# Analysis of unstructured data

## Lecture 7 - Text analysis (basics)

### Janusz Szwabiński

Outlook:

- Text mining
- Case study - cycling aroud the world
- Text document representation
- Text preparation
- Frequency matrix
- Zipf law
- Stemming and lemmatization
- Frequency matrix revisited

In [1]:

```
%matplotlib inline
import matplotlib.pyplot as plt
```

# Text mining

Text mining (or *text data mining*, TDM) is the process of deriving high-quality information from text.

Text Mining
=
Data Mining (applied to text data)
+
basic linguistics

It usually involves:

- the process of structuring the input text (parsing, along with the addition of some derived linguistic features and the removal of others, and subsequent insertion into a database)
- deriving patterns within the structured data
- evaluation and interpretation of the output

Goals:

- document classification
- document grouping
- document summarization (automatic generation of summaries)
- automatic language recognition
- concept/entity extraction
- visualization and navigation

Applications - mainly statistical processing of:

- journal articles
- emails
- open questions in surveys
- medical records
- trading sessions' comments
- motivation letters and CVs
- consumer complaints

Difficulties:

- representation of abstract concepts
- "countless" combinations of subtle, abstract relations between concepts
- many ways to represent similar concept (e.g. UFO, spaceship, flying saucer)
- visualization of concepts
- high dimensional data
- thousands of features that have to be described

Any good news?

- highly redundant data
- even the simplest algorithm helps to extract something interesting from text:
  - pulling out "important" phrases
  - finding "meaningfully" related words
  - creating some sort of summaries from documents

# Case study - cycling route around the world

Let us assume that our goal is to draw Tom Davies' cycling route around the world (see https://tomdaviesrtw.com/blog/ (https://tomdaviesrtw.com/blog/) for details).

Having a look at the html code of the blog, we find that the starting and destination points of each stage are cointained within the `h1` tag of class `entry-title`. We have to scrap them:

In [2]:

```
import requests
```

In [3]:

```
from bs4 import BeautifulSoup

route = []

for i in range(1,20):
    r = requests.get('https://tomdaviesrtw.com/blog/page/%s'%(i))
    soup = BeautifulSoup(r.content,"lxml")

    hits = soup.findAll("h1", { "class" : "entry-title" })
    for h in hits:
        route.append(h.text.strip())
```

In [4]:

```
route
```

Out[4]:

```
['An Update – 2 Years\xa0Later',
 'Day 174 – Winchester to London – Final\xa0Day!',
 'Day 173 – Poole to\xa0Winchester',
 'Day 172 – Fougères to\xa0Cherbourg',
 'Day 171 – Cholet to\xa0Fougères',
 'Day 170 – Pons to\xa0Cholet',
 'Day 169 – Mimizan to\xa0Pons',
 'Day 168 – Zumaia to\xa0Mimizan',
 'Day 167 – Briviesca to\xa0Zumaia',
 'Day 166 – Valladolid to\xa0Briviesca',
 'Day 165 – Ciudad Rodrigo to\xa0Valladolid',
 'Day 164 – Nisa to Cuidad\xa0Rodrigo',
 'Day 163 – Lisbon to Nisa – Start of Leg\xa08',
 'Leg 7 – San Francisco to Boston –\xa0Summary',
 'Day 162 – Leominster to Boston – End of Leg\xa07!',
 'Day 161 – Bennington to\xa0Leominster',
 'Day 160 – Utica to\xa0Bennington',
 'Day 159 – Canandaigua to\xa0Utica',
 'Day 158 – Niagara Falls to\xa0Canandaigua',
 'Day 157 – Orangeville to\xa0Niagara',
 'Day 156 – Bracebride to\xa0Orangeville',
 'Day 155 – Jamot to\xa0Bracebridge',
 'Day 154 – Espanola to\xa0Jamot',
 'Day 153 – Bruce Mines to\xa0Espanola',
 'Day 152 – Montreal River to Bruce\xa0Mines',
 'Day 151 – White River to Montreal River\xa0Harbor',
 'Day 150 – Schreiber to White\xa0River',
 'Day 149 – Thunder Bay to\xa0Schreiber',
 'Day 148 – Grand Marais to Thunder\xa0Bay',
 'Day 147 – Two Harbors to Grand\xa0Marais',
 'Day 146 – Grand Rapids to Two\xa0Harbors*',
 'Day 145 – Bemidji to Grand\xa0Rapids',
 'Day 144 – Grand Forks to\xa0Bemidji',
 'Day 143 – Devils Lake to Grand\xa0Forks',
 'Day 142 – Minot to Devils\xa0Lake',
 'Day 141 – Williston to\xa0Minot',
 'Day 140 – Glendive to\xa0Williston',
 'Day 139 – Forsyth to\xa0Glendive',
 'Day 138 – Roundup to\xa0Forsyth',
 'Day 137 – White Sulphur Springs to\xa0Roundup',
 'Day 136 – Helena to White Sulphur\xa0Springs',
 'Day 135 – Missoula to\xa0Helena',
 'Day 134 – Thompson Falls to\xa0Missoula',
 'Day 133 – Priest River to Thompson\xa0Falls',
 'Day 132 – Colfax to Priest\xa0River',
 'Day 131 – Othello to\xa0Colfax',
 'Day 130 – Rimrock Lake to\xa0Othello',
 'Day 129 – Castle Rock to Rimrock\xa0Lake',
 'Day 128 – Newberg to Castle\xa0Rock',
 'Day 127 – Newport to\xa0Newberg',
 'Day 126 – Coos Bay to\xa0Newport',
 'Day 125 – Brookings to Coos\xa0Bay',
 'Day 124 – Arcata to\xa0Brookings',
 'Day 123 – The Middle of Nowhere to\xa0Arcata',
 'Day 122 – Gualala to a random campsite somewhere on the Redwood Hi
ghway between Riverdale and\xa0Pierce',
 'Day 121 RTW – San Francisco (Tiburon) to Gualala – Start of Leg\xa
07',
 'Leg 6 – New Zealand\xa0Summary…',
 'Day 120 – Miranda to Auckland – End of Leg\xa06'
```

```
    'Day 120 – Miranda to Auckland – End of Leg\xa06',
    'Day 119 – Te Kuiti to\xa0Miranda',
    'Day 118 – New Plymouth to Te\xa0Kuiti',
    'Day 117 – Wanganui to New\xa0Plymouth',
    'Day 116 – Wellington to\xa0Wanganui',
    'Day 115 – Kaikoura to\xa0Picton',
    'Day 114 – Christchurch to\xa0Kaikoura',
    'Day 113 – Timaru to\xa0Christchurch',
    'Day 112 – Dunedin to Timaru – Start of Leg\xa06',
    'Leg 5 – Australia\xa0Summary',
    'Day 111 – Aratula to\xa0Brisbane',
    'Day 110 – Tenterfield to\xa0Aratula',
    'Day 109 – Armidale to\xa0Tenterfield',
    'Day 108 – Quirindi to\xa0Armidale',
    'Day 107 – Cessnock to\xa0Quirindi',
    'Day 106 – Sydney to\xa0Cessnock',
    'Day 105 – Wollongong to\xa0Sydney',
    'Day 104 – Batemans Bay to\xa0Wollongong',
    'Day 103 – Tathra to Batemans\xa0Bay',
    'Day 102 – Cann River to\xa0Tathra',
    'Day 101 – Bairnsdale to Cann\xa0River',
    'Day 100 – Mirboo North to\xa0Bairnsdale',
    'Day 99 – Queenscliff to Mirboo\xa0North',
    'Day 98 – Lavers Hill to\xa0Queenscliff',
    'Day 97 – Portland to Lavers\xa0Hill',
    'Day 96 – Penola to\xa0Portland',
    'Day 95 – Keith to\xa0Penola',
    'Day 94 – Murray Bridge to\xa0Keith',
    'Day 93 – Adelaide to Murray\xa0Bridge',
    'Day 92 – Clare to\xa0Adelaide',
    'Day 91 – Port Augusta to\xa0Clare',
    'Day 90 – Cowell to Port\xa0Augusta',
    'Day 89 – Port Lincoln to\xa0Cowell',
    'Day 88 – Port Kenny to Port Lincoln….Halfway',
    'Day 87 – Ceduna to Port\xa0Kenny',
    'Day 86 – Nundroo to\xa0Ceduna',
    'Day 85 – Nullabor to\xa0Nundroo',
    'Day 84 – Eucla to\xa0Nullabor',
    'Day 83 – Madura to\xa0Eucla',
    'Day 82 – Caiguna to\xa0Madura',
    'Day 81 – Balladonia to\xa0Caiguna',
    'Day 80 – Norseman to\xa0Balladonia',
    'Day 79 – Esperance to\xa0Norseman',
    'Day 78 – Ravensthorpe to\xa0Esperance',
    'Day 77 – Boxwood Hill to\xa0Ravensthorpe',
    'Day 76 – Denmark to Boxwood\xa0Hill',
    'Day 75 – Pemberton to\xa0Denmark',
    'Day 74 – Bunbury to\xa0Pemberton',
    'Day 73 – Perth to\xa0Bunbury',
    'A Summary of Leg 3 & 4 – Mandalay to\xa0Singapore',
    'Day 72 – Mersing to Singapore – End of Leg\xa04',
    'DAY 71 – Pekan to\xa0Mersing',
    'Day 70 – Dungun to\xa0Pekan',
    'Day 69 – Bukit Keluang to\xa0Dungun',
    'Day 68 – Belum Rainforest to Bukit\xa0Keluang',
    'Day 67 – Alor Setar to Belum\xa0Rainforest',
    'Day 66 – Tamot (Thailand) to Alor Setar\xa0(Malaysia)',
    'Day 65 – Somewhere in Thailand to\xa0Tamot',
    'Day 64 – Lang Suan to somewhere slightly south of Khao\xa0Niphan',
    'Day 63 – Bang Saphan to Lang\xa0Suan',
    '',
    'Day 62 – Hua Hin to Bang\xa0Saphan',
```

```
    'Day 61 – Don Tum to Hua-Hin – End of Leg\xa03',
    'Day 60 – Ban Doem Bang to Don\xa0Tum',
    'Day 59 – Kamphaeng Phet to Ban Doem\xa0Bang',
    'Day 58 – Mae Sot to Kamphaeng\xa0Phet',
    'Day 57 – Hpa-Ann to Mae\xa0Sot',
    'Day 56 – Kyaikto to\xa0Hpa-An',
    'Day 55 – Pyinpongyi (Moyingyi Lake) to\xa0Kyaikto',
    'Day 54 – Phyu to Pyinpongyi (Moyingyi\xa0Lake)',
    'Day 53 – Naypyitaw to\xa0Phyu',
    'Day 52 – Meiktila to\xa0Naypyitaw',
    'Mandalay and Day 51 – Mandalay to\xa0Meiktila',
    'Mumbai to Kanyakumari to Kolkata – A summary of\xa0sorts',
    'Day 50 – Balasore to Panskura – End of Leg\xa02',
    'Day 49 – Cuttack to\xa0Balasore',
    'Day 48 – Chatrapur to\xa0Cuttack',
    'Day 47 – Srikakulam to\xa0Chatrapur',
    'DAY 46 – TUNI TO\xa0SRIKAKULAM',
    'Day 45 – Bhimavaram to\xa0Tuni',
    'Day 44 – Chirala to\xa0Bhimavaram',
    'Day 43 – Nellore to\xa0Chirala',
    'Day 42 – Chennai to\xa0Nellore',
    'Day 41 – Puducherry to\xa0Chennai',
    'Day 40 Velankanni to\xa0Puducherry',
    'Day 39 – Ramanathapuram to\xa0Velankanni',
    'Day 38 Tiruchendur to\xa0Ramanathapuram',
    'Day 37 Marthandam to\xa0Tiruchendur',
    'Day 36 – Haripad to\xa0Marthandam',
    'Day 35 – Guruvayoor to\xa0Haripad',
    'Day 34 – Thalassery to\xa0Guruvayoor',
    'Day 33 – Mangalore to\xa0Thalassery',
    'Day 32 – Murudeshwar to\xa0Mangalore',
    'Day 31 – Benaulim to\xa0Murudeshwar',
    'Day 30 – Kankavali to\xa0Benaulim',
    'Day 29 -Hathkhamba to\xa0Kankavali',
    'Day 28 – Somewhere in India to somewhere else in\xa0India',
    'Mumbai and Day\xa027',
    'Goodbye Europe!',
    'A bit of a Summary – London to\xa0Istanbul',
    'Day 26 – Alexandroupoli to Tekirdag – End of leg\xa01',
    'Day 25 – Nea Peramos to\xa0Alexandroupoli',
    'Day 24 – Giannitsa to Nea\xa0Peramos',
    'Day 23 – Aetos to\xa0Giannitsa',
    'Day 22 – Kastoria to\xa0Aetos',
    'Day 21 – Pogradec to\xa0Kastoria',
    'Day 20 – Tirana to\xa0Pogradec',
    'Day 19 – Bar to\xa0Tirana',
    'Day 18 – Slano to\xa0Bar',
    'Day 17 – Omiš to\xa0Slano',
    'Day 16 – Sukošan to\xa0Omiš',
    'Day 15 – Senj to\xa0Sukošan',
    'Day 14 – Part 2 – Language\xa0Warning',
    'Day 14 – Monfalcone to\xa0Senj',
    'Day 13 – Correzzola to\xa0Monfalcone',
    'Day 12 – Cannot Sull’Oglio to\xa0Correzzola',
    'Day 11 – Alessandria to Canneto Sull’Oglio',
    'Day 10 – Diano Marina to\xa0Alessandria',
    '‘Rest Day’ – 1 &\xa02',
    'Day 9 – Fréjus to Diano\xa0Marina',
    'Day 8 – Salon-de-Provence to\xa0Fréjus',
    'Day 7 – Le Pouzin to Salon-de-Provence',
    'Day 6 – Sarcey to Le\xa0Pouzin',
```

```
 'Day 5  – Decize to\xa0Sarcey',
 'Day 4 – Sully-sur-Loire to\xa0Decize',
 'Day 3 – Manou to\xa0Sully-sur-Loire',
 'Day 2 – Caen to\xa0Manou',
 'Day 1 – London to\xa0Portsmouth',
 'Blog']
```

We want to do some data cleaning:

In [5]:

```
route2 = [i for i in route if i.startswith('Day')]
```

In [6]:

```
route2.reverse()
```

In [7]:

```
route2
```

Out[7]:

```
['Day 1 – London to\xa0Portsmouth',
 'Day 2 – Caen to\xa0Manou',
 'Day 3 – Manou to\xa0Sully-sur-Loire',
 'Day 4 – Sully-sur-Loire to\xa0Decize',
 'Day 5  – Decize to\xa0Sarcey',
 'Day 6 – Sarcey to Le\xa0Pouzin',
 'Day 7 – Le Pouzin to Salon-de-Provence',
 'Day 8 – Salon-de-Provence to\xa0Fréjus',
 'Day 9 – Fréjus to Diano\xa0Marina',
 'Day 10 – Diano Marina to\xa0Alessandria',
 'Day 11 – Alessandria to Canneto Sull’Oglio',
 'Day 12 – Cannot Sull’Oglio to\xa0Correzzola',
 'Day 13 – Correzzola to\xa0Monfalcone',
 'Day 14 – Monfalcone to\xa0Senj',
 'Day 14 – Part 2 – Language\xa0Warning',
 'Day 15 – Senj to\xa0Sukošan',
 'Day 16 – Sukošan to\xa0Omiš',
 'Day 17 – Omiš to\xa0Slano',
 'Day 18 – Slano to\xa0Bar',
 'Day 19 – Bar to\xa0Tirana',
 'Day 20 – Tirana to\xa0Pogradec',
 'Day 21 – Pogradec to\xa0Kastoria',
 'Day 22 – Kastoria to\xa0Aetos',
 'Day 23 – Aetos to\xa0Giannitsa',
 'Day 24 – Giannitsa to Nea\xa0Peramos',
 'Day 25 – Nea Peramos to\xa0Alexandroupoli',
 'Day 26 – Alexandroupoli to Tekirdag – End of leg\xa01',
 'Day 28 – Somewhere in India to somewhere else in\xa0India',
 'Day 29 -Hathkhamba to\xa0Kankavali',
 'Day 30 – Kankavali to\xa0Benaulim',
 'Day 31 – Benaulim to\xa0Murudeshwar',
 'Day 32 – Murudeshwar to\xa0Mangalore',
 'Day 33 – Mangalore to\xa0Thalassery',
 'Day 34 – Thalassery to\xa0Guruvayoor',
 'Day 35 – Guruvayoor to\xa0Haripad',
 'Day 36 – Haripad to\xa0Marthandam',
 'Day 37 Marthandam to\xa0Tiruchendur',
 'Day 38 Tiruchendur to\xa0Ramanathapuram',
 'Day 39 – Ramanathapuram to\xa0Velankanni',
 'Day 40 Velankanni to\xa0Puducherry',
 'Day 41 – Puducherry to\xa0Chennai',
 'Day 42 – Chennai to\xa0Nellore',
 'Day 43 – Nellore to\xa0Chirala',
 'Day 44 – Chirala to\xa0Bhimavaram',
 'Day 45 – Bhimavaram to\xa0Tuni',
 'Day 47 – Srikakulam to\xa0Chatrapur',
 'Day 48 – Chatrapur to\xa0Cuttack',
 'Day 49 – Cuttack to\xa0Balasore',
 'Day 50 – Balasore to Panskura – End of Leg\xa02',
 'Day 52 – Meiktila to\xa0Naypyitaw',
 'Day 53 – Naypyitaw to\xa0Phyu',
 'Day 54 – Phyu to Pyinpongyi (Moyingyi\xa0Lake)',
 'Day 55 – Pyinpongyi (Moyingyi Lake) to\xa0Kyaikto',
 'Day 56 – Kyaikto to\xa0Hpa-An',
 'Day 57 – Hpa-Ann to Mae\xa0Sot',
 'Day 58 – Mae Sot to Kamphaeng\xa0Phet',
 'Day 59 – Kamphaeng Phet to Ban Doem\xa0Bang',
 'Day 60 – Ban Doem Bang to Don\xa0Tum',
 'Day 61 – Don Tum to Hua-Hin – End of Leg\xa03',
 'Day 62 – Hua Hin to Bang\xa0Saphan',
```

```
 'Day 62 – Hua Hin to Bang\xa0Saphan',
 'Day 63 – Bang Saphan to Lang\xa0Suan',
 'Day 64 – Lang Suan to somewhere slightly south of Khao\xa0Niphan',
 'Day 65 – Somewhere in Thailand to\xa0Tamot',
 'Day 66 – Tamot (Thailand) to Alor Setar\xa0(Malaysia)',
 'Day 67 – Alor Setar to Belum\xa0Rainforest',
 'Day 68 – Belum Rainforest to Bukit\xa0Keluang',
 'Day 69 – Bukit Keluang to\xa0Dungun',
 'Day 70 – Dungun to\xa0Pekan',
 'Day 72 – Mersing to Singapore – End of Leg\xa04',
 'Day 73 – Perth to\xa0Bunbury',
 'Day 74 – Bunbury to\xa0Pemberton',
 'Day 75 – Pemberton to\xa0Denmark',
 'Day 76 – Denmark to Boxwood\xa0Hill',
 'Day 77 – Boxwood Hill to\xa0Ravensthorpe',
 'Day 78 – Ravensthorpe to\xa0Esperance',
 'Day 79 – Esperance to\xa0Norseman',
 'Day 80 – Norseman to\xa0Balladonia',
 'Day 81 – Balladonia to\xa0Caiguna',
 'Day 82 – Caiguna to\xa0Madura',
 'Day 83 – Madura to\xa0Eucla',
 'Day 84 – Eucla to\xa0Nullabor',
 'Day 85 – Nullabor to\xa0Nundroo',
 'Day 86 – Nundroo to\xa0Ceduna',
 'Day 87 – Ceduna to Port\xa0Kenny',
 'Day 88 – Port Kenny to Port Lincoln….Halfway',
 'Day 89 – Port Lincoln to\xa0Cowell',
 'Day 90 – Cowell to Port\xa0Augusta',
 'Day 91 – Port Augusta to\xa0Clare',
 'Day 92 – Clare to\xa0Adelaide',
 'Day 93 – Adelaide to Murray\xa0Bridge',
 'Day 94 – Murray Bridge to\xa0Keith',
 'Day 95 – Keith to\xa0Penola',
 'Day 96 – Penola to\xa0Portland',
 'Day 97 – Portland to Lavers\xa0Hill',
 'Day 98 – Lavers Hill to\xa0Queenscliff',
 'Day 99 – Queenscliff to Mirboo\xa0North',
 'Day 100 – Mirboo North to\xa0Bairnsdale',
 'Day 101 – Bairnsdale to Cann\xa0River',
 'Day 102 – Cann River to\xa0Tathra',
 'Day 103 – Tathra to Batemans\xa0Bay',
 'Day 104 – Batemans Bay to\xa0Wollongong',
 'Day 105 – Wollongong to\xa0Sydney',
 'Day 106 – Sydney to\xa0Cessnock',
 'Day 107 – Cessnock to\xa0Quirindi',
 'Day 108 – Quirindi to\xa0Armidale',
 'Day 109 – Armidale to\xa0Tenterfield',
 'Day 110 – Tenterfield to\xa0Aratula',
 'Day 111 – Aratula to\xa0Brisbane',
 'Day 112 – Dunedin to Timaru – Start of Leg\xa06',
 'Day 113 – Timaru to\xa0Christchurch',
 'Day 114 – Christchurch to\xa0Kaikoura',
 'Day 115 – Kaikoura to\xa0Picton',
 'Day 116 – Wellington to\xa0Wanganui',
 'Day 117 – Wanganui to New\xa0Plymouth',
 'Day 118 – New Plymouth to Te\xa0Kuiti',
 'Day 119 – Te Kuiti to\xa0Miranda',
 'Day 120 – Miranda to Auckland – End of Leg\xa06',
 'Day 121 RTW – San Francisco (Tiburon) to Gualala – Start of Leg\xa
07',
 'Day 122 – Gualala to a random campsite somewhere on the Redwood Hi
ghway between Riverdale and\xa0Pierce',
```

```
 'Day 123 – The Middle of Nowhere to\xa0Arcata',
 'Day 124 – Arcata to\xa0Brookings',
 'Day 125 – Brookings to Coos\xa0Bay',
 'Day 126 – Coos Bay to\xa0Newport',
 'Day 127 – Newport to\xa0Newberg',
 'Day 128 – Newberg to Castle\xa0Rock',
 'Day 129 – Castle Rock to Rimrock\xa0Lake',
 'Day 130 – Rimrock Lake to\xa0Othello',
 'Day 131 – Othello to\xa0Colfax',
 'Day 132 – Colfax to Priest\xa0River',
 'Day 133 – Priest River to Thompson\xa0Falls',
 'Day 134 – Thompson Falls to\xa0Missoula',
 'Day 135 – Missoula to\xa0Helena',
 'Day 136 – Helena to White Sulphur\xa0Springs',
 'Day 137 – White Sulphur Springs to\xa0Roundup',
 'Day 138 – Roundup to\xa0Forsyth',
 'Day 139 – Forsyth to\xa0Glendive',
 'Day 140 – Glendive to\xa0Williston',
 'Day 141 – Williston to\xa0Minot',
 'Day 142 – Minot to Devils\xa0Lake',
 'Day 143 – Devils Lake to Grand\xa0Forks',
 'Day 144 – Grand Forks to\xa0Bemidji',
 'Day 145 – Bemidji to Grand\xa0Rapids',
 'Day 146 – Grand Rapids to Two\xa0Harbors*',
 'Day 147 – Two Harbors to Grand\xa0Marais',
 'Day 148 – Grand Marais to Thunder\xa0Bay',
 'Day 149 – Thunder Bay to\xa0Schreiber',
 'Day 150 – Schreiber to White\xa0River',
 'Day 151 – White River to Montreal River\xa0Harbor',
 'Day 152 – Montreal River to Bruce\xa0Mines',
 'Day 153 – Bruce Mines to\xa0Espanola',
 'Day 154 – Espanola to\xa0Jamot',
 'Day 155 – Jamot to\xa0Bracebridge',
 'Day 156 – Bracebride to\xa0Orangeville',
 'Day 157 – Orangeville to\xa0Niagara',
 'Day 158 – Niagara Falls to\xa0Canandaigua',
 'Day 159 – Canandaigua to\xa0Utica',
 'Day 160 – Utica to\xa0Bennington',
 'Day 161 – Bennington to\xa0Leominster',
 'Day 162 – Leominster to Boston – End of Leg\xa07!',
 'Day 163 – Lisbon to Nisa – Start of Leg\xa08',
 'Day 164 – Nisa to Cuidad\xa0Rodrigo',
 'Day 165 – Ciudad Rodrigo to\xa0Valladolid',
 'Day 166 – Valladolid to\xa0Briviesca',
 'Day 167 – Briviesca to\xa0Zumaia',
 'Day 168 – Zumaia to\xa0Mimizan',
 'Day 169 – Mimizan to\xa0Pons',
 'Day 170 – Pons to\xa0Cholet',
 'Day 171 – Cholet to\xa0Fougères',
 'Day 172 – Fougères to\xa0Cherbourg',
 'Day 173 – Poole to\xa0Winchester',
 'Day 174 – Winchester to London – Final\xa0Day!']
```

We may use the package `geotext` to extract city mentions from our list:

In [8]:

```
from geotext import GeoText
```

In [9]:

```
cities = []
for i in route2:
    i.replace('\xa0'," ") #removing of the control sequence
    places = GeoText(i)
    for p in places.cities:
        if p not in cities:
            cities.append(p)
```

In [10]:

```
cities
```

Out[10]:

```
['London',
 'Portsmouth',
 'Caen',
 'Fréjus',
 'Marina',
 'Alessandria',
 'Monfalcone',
 'Bar',
 'Tirana',
 'Mangalore',
 'Thalassery',
 'Ramanathapuram',
 'Puducherry',
 'Chennai',
 'Nellore',
 'Tuni',
 'Chatrapur',
 'Cuttack',
 'Balasore',
 'Meiktila',
 'Kyaikto',
 'Hpa-An',
 'Mae Sot',
 'Kamphaeng Phet',
 'Hua Hin',
 'Bang Saphan',
 'Lang Suan',
 'Alor Setar',
 'Pekan',
 'Mersing',
 'Perth',
 'Bunbury',
 'Augusta',
 'Adelaide',
 'Murray',
 'Murray Bridge',
 'Portland',
 'Bay',
 'Wollongong',
 'Sydney',
 'Cessnock',
 'Armidale',
 'Brisbane',
 'Dunedin',
 'Timaru',
 'Christchurch',
 'Wellington',
 'Wanganui',
 'Plymouth',
 'New Plymouth',
 'Auckland',
 'San Francisco',
 'Riverdale',
 'Arcata',
 'Brookings',
 'Coos Bay',
 'Newport',
 'Newberg',
 'Castle Rock',
 'Missoula'
```

```
 'Missoula',
 'Helena',
 'Springs',
 'Minot',
 'Grand Forks',
 'Grand Rapids',
 'Thunder Bay',
 'White River',
 'Orangeville',
 'Niagara Falls',
 'Utica',
 'Leominster',
 'Boston',
 'Lisbon',
 'Valladolid',
 'Cholet',
 'Fougères',
 'Poole',
 'Winchester']
```

In order to find the geographical coordinates of every city we may use `geopy` - a Python client for several popular geocoding web services:

In [11]:

```
import time
from geopy.geocoders import Nominatim
geolocator = Nominatim()

positions = dict()

for c in cities:
    while True:
        try:
            position = geolocator.geocode(c)
        except:
            time.sleep(5)
            continue
        break
    if position:
        location = [position.latitude, position.longitude]
        positions.update({c : location})
        del position
    else:
        print("Could not get position for {}".format(c))

print(positions)
```

{'Castle Rock': [39.3724232, -104.8586706], 'Riverdale': [41.544477
8, -90.4581879], 'Grand Rapids': [42.9632405, -85.6678639], 'Bosto
n': [42.3604823, -71.0595678], 'Tuni': [34.9403711, -90.7089982], 'C
hatrapur': [19.3556141, 84.9796383], 'Tirana': [7.3021792, -75.07474
34], 'Murray': [36.6103334, -88.314761], 'Coos Bay': [43.3665007, -1
24.2178903], 'Mae Sot': [16.7133787, 98.5699283], 'Bunbury': [-33.32
67797, 115.636698], 'Arcata': [40.866517, -124.08284], 'Newberg': [4
5.300596, -122.9725418], 'Ramanathapuram': [9.5, 78.5], 'Fougères':
 [48.3536781, -1.2022211], 'Nellore': [14.4493717, 79.9873763], 'Spr
ings': [41.0162112, -72.1592444], 'Kamphaeng Phet': [16.4183581, 99.
6111616], 'Minot': [48.23251, -101.296273], 'Balasore': [21.5017098,
86.9216712], 'Chennai': [13.0801721, 80.2838331], 'Augusta': [33.470
9714, -81.9748429], 'Dunedin': [-45.8739282, 170.503488], 'Monfalcon
e': [45.8080945, 13.5310806], 'Mangalore': [15.9231076, 75.7622963],
'Fréjus': [43.4330308, 6.7360182], 'Lang Suan': [9.9499186, 99.07227
22], 'Wollongong': [-34.4243941, 150.89385], 'Thunder Bay': [48.4064
14, -89.259796], 'Missoula': [46.8700801, -113.9952796], 'Hua Hin':
 [49.94155235, 11.5757717283903], 'Cessnock': [-32.89403765, 151.301
463938301], 'Bang Saphan': [11.2132683, 99.5034992], 'Winchester':
 [51.0612766, -1.3131692], 'Leominster': [42.5250906, -71.759794],
 'Brookings': [44.311461, -96.79844], 'Cholet': [47.0617293, -0.8801
349], 'Lisbon': [46.441634, -97.68121], 'Utica': [43.1009031, -75.23
26641], 'Niagara Falls': [43.1030928, -79.0302618], 'Meiktila': [20.
8762431, 95.8601123], 'Helena': [46.592712, -112.036109], 'Timaru':
 [-44.3930254, 171.2509786], 'Grand Forks': [47.9078244, -97.059202
8], 'New Plymouth': [-39.0579941, 174.0806474], 'Thalassery': [11.74
90639, 75.4931578], 'London': [51.5073219, -0.1276474], 'Christchurc
h': [-43.530955, 172.6366455], 'Perth': [-31.9527121, 115.8604796],
 'Mersing': [2.4276183, 103.836363], 'Brisbane': [-27.4689682, 153.0
234991], 'Hpa-An': [16.8902634, 97.6362745], 'Puducherry': [11.93405
68, 79.8306447], 'Armidale': [-30.41250495, 151.940139839798], 'Peka
n': [3.4900731, 103.3936995], 'Murray Bridge': [-35.1199938, 139.278
3817], 'Bar': [42.0979745, 19.0954528], 'Newport': [51.5882332, -2.9
974967], 'Wellington': [-41.2887639, 174.7772239], 'Cuttack': [20.46
86, 85.8792], 'Orangeville': [43.9194098, -80.0988238], 'White Rive
r': [43.7953781, -72.5001951], 'Alor Setar': [6.1231908, 100.368396
2], 'Portland': [45.5202471, -122.6741949], 'Adelaide': [-34.927428
4, 138.599899], 'Plymouth': [50.3712659, -4.1425658], 'Portsmouth':
 [50.8036831, -1.075614], 'San Francisco': [8.444585, 125.9512491854
75], 'Caen': [49.1828008, -0.3690815], 'Bay': [14.1412627, 121.26969
422834], 'Auckland': [-36.8534665, 174.7655514], 'Kyaikto': [17.3091
636, 97.0096713], 'Marina': [36.6900785, -121.800639830262], 'Wangan
ui': [-39.9347494, 175.0539001], 'Poole': [50.74018055, -1.953773835
24348], 'Alessandria': [44.9129225, 8.615321], 'Sydney': [-33.854815
7, 151.2164539], 'Valladolid': [41.6521328, -4.728562]}

The last step is to visualize the route on a map. The `folium` package provides an easy way to put marks on a Leaflet.js map:

In [12]:

```
import folium
```

In [13]:

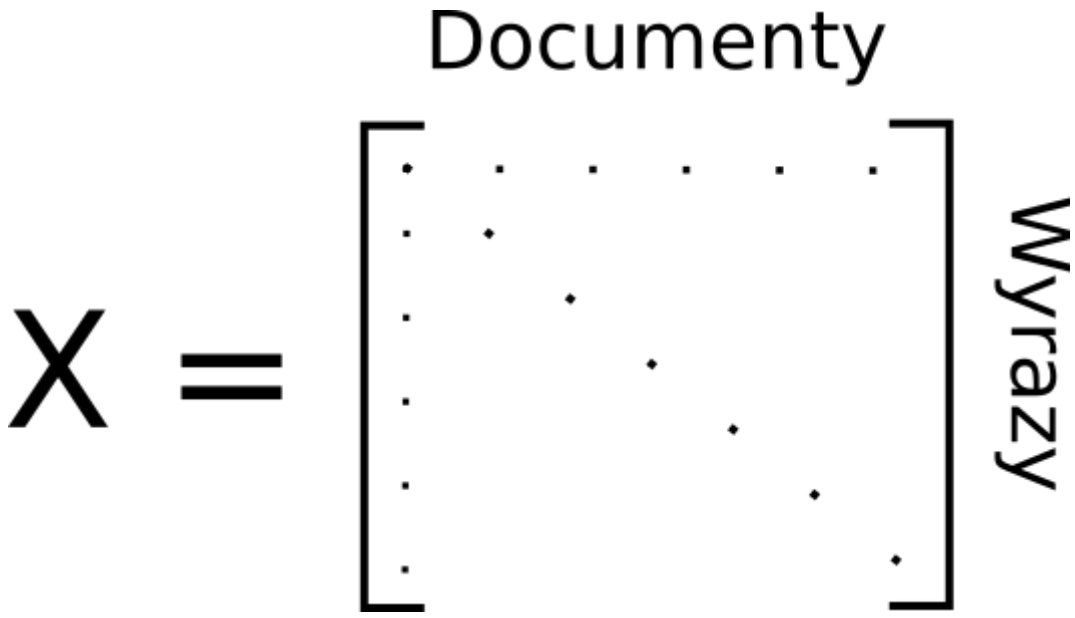
```
mapa = folium.Map(location=[0, 0], zoom_start=2)


for city, location in positions.items():

mapa.add_child(folium.Marker(location=location,popup=folium.Popup(city),icon=fol
ium.Icon(color='orange',icon='ok')))

mapa
```

Out[13]:

# Text documents' representation

## Unigram representation

- **bag-of-words**
- simple
- the most popular way of representing text documents
- a text is represented as the multiset (bag) of its words
- grammar and word order are ignored
- types:
  - binary (yes/no)
  - frequency




Element $x_{ij}$ of the above matrix is the number of occurrences of $i$-th word in $j$-th document.

## $n$-gram representation

- $n$-gram is a contiguous sequence of $n$ items from a given sequence of text
- items can be phonemes, syllables, letters, words or base pairs according to the application
- when the items are words, $n$-grams may also be called **shingles**
- allows to take into account co-occurrences of $n$ words (e.g. "flying saucer")
- usually $n \leq 3$ (bigrams and trigrams)
- does not perform well for inflected languages like for instance Polish
- very useful for analytic languages (like English)

*Twas brillig, and the slithy toves*
*Did gyre and gimble in the wabe*

| | twas | brillig | and | the | slithy | toves | did | gyre | gimble | in | wabe |
|---|---|---|---|---|---|---|---|---|---|---|---|
| twas | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| brillig | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| and | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| the | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| slithy | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| toves | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| did | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| gyre | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| gimble | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| in | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| wabe | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### $\gamma$-gram representation

- similar to $n$-gram, but allows for sequences of varying lengths

### Positional representation

- information about the position of words is stored
- a document is often divided into segments
- for every segment, a word-position matrix is generated
- important for 2 heuristics:
  - words distributed uniformly in the whole document have usually less meaning that those appearing only in one segment of the document
  - in similar documents, segments with multiple occurrences of same words are similar

## Text preparation

- transformation of documents to plain text (e.g. `pdftotext`)
- consistent character encoding (e.g. `iconv` under Unix/Linux, http://gzegzolka.com/ (http://gzegzolka.com/))
- consistent capitalization
- removing of control sequences
- removing of punctuation
- word extraction

### Examples

#### Convert pdf to plain text

In [3]:

```
! pdftotext 5939-efficient-and-parsimonious-agnostic-active-learning.pdf active_
learning.txt
```

In [26]:

```
! head -20 active_learning.txt
```

```
Efficient and Parsimonious Agnostic Active Learning

Tzu-Kuo Huang
Microsoft Research, NYC

Alekh Agarwal
Microsoft Research, NYC

Daniel Hsu
Columbia University

tkhuang@microsoft.com

alekha@microsoft.com

djhsu@cs.columbia.edu

John Langford
Microsoft Research, NYC
```

#### Change character encoding

In [32]:

```
! cat tekst_lat2.txt
```

```
���涼��
```

In [33]:

```
with open('tekst_lat2.txt','r') as f:
    print(f.read())
```

```
---------------------------------------------------------------------
-------
UnicodeDecodeError                        Traceback (most recent cal
l last)
<ipython-input-33-e7405cbd1ee8> in <module>()
      1 with open('tekst_lat2.txt','r') as f:
----> 2     print(f.read())

/usr/lib/python3.5/codecs.py in decode(self, input, final)
    319         # decode input (taking the buffer into account)
    320         data = self.buffer + input
--> 321         (result, consumed) = self._buffer_decode(data,
self.errors, final)
    322         # keep undecoded input until the next call
    323         self.buffer = data[consumed:]

UnicodeDecodeError: 'utf-8' codec can't decode byte 0xb1 in position
 0: invalid start byte
```

In [34]:

```
infile = open('tekst_lat2.txt', 'rb')
outfile = open('tekst_utf_8.txt', 'wb')
BLOCKSIZE = 65536 # experiment with optimal size for large files
while True:
    block = infile.read(BLOCKSIZE)
    if not block: break
    outfile.write(block.decode('latin2').encode('utf8'))
infile.close()
outfile.close()
```

In [35]:

```
with open('tekst_utf_8.txt','r') as f:
    print(f.read())
```

```
ąęóćśźż
```

The same with UNIX cli:

In [36]:

```
!iconv -f latin2 -t utf8 tekst_lat2.txt > new_file
```

In [37]:

```
!cat new_file
```

```
ąęóćśźż
```

#### Capitalization

In [38]:

```
text = '''Pruska Biblioteka Państwowa. Jej dawne zbiory znane pod nazwą
"Berlinka" to skarb kultury i sztuki niemieckiej. Przewiezione przez
Niemców pod koniec II wojny światowej na Dolny Śląsk, zostały
odnalezione po 1945 r. na terytorium Polski. Trafiły do Biblioteki
Jagiellońskiej w Krakowie, obejmują ponad 500 tys. zabytkowych
archiwaliów, m.in. manuskrypty Goethego, Mozarta, Beethovena, Bacha.'''
```

In [39]:

```
text.lower()
```

Out[39]:

```
'pruska biblioteka państwowa. jej dawne zbiory znane pod nazwą\n"ber
linka" to skarb kultury i sztuki niemieckiej. przewiezione przez\nni
emców pod koniec ii wojny światowej na dolny śląsk, zostały\nodnalez
ione po 1945 r. na terytorium polski. trafiły do biblioteki\njagiell
ońskiej w krakowie, obejmują ponad 500 tys. zabytkowych\narchiwalió
w, m.in. manuskrypty goethego, mozarta, beethovena, bacha.'
```

In [40]:

```
text.upper()
```

Out[40]:

```
'PRUSKA BIBLIOTEKA PAŃSTWOWA. JEJ DAWNE ZBIORY ZNANE POD NAZWĄ\n"BER
LINKA" TO SKARB KULTURY I SZTUKI NIEMIECKIEJ. PRZEWIEZIONE PRZEZ\nNI
EMCÓW POD KONIEC II WOJNY ŚWIATOWEJ NA DOLNY ŚLĄSK, ZOSTAŁY\nODNALEZ
IONE PO 1945 R. NA TERYTORIUM POLSKI. TRAFIŁY DO BIBLIOTEKI\nJAGIELL
OŃSKIEJ W KRAKOWIE, OBEJMUJĄ PONAD 500 TYS. ZABYTKOWYCH\nARCHIWALIÓ
W, M.IN. MANUSKRYPTY GOETHEGO, MOZARTA, BEETHOVENA, BACHA.'
```

#### Control sequences

- the names of all control sequences start with 'C'
- we can exploit that:

In [41]:

```
import unicodedata
def remove_control_characters(s):
    return "".join(ch for ch in s if unicodedata.category(ch)[0]!="C")
```

In [42]:

```
text = "This is an example of a text\n with the control sequence of a new line!"
print(text)
```

```
This is an example of a text
 with the control sequence of a new line!
```

In [43]:

```
print(remove_control_characters(text))
```

```
This is an example of a text with the control sequence of a new lin
e!
```

In [44]:

```
print(unicodedata.category('\n'))
```

```
Cc
```

#### Punctuation

In [45]:

```
import re, string, timeit

s = "string. With. Punctuation"
exclude = set(string.punctuation)
table = str.maketrans('','',string.punctuation)
regex = re.compile('[%s]' % re.escape(string.punctuation))

def test_set(s):
    return ''.join(ch for ch in s if ch not in exclude)

def test_re(s):
    return regex.sub('', s)

def test_trans(s):
    return s.translate(table)

def test_repl(s):  # From S.Lott's solution
    for c in string.punctuation:
        s=s.replace(c,"")
    return s

print("sets      :",timeit.Timer('f(s)', 'from __main__ import s,test_set as
f').timeit(1000000))
print("regex     :",timeit.Timer('f(s)', 'from __main__ import s,test_re as
f').timeit(1000000))
print("translate :",timeit.Timer('f(s)', 'from __main__ import s,test_trans as
 f').timeit(1000000))
print("replace   :",timeit.Timer('f(s)', 'from __main__ import s,test_repl as
f').timeit(1000000))
```

```
sets      : 2.7463853059743997
regex     : 1.145417162013473
translate : 1.1952509529946838
replace   : 5.275555466010701
```

**Good to know: `translate`**

- `maketrans()` method returns a translation table that maps each character in the string given as the first argument into the character at the same position in the second one. If there is a third argument, it must be a string, whose characters will be mapped to None in the result
- `translate()` method returns a copy of the string in which all characters have been translated using the translation table
- optionally, it can delete all characters found in the string given as the second argument

In [46]:

```
import string
intab = "aeiou"
outtab = "12345"
trantab = str.maketrans(intab,outtab)

text = 'this is a string example... wow!!!'
print(text.translate(trantab))
```

```
th3s 3s 1 str3ng 2x1mpl2... w4w!!!
```

As presented above, we can use these method to remove punctuation:

In [47]:

```
print(text.translate(table))
```

```
this is a string example wow
```

#### Punctuation - different approach

In [48]:

```
from nltk.tokenize import RegexpTokenizer

text = "She looked at   her father's arm-chair."

tokenizer = RegexpTokenizer(r'\w+')
' '.join(tokenizer.tokenize(text))
```

Out[48]:

```
'She looked at her father s arm chair'
```

#### Word extraction

A "naive" method:

In [49]:

```
text = "She looked at   her father's arm-chair."
```

In [50]:

```
text.split(' ')
```

Out[50]:

```
['She', 'looked', 'at', '', '', 'her', "father's", 'arm-chair.']
```

In [51]:

```
import string
table = str.maketrans('','',string.punctuation)
```

In [52]:

```
print([s.translate(table) for s in text.split(' ') if s != ''])
```

```
['She', 'looked', 'at', 'her', 'fathers', 'armchair']
```

With the module `scikit-learn`:

In [53]:

```
from sklearn.feature_extraction.text import CountVectorizer
CountVectorizer().build_tokenizer()(text)
```

Out[53]:

```
['She', 'looked', 'at', 'her', 'father', 'arm', 'chair']
```

With the `nltk` module:

In [37]:

```
from nltk.tokenize import word_tokenize
word_tokenize(text)
```

Out[37]:

```
['She', 'looked', 'at', 'her', 'father', "'s", 'arm-chair', '.']
```

In [54]:

```
from nltk.tokenize import RegexpTokenizer

tokenizer = RegexpTokenizer(r'\w+')
tokenizer.tokenize(text)
```

Out[54]:

```
['She', 'looked', 'at', 'her', 'father', 's', 'arm', 'chair']
```

In [55]:

```
text = '''Pruska Biblioteka Państwowa. Jej dawne zbiory znane pod nazwą
"Berlinka" to skarb kultury i sztuki niemieckiej. Przewiezione przez
Niemców pod koniec II wojny światowej na Dolny Śląsk, zostały
odnalezione po 1945 r. na terytorium Polski. Trafiły do Biblioteki
Jagiellońskiej w Krakowie, obejmują ponad 500 tys. zabytkowych
archiwaliów, m.in. manuskrypty Goethego, Mozarta, Beethovena, Bacha.'''

tokenizer.tokenize(text)
```

Out[55]:

```
['Pruska',
 'Biblioteka',
 'Państwowa',
 'Jej',
 'dawne',
 'zbiory',
 'znane',
 'pod',
 'nazwą',
 'Berlinka',
 'to',
 'skarb',
 'kultury',
 'i',
 'sztuki',
 'niemieckiej',
 'Przewiezione',
 'przez',
 'Niemców',
 'pod',
 'koniec',
 'II',
 'wojny',
 'światowej',
 'na',
 'Dolny',
 'Śląsk',
 'zostały',
 'odnalezione',
 'po',
 '1945',
 'r',
 'na',
 'terytorium',
 'Polski',
 'Trafiły',
 'do',
 'Biblioteki',
 'Jagiellońskiej',
 'w',
 'Krakowie',
 'obejmują',
 'ponad',
 '500',
 'tys',
 'zabytkowych',
 'archiwaliów',
 'm',
 'in',
 'manuskrypty',
 'Goethego',
 'Mozarta',
 'Beethovena',
 'Bacha']
```

# Frequency matrix (*bag of words*, BOW)

- one of the simplest representations of text
- very popular
- only the counts of words matter, no grammar or spatial information
- often used for classification purposes

## Example

Consider 2 documents:

(1) John likes to watch movies. Mary likes too.
(2) John also likes to watch football games.

We may use the `Counter` class from the `collections` module to build the bad of words:

In [56]:

```
import collections, re
```

In [57]:

```
docs = ['John likes to watch movies. Mary likes too.', 'John also likes to watch
 football games.']
bagsofwords = [collections.Counter(re.findall(r'\w+',txt)) for txt in docs]
```

In [58]:

```
print(bagsofwords[0])
```

```
Counter({'likes': 2, 'Mary': 1, 'John': 1, 'movies': 1, 'to': 1, 'wa
tch': 1, 'too': 1})
```

In [59]:

```
print(bagsofwords[1])
```

```
Counter({'games': 1, 'John': 1, 'also': 1, 'to': 1, 'likes': 1, 'wat
ch': 1, 'football': 1})
```

In [60]:

```
sumbags = sum(bagsofwords, collections.Counter())
sumbags
```

Out[60]:

```
Counter({'John': 2,
         'Mary': 1,
         'also': 1,
         'football': 1,
         'games': 1,
         'likes': 3,
         'movies': 1,
         'to': 2,
         'too': 1,
         'watch': 2})
```

Another method is to exploit the `scikit-learn` module:

In [61]:

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(analyzer = "word", tokenizer = None, preprocessor =
 None, stop_words = None,
                             max_features = 5000)
```

In [62]:

```
bows = vectorizer.fit_transform(docs)
bows = bows.toarray()
vocab = vectorizer.get_feature_names()
print(vocab)
```

```
['also', 'football', 'games', 'john', 'likes', 'mary', 'movies', 't
o', 'too', 'watch']
```

In [63]:

```
print(bows.shape)
```

```
(2, 10)
```

In [64]:

```
import numpy as np
for doc in bows:
    dic = dict(zip(vocab, doc))
    print(dic)
```

```
{'john': 1, 'games': 0, 'also': 0, 'movies': 1, 'to': 1, 'watch': 1,
'likes': 2, 'too': 1, 'mary': 1, 'football': 0}
{'john': 1, 'games': 1, 'also': 1, 'movies': 0, 'to': 1, 'watch': 1,
'likes': 1, 'too': 0, 'mary': 0, 'football': 1}
```

# Reducing the representation

- high frequency words, i.e. most common words in a language, are usually not very important (in the sense that they do not contain any interesting information)
- they can be removed
- *stop words list*:
  - conjunctions
  - popular words
- different lists for different languages
- within a language - different lists for different domains
- types of removal:
  - dictionary-like - all words from a dictionary (list) are removed from text
  - statistical - all words with a frequency of occurrence from a given range are removed
  - hybrid - combination of the above techniques
- a risky operation, it may lead to information loss
- it increases processing performance
- sometimes it improves classification results (due to the removal of information noise)

## Stop word list in English

In [65]:

```
from nltk.corpus import stopwords
stop = stopwords.words('english')
sentence = "this is a foo bar sentence"
print([i for i in sentence.split() if i not in stop])
```

```
['foo', 'bar', 'sentence']
```

In [66]:

```
print(stop)
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you',
'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'hims
elf', 'she', 'her', 'hers', 'herself', 'it', 'its', 'itself', 'the
y', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who',
'whom', 'this', 'that', 'these', 'those', 'am', 'is', 'are', 'was',
 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'd
o', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'o
r', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'wit
h', 'about', 'against', 'between', 'into', 'through', 'during', 'bef
ore', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in',
 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'o
nce', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any',
 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'n
o', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'ver
y', 's', 't', 'can', 'will', 'just', 'don', 'should', 'now', 'd', 'l
l', 'm', 'o', 're', 've', 'y', 'ain', 'aren', 'couldn', 'didn', 'doe
sn', 'hadn', 'hasn', 'haven', 'isn', 'ma', 'mightn', 'mustn', 'need
n', 'shan', 'shouldn', 'wasn', 'weren', 'won', 'wouldn']
```

As already mentioned, there is some risk related with the usage of stop word lists:

In [67]:

```
stop = stopwords.words('english')
sentence = "to be or not to be"
print([i for i in sentence.split() if i not in stop])
```

```
[]
```

**Remark** For the sake of efficiency it is a good idea to convert lists into sets, because in Python searching for elements in a set is faster than in a list.

## Stop words in other languages

- https://github.com/trec-kba/many-stop-words (https://github.com/trec-kba/many-stop-words) - simple Python package that provides a single function for loading sets of stop words for different languages (English, French, German, Finish, Hungarian, Turkish, Russian, Czech, Greek, Arabic, Chinese, Japanese, Korean, Catalan, Polish, Hebrew, Norwegian, Swedish, Italian, Portuguese and Spanish)
- http://www.ranks.nl/stopwords (http://www.ranks.nl/stopwords) - collection of stopword lists in 40+ languages

In [68]:

```
import requests
from bs4 import BeautifulSoup

r = requests.get('http://www.ranks.nl/stopwords/polish',verify=False)
soup = BeautifulSoup(r.content,"lxml")

stoppl = []
for i in soup.find_all('td'):
    stoppl.extend(i.text.split())
```

```
/usr/local/lib/python3.5/dist-packages/urllib3/connectionpool.py:85
8: InsecureRequestWarning: Unverified HTTPS request is being made. A
dding certificate verification is strongly advised. See: https://url
lib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warnings
  InsecureRequestWarning)
```

In [69]:

```
print(stoppl)
```

```
['ach', 'aj', 'albo', 'bardzo', 'bez', 'bo', 'być', 'ci', 'cię', 'ci
ebie', 'co', 'czy', 'daleko', 'dla', 'dlaczego', 'dlatego', 'do', 'd
obrze', 'dokąd', 'dość', 'dużo', 'dwa', 'dwaj', 'dwie', 'dwoje', 'dz
iś', 'dzisiaj', 'gdyby', 'gdzie', 'go', 'ich', 'ile', 'im', 'inny',
 'ja', 'ją', 'jak', 'jakby', 'jaki', 'je', 'jeden', 'jedna', 'jedn
o', 'jego', 'jej', 'jemu', 'jeśli', 'jest', 'jestem', 'jeżeli', 'ju
ż', 'każdy', 'kiedy', 'kierunku', 'kto', 'ku', 'lub', 'ma', 'mają',
 'mam', 'mi', 'mną', 'mnie', 'moi', 'mój', 'moja', 'moje', 'może',
 'mu', 'my', 'na', 'nam', 'nami', 'nas', 'nasi', 'nasz', 'nasza', 'n
asze', 'natychmiast', 'nią', 'nic', 'nich', 'nie', 'niego', 'niej',
 'niemu', 'nigdy', 'nim', 'nimi', 'niż', 'obok', 'od', 'około', 'o
n', 'ona', 'one', 'oni', 'ono', 'owszem', 'po', 'pod', 'ponieważ',
 'przed', 'przedtem', 'są', 'sam', 'sama', 'się', 'skąd', 'tak', 'ta
ki', 'tam', 'ten', 'to', 'tobą', 'tobie', 'tu', 'tutaj', 'twoi', 'tw
ój', 'twoja', 'twoje', 'ty', 'wam', 'wami', 'was', 'wasi', 'wasz',
 'wasza', 'wasze', 'we', 'więc', 'wszystko', 'wtedy', 'wy', 'żaden',
'zawsze', 'że']
```

In [70]:

```
sentence = "być albo nie być"
print([i for i in sentence.split() if i not in stoppl])
```

```
[]
```

In [71]:

```
sentence = "to zdanie powinno być lepszym przykładem"
print([i for i in sentence.split() if i not in stoppl])
```

```
['zdanie', 'powinno', 'lepszym', 'przykładem']
```

# Zipf's law

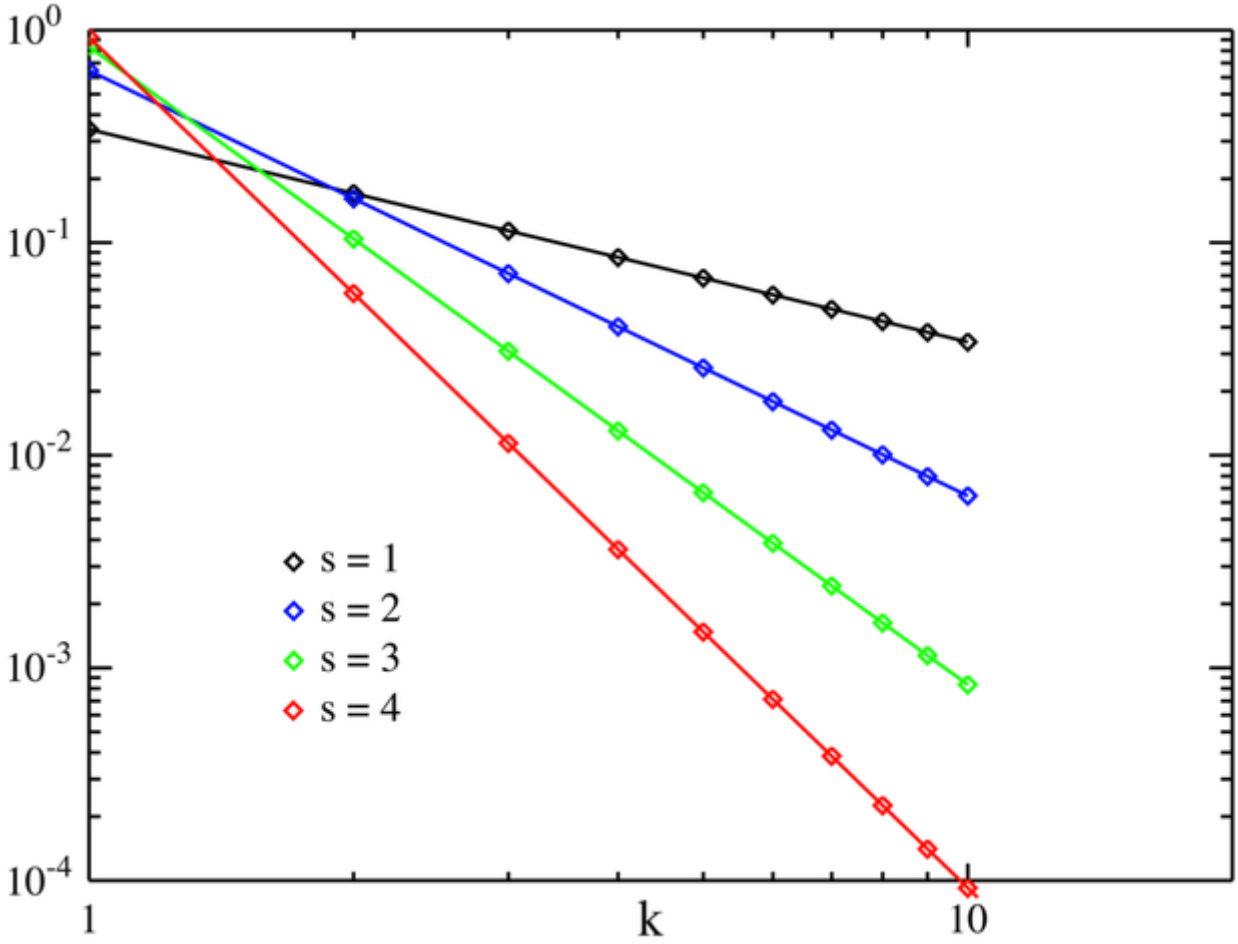
- empirical law
- models the distribution of words across documents
- first noticed by the French stenographer Jean-Baptiste Estoup at the end of XIX century
- verified and popularized by the American linguist George Kingsley Zipf

> If $t_1$ is the most common term in the collection, $t_2$ is the next most common, and so on, then the collection frequency $f_i$ of the $i$th most common term is proportional to $1/i$ - **frequency decreases very rapidly with rank**

- if $r$ is the rank and $f$ the frequency of occurrence, then according to the Zipf-Estoup law we have:

$$r \times f = const$$

- the value of the constant depends on the length of the text
- Zipfian distribution (a discrete power law distribution) is a theoretical model
- empirical distribution may differ from the law, in particular for less frequent words
- nevertheless, it is a very good model for most languages, even the non-natural ones like Esperanto
- it is still not well understood, why the law holds
  - one of the explanations is based on the statistical analysis (by Vitold Belevitch)
    - take a large class of well-behaved statistical distributions (not only the normal distribution)
    - express them in terms of a rank
    - expand each expression into a Taylor series
    - in every case **a first-order truncation of the series resulted in Zipf's law**
  - **principle of least effort** (proposed by Zipf himself) - neither speakers nor hearers using a given language want to work any harder than necessary to reach understanding, and the process that results in approximately equal distribution of effort leads to the observed Zipf distribution
- interestingly, the more the word distribution of an author differs from the Zipf's law, the less understandable his style is




## Example - conclusions from Zipf's law

In [72]:

```
import collections
import requests
import re

r = requests.get('http://www.gutenberg.org/cache/epub/103/pg103.txt')
```

In [73]:

```
bag = collections.Counter(re.findall(r'\w+',r.content.decode('utf-8')))
```

In [74]:

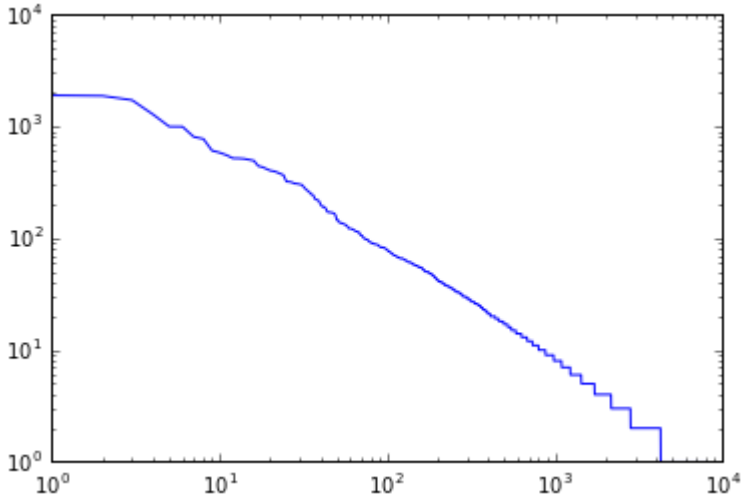
```
import matplotlib.pyplot as plt
```

In [75]:

```
plt.loglog(sorted(bag.values(),reverse=True))
```

Out[75]:

```
[<matplotlib.lines.Line2D at 0x7fcd9efb16d8>]
```




In [76]:

```
print(bag.most_common(20))
```

```
[('the', 4312), ('and', 1896), ('of', 1876), ('to', 1720), ('a', 128
1), ('in', 997), ('was', 995), ('his', 807), ('he', 773), ('Fogg', 6
08), ('at', 583), ('with', 553), ('not', 519), ('The', 518), ('tha
t', 514), ('on', 504), ('had', 494), ('which', 442), ('for', 433),
 ('it', 418)]
```

- Zipf's law approximates well the empirical distribution
- usually, most frequent words are the stop words
- however, the name of the protagonist is on the top 20 most frequent words
- **reduce representation with care**

## Interesting fact - lipograms

- a kind of constrained writing or word game
- writing paragraphs or longer works in which a particular letter or group of letters is avoided
- ancient Greek texts avoiding the letter sigma are the earliest examples of lipograms
- lipograms may break the Zipf's law

#### Ernest Vincent Wright, "Gadsby"

- a novel, 50000 words
- the author claimed that there is no word with the letter "e" in the novel
- published versions of the book do contain a handful of uses of the letter "e"

> If Youth, throughout all history, had had a champion to stand up for it; to show a doubting world that a child can think; and, possibly, do it practically; you wouldn't con- stantly run across folks today who claim that "a child don't know anything." A child's brain starts functioning at birth ; and has, amongst its many in- fant convolutions, thousands of dormant atoms, in- to which God has put a mystic possibility for notic- ing an adult's act, and figuring out its purport.

#### Julian Tuwim, "Pegaz dęba"

- a paragraph with no letter "r"

> Słońce tego dnia wstało jakieś dziwnie leniwe, matowe bez blasku. Około południa na powleczone niezwykłą bladością niebo wypełzły zwały skłębionych żółtych obłoków i w jednej chwili świat zasnuł się ciemnością.

# Stemming and lemmatization

- **stemming** - is a process of transforming a word into its stem, i.e. a normalized form

  learns, learned, learning --> learn
  stems, stemmer, stemming, stemmed --> stem

- not a big problem for English - publicly available algorithms give good results
- in Polish language many different forms correspond to the same word:

śmiać, śmieję, śmiałem, śmiałam, śmiejesz, śmiałeś, śmiałaś, śmieje, śmiał, śmiała, śmiejemy, śmialiśmy, śmiejecie, śmialiście, śmieją, śmiali

- **lemmatization** - a process of grouping together the inflected forms of a word so they can be analysed as a single item, identified by the word's lemma, or dictionary form, e.g.

  better --> good
  worse --> bad

- first algorithms proposed already in 1960s
- stemming and lemmatization are closely related to each other
  - a stemmer operates on a single word without knowledge of the context (it cannot discriminate between words which have different meanings depending on part of speech)
  - stemmers are typically easier to implement and run faster
  - their reduced "accuracy" may not matter for some applications

## Useful definitions

- **morphology** - the study of words, how they are formed, and their relationship to other words in the same language
- **morpheme** - the smallest meaningful unit of a language (not identical to a word)
- **morphological analysis** - automatic identification of morphemes in a word
- **root** - the primary lexical unit of a word
- **affix** - a morpheme that is attached to a word stem (root) to form a new word or word form
  - **prefix**, e.g. **un**-do
  - **suffix**, e.g. do-**ing**
  - **infix**, e.g. Abso-**bloody**-lutely
- **inflection** - the modification of a word to express different grammatical categories such as tense, case, voice, aspect, person, number, gender, and mood
- **morphological derivation** - the process of forming a new word from an existing word, often by adding a prefix or suffix (happiness and unhappy derive from the root word hap)

## Porter algorithm

- stemming algorithm
- unwritten standard for English
- invented by Martin Potter in 1979
- an open source implementation of the algorithm, published by the author, may be found at http://tartarus.org/~martin/PorterStemmer/ (http://tartarus.org/~martin/PorterStemmer/)
- Snowball (http://snowball.tartarus.org/texts/introduction.html (http://snowball.tartarus.org/texts/introduction.html)) - a small string processing language designed

for creating stemming algorithms
- a general rule is an iterative removal of suffixes

#### Rules and steps

The algorithm consist of 5 main steps, each of which is described by a set of rules for suffix stripping:

1. Depluralization (removing a plural morpheme) and simple suffixes ("-es", "-ed" "-ing")
2. Double suffix replacement ("tional" --> "tion")
3. Removal of adjective and infinitive forms etc. ("ness" --> "", "alize" --> "al", "icate" --> "ic")
4. Removal of "ant", "ence" etc.
5. Removal of 'e' suffix, double consonant reduction ("ll" --> "l")

The source code for the Porter stemmer in Python may be downloaded from
http://tartarus.org/~martin/PorterStemmer/python.txt (http://tartarus.org/~martin/PorterStemmer/python.txt)

#### Porter's algorithm in NLTK

In [77]:

```
from nltk.stem.porter import PorterStemmer
porter_stemmer = PorterStemmer()

sentence = 'John also likes to watch football games'

for i in sentence.split():
    print(porter_stemmer.stem(i))
```

```
john
also
like
to
watch
footbal
game
```

## Stemming in other languages

- NLTK package provide stemmers for several languages: English, Danish, Dutch, Finnish, French, German, Hungarian, Italian, Norwegian, Portuguese, Romanian, Russian, Spanish, Swedish
- in case of Polish there are several possibilities:
  - https://github.com/Tutanchamon/pl_stemmer/blob/master/pl_stemmer.py (https://github.com/Tutanchamon/pl_stemmer/blob/master/pl_stemmer.py) - very simple stemmer
  - http://morfologik.blogspot.com/ (http://morfologik.blogspot.com/) (Java)
  - http://www.getopt.org/stempel/ (http://www.getopt.org/stempel/) (Java)