizionario python. Ogni riga contiene al massimo una coppia `nome:numero`.

Vogliamo qualcosa che produca un output come questo

```
rubrica = '''                                1
Marco: 5551234                                2
Luisa: 5557653                                3

Sara: 5558723'''                              4
                                              5
dizionario = rubrica_to_dict(rubrica)         6
print(dizionario)                             7
                                              8
```

```
{'marco': '5551234', 'luisa': '5557653', 'sara': '5558723'}
```

## Esempio: costruzione di una rubrica (2)

```
def rubrica_to_dict(elenco):           1
    '''Converte un elenco da testo a tabella.''' 2
    d = {}                               3
    elenco = elenco.lower().splitlines()         4
    for e in elenco:                       5
        tokens = e.split(':')              6
        if len(tokens) != 2:               7
            # le righe mal formattate
            continue                       8
            # vengono ignorate
        nome, numero = e.split(':')         9
        d[nome.strip()] = numero.strip()    10
    return d                               11
```

# Tutte le parole in una testo

Per il nostro piccolo motore di ricerca dovremo identificare tutte le parole **distinte** nel documento. Potremmo usare `split`, ma c'è un problema:

```
s="Abbiamo un cane, un gatto, e un altro cane." 1
2
print("cane"      in s.split())    # la parola 'cane' compare? 3
print("gatto"     in s.split())    # la parola 'gatto' compare? 4
print("abbiamo"   in s.split())    # la parola 'abbiamo' compare?5
```

```
False
False
False
```

La punteggiatura, le maiuscole, ecc... confondono.

# “Pulitura” del testo

I dati prima di essere analizzati devono essere **puliti** o **preparati**. In questo caso trasformiamo

- i caratteri non alfabetici in spazi
- il testo in minuscolo con `lower()`

così `split()` ci darà la lista delle parole nel testo.

```
s="abbiamo un cane un gatto e un altro cane " 1
2
print("cane" in s.split()) # la parola 'cane' compare? 3
print("gatto" in s.split()) # la parola 'gatto' compare? 4
print("abbiamo" in s.split()) # la parola 'abbiamo' compare?5
```

```
True
True
True
```

# Raccogliamo i caratteri non alfabetici

`stringa.isalpha()` è `True` quando la stringa è fatta da caratteri alfabetici. Qui la usiamo su singoli caratteri.

```
def noalpha(testo):  
    '''Ritorna una stringa contenente tutti i  
    caratteri non alfabetici contenuti in testo,  
    senza ripetizioni'''  
    noa = ''  
    for c in testo:  
        if not c.isalpha() and c not in noa:  
            noa += c  
    return noa  
  
print(noalpha("Frase con numeri 0987"))  
print(noalpha("Abbiamo un cane, un gatto, e un altro cane."))  
print(noalpha("Frase con simboli vari [],{} %&#@"))
```

0987

,.  
[],{}%&#@



# La lista di tutte le parole in una stringa

```
def words(s):                                     1
    '''Ritorna la lista delle parole contenute    2
    nella stringa s'''                            3
    noa = noalpha(s)                              4
    for c in noa:                                5
        s = s.replace(c, ' ') # eliminiamo i caratteri  # non alfabetici 6
    return s.lower().split() # eliminiamo le maiuscole 7
print(words("Che bel tempo, usciamo!"))          8
                                                    9
                                                    10
```

```
['che', 'bel', 'tempo', 'usciamo']
```

# La lista di tutte le parole in un file

```
def fwords(fname,encoding):          1
    with open(fname, encoding=encoding) as f:      2
        testo = f.read()                        3
    return words(testo)                      4
                                           5
parole = fwords('alice.txt','utf-8-sig')          6
print(len(parole))                                7
print(parole[897:903])                            8
```

30423

['so', 'alice', 'soon', 'began', 'talking', 'again']

# Analisi del testo

# Motore di ricerca giocattolo

Abbiamo in input:

- una serie di documenti
- delle parole di ricerca

Vogliamo in output:

- la lista ordinata dei documenti, dal più rilevante al meno rilevante.

# Strategia del motore di ricerca

1. Di ogni parola che vogliamo cercare calcoliamo la **frequenza relativa** nel documento, ovvero

$$\text{frequenza} = \# \text{occorrenze} / \# \text{parole totali}$$

2. La **rilevanza** di un documento rispetto alle parole di ricerca è la somma delle frequenze relative di quelle parole.
3. Ordiniamo i documenti rispetto alla rilevanza degli stessi rispetto alla ricerca.

# Calcoliamo le frequenze relative

- un file
- la lista delle parole
- encoding del file

```
def wfreq(fname, ricerca, enc):  
    # ottiene la lista delle parole  
    parole = fwords(fname, enc)  
    # prepare il dizionario delle frequenze  
    frequenze = {}  
  
    for parola in ricerca:  
        occ = parole.count(parola.lower())  
        freq = occ*100/len(parole) # in percentuale  
        frequenze[parola] = round(freq,3) # tre decimali  
    return frequenze  
  
ricerca = ['alice','rabbit','turtle','queen']  
freq = wfreq('alice.txt', ricerca, 'utf-8-sig')  
print(freq)
```

```
{'alice': 1.325, 'rabbit': 0.168, 'turtle': 0.194, 'queen': 0.247}
```

# Rilevanza dei documenti

- una **lista di nomi di file**
- la lista delle parole
- encoding del file

```
def scores(fnames, ricerca, enc):           1
    punteggio = {}                          2
    for fname in fnames:                   3
        # dizionario delle frequenze di fname  4
        f = wfreq(fname, ricerca, enc)      5
        # score arrotondata                 6
        punteggio[fname] = round(sum(f.values()), 3) 7
    return punteggio                        8
```

# Rilevanza dei documenti (2)

```
documenti = [ 'alice.txt', 'holmes.txt',           1
               'frankenstein.txt', 'prince.txt',    2
               'mobydick.txt', 'treasure.txt' ]      3
                                                    4
ricerca = [ 'monster', 'horror', 'night' ]          5
                                                    6
from pprint import pprint    # print a bit better   7
                                                    8
pprint(scores(documenti, ricerca, 'utf-8-sig'))     9
```

```
{'alice.txt': 0.016,
 'frankenstein.txt': 0.216,
 'holmes.txt': 0.119,
 'mobydick.txt': 0.095,
 'prince.txt': 0.023,
 'treasure.txt': 0.066}
```



# Mettiamo tutto insieme

```
def extract_value(kv): return kv[1] 1
2
3
def searchdocument(fnames, ricerca, enc): 4
    '''Ritorna la lista ordinata per score dei 5
    documenti in fnames per le parole in ricerca.''' 6
    s = scores(fnames, ricerca, enc) 7
    return sorted(s.items(), 8
                  key=extract_value, 9
                  reverse=True) 10
```

# Esempi di ricerca (1)

```
documenti = [ 'alice.txt', 'holmes.txt',           1
               'frankenstein.txt', 'prince.txt',    2
               'mobydick.txt', 'treasure.txt' ]      3
                                                    4
ricerca = [ 'monster', 'horror', 'night' ]          5
                                                    6
                                                    7
pprint(searchdocument(documenti,                   8
                       ricerca,                     9
                       'utf-8-sig'))                10
```

```
[('frankenstein.txt', 0.216),
 ('holmes.txt', 0.119),
 ('mobydick.txt', 0.095),
 ('treasure.txt', 0.066),
 ('prince.txt', 0.023),
 ('alice.txt', 0.016)]
```

## Esempi di ricerca (2)

```
documenti = [ 'alice.txt', 'holmes.txt',           1
               'frankenstein.txt', 'prince.txt',    2
               'mobydick.txt', 'treasure.txt' ]      3
                                                    4
ricerca = [ 'ship', 'pirate', 'sea' ]               5
                                                    6
                                                    7
pprint(searchdocument(documenti,                    8
                       ricerca,                      9
                       'utf-8-sig'))                 10
```

```
[('mobydick.txt', 0.441),
 ('treasure.txt', 0.323),
 ('frankenstein.txt', 0.053),
 ('alice.txt', 0.046),
 ('holmes.txt', 0.014),
 ('prince.txt', 0.013)]
```

# Lecture

Capitolo 11 del libro di Python.