Data Mining using Python — a case

Finn Årup Nielsen

DTU Compute
Technical University of Denmark

August 31, 2014

A case

Download information from the web (here a Semantic MediaWiki with information about papers on Wikipedia research)

Extract features

Topic mining of texts

Graph mining of coauthors

See Python code at https://gist.github.com/3603279

Downloading a Web page

Define what we want to read:

```
# Define a url as a Python string
url = "http://wikilit.referata.com/" + \
    "wiki/Special:Ask/" + \
    "-5B-5BCategory:Publications-5D-5D/" + \
    "-3FHas-20author%3DAuthor(s)/-3FYear/" + \
    "-3FPublished-20in/-3FAbstract/-3FHas-20topic%3DTopic(s)/" + \
    "-3FHas-20domain%3DDomain(s)/" +  \
    "format%3D-20csv/limit%3D-20100/offset%3D0"
```

Downloading a Web page

Read information from the Web (Martelli et al., 2005, p. 489)

```
# Import the 'urllib' module for Web page retrieval
from urllib import urlopen

# Get help on how to use the module
help(urlopen)

# Get and read the web page
doc = urlopen(url).read()  # Object from urlopen has read method
```

`urlopen(url)` returns a file-like object that has a read method

The '`doc`' variable is a Python string.

Output from the print function

Show the first 1000 characters from the string with

```
print(doc[:1000])
```

gives the result

```
,Author(s),Year,"Published in",Abstract,Topic(s),Domain(s)
"'Wikipedia, the free encyclopedia' as a role model? Lessons for open innovation
 from an exploratory examination of the supposedly democratic-anarchic nature of
 Wikipedia","Gordon Müller-Seitz,Guido Reger",2010,"International Journal of Tec
hnology Management","Accounts of open source software {(OSS)} development projec
ts frequently stress their democratic, sometimes even anarchic nature, in contra
st to for-profit organisations. Given this observation, our research evaluates q
ualitative data from Wikipedia, a free online encyclopaedia whose development me
chanism allegedly resembles that of {OSS} projects. Our research offers contribu
tions to the field of open innovation research with three major findings. First,
 we shed light on Wikipedia as a phenomenon that has received scant attention fr
om management scholars to date. Second, we show that {OSS-related} motivational
mechanisms partially apply to Wikipedia participants. Third,
```

Comma-separated values

Reading the comma separated values

Import a CSV reader/writer module

```
# Note: usually you will have all the imports at the
# top of the Python file.
import csv

web = urlopen(url)
# 'web' is now a file-like handle

lines = csv.reader(web, delimiter=',', quotechar='"')
# 'papers' is now an object that can be iterated

# Iterate over 'lines' with a Python for loop
for line in lines:
    # process each line here
```

Outout from the print function

Each row is of a Python 'list' type

```
isinstance(line, list) == True    # Or  type(line) == list
```

Pretty printing a line:

```
>>> import pprint
>>> pprint.pprint(line)
[u'Cross-cultural analysis of the Wikipedia community',
 u'Noriko Hara,Pnina Shachaf,Khe Foon Hew',
 u'2010',
 u'Journal of the American Society for Information Science ...
 u"This article reports a cross-cultural ...
```

Or JSON reading

Change the Semantic MediaWiki query URL slightly

```
# JSON format instead that Semantic MediaWiki also exports
url_json = "http://wikilit.referata.com/" + \
    "wiki/Special:Ask/" + \
    "-5B-5BCategory:Publications-5D-5D/" + \
    "-3FHas-20author/-3FYear/" + \
    "-3FPublished-20in/-3FAbstract/-3FHas-20topic)/" + \
    "-3FHas-20domain/" +  \
    "format%3D-20json"
```

Or JSON reading

Python module for JSON reading

```
import simplejson as json

# Read JSON into a Python structure
response = json.load(urlopen(url_json))

# 'response' is now a hash/dictionary
response.keys()
# Result: ['rows', 'results', 'printrequests']

# response['printrequests'] is a list, map iterates over the list
columns = map(lambda item: item['label'], response['printrequests'])
# gives ['', 'Has author', 'Year', 'Published in', 'Abstract',
#        'Has topic)', 'Has domain']
```

Back to CSV

```
# Reread CSV
lines = csv.reader(urlopen(url), delimiter=',', quotechar='"')

# Iterate over 'lines' and insert the into a list of dictionaries
header = []
papers = []
for row in lines:
    # csv module lacks unicode support!
    line = [unicode(cell, 'utf-8') for cell in row]
    if not header:
        # Read the first line as header
        header = line
        continue
    papers.append(dict(zip(header, line)))
```

List of dictionaries

"papers" is now an list of dictionaries

To get the first abstract:

```
>>> papers[0]['Abstract']
u'Accounts of open source software {(OSS)} development projects
frequently stress their democratic, sometimes even anarchic
nature, in contrast to for-profit organisations.
Given this observation,...
```

Note Python indexes from 0, i.e., papers[0] is the first element in the list.

papers[0]['Abstract'] is the value indexed by the string "Abstract" in the first element in the list.

Natural language processing

The `nltk` module is a versatile natural language processing toolkit. Here used to find the individual words and other tokens in an abstract:

```
# Get some natural language processing tools
import nltk

# Get words from first abstract
nltk.word_tokenize(papers[0]['Abstract'])
# Result: [u'Accounts', u'of', u'open', u'source', ...
```

We would like to have lower case words for all the elements in the list.

Note we are doing it wrong here: We should do sentence tokenization before word tokenization!

Lower case

The map function let one apply a function to individual elements in a list.

```
# Lower case words. 'string' module from Python library
import string

map(string.lower, nltk.word_tokenize(papers[0]['Abstract']))
# Result: [u'accounts', u'of', u'open', u'source'
```

Here the function is the "lower" function available in the string module and the list is the list of words.

Instead of using `map` we could have used list comprehension:

```
[word.lower() for word in nltk.word_tokenize(papers[0]['Abstract'])]
```

Now for all papers

Iterate over all papers and extract the words for each paper adding the list of words to a field in the dictionary:

```
for paper in papers:
    words = map(string.lower, nltk.word_tokenize(paper['Abstract']))
    paper.update({'words': words})
```

Now the words is in a list in a dictionary in list:

```
>>> print(papers[0]['words'])
[u'accounts', u'of', u'open', u'source', u'software', u'{', ...
```

Note that ‘paper’ is a reference to individual elements in ‘papers’, — not a copy.

Word statistics

```
# Double list comprehension
all_words = [word for paper in papers for word in paper['words']]

len(all_words)
# Result: 17059

# Unique words, using a Python 'set' type
len(set(all_words))
# Result: 3484

# Count the occurences of all words
wordcounts = dict([[t, all_words.count(t)] for t in set(all_words)])

# Another way
wordcounts = {}
for term in all_words:
    wordcounts[term] = wordcounts.get(term, 0) + 1
```

Show most frequent words

```
# Change the ordering of value and key for sorting
items = [(v, k) for k, v in wordcounts.items()]

for count, word in sorted(items, reverse=True)[:5]:
    print("%5d %s" % (count, word))

#  913 the
#  706 of
#  658 ,
#  507 and
#  433 to
```

For topic mining we would like to get rid of these words.

Determine “interesting” words

Reading a stopword list from the NLTK corpus

```
import nltk.corpus
stopwords = nltk.corpus.stopwords.words('english')

terms = {}
for word, count in wordcounts.iteritems():
    if count > 2 and word not in stopwords and word.isalpha():
        terms[word] = count
```

New word statistics

```
# Change the ordering of value and key for sorting
items = [(v, k) for k, v in terms.items()]

for count, word in sorted(items, reverse=True)[:5]:
    print("%5d %s" % (count, word))

#  213 wikipedia
#   64 knowledge
#   64 article
#   54 information
#   50 articles

# Wikipedia is the main topic of all the papers to remove it
terms.pop('wikipedia')

# Convert the dictionary to a list.
terms = list(terms)
```

Making a matrix

To make numerical processing in Python import the `numpy` module and then initialize the elements

```
# Import of the numerical module
import numpy as np

# Construct a bag-of-words matrix
M = np.asmatrix(np.zeros([len(papers), len(terms)]))
for n, paper in enumerate(papers):
    for m, term in enumerate(terms):
        M[n, m] = paper['words'].count(term)
```

The `M` matrix has the size papers-by-terms and each element is set to the number of times a term (i.e., a word) occurs in an abstract.

Multivariate analysis algorithm

An algorithm for non-negative matrix factorization to determine topics in the corpus:

```
# Define a topic mining function (non-negative matrix factorization)
def nmf(M, components=5, iterations=5000):
    # Initialize to matrices
    W = np.asmatrix(np.random.random(([M.shape[0], components])))
    H = np.asmatrix(np.random.random(([components, M.shape[1]])))
    for n in range(0, iterations):
        H = np.multiply(H, (W.T * M) / (W.T * W * H + 0.001))
        W = np.multiply(W, (M * H.T) / (W * (H * H.T) + 0.001))
        print "%d/%d" % (n, iterations)    # Note 'logging' module
    return (W, H)
```

Python function with 3 input arguments, where one is required and 2 has default arguments.

Using the algorithm on the matrix

Call the function and show the results:

```
# Perform the actual computation
W, H = nmf(M, iterations=50, components=3)

# Show the results in some format
for component in range(W.shape[1]):
    print("=" * 80)
    print("COMPONENT %d: " % (component,))
    indices = (-H[component,:]).getA1().argsort()
    print(" - ".join([ terms[i] for i in indices[:6] ]))
    print("-")
    indices = (-W[:,component]).getA1().argsort()
    print("\n".join([ papers[i][''] for i in indices[:5] ]))
```

Results

```
==========================================================================
COMPONENT 0:
knowledge - article - information - articles - approach - use
-
Constructing commons in the cultural environment
A systemic and cognitive view on collaborative knowledge building with wikis
Academics and Wikipedia: reframing Web 2.0+as a disruptor of traditional ...
Addressing gaps in knowledge while reading
Contextual retrieval of single Wikipedia articles to support the ...
==========================================================================
COMPONENT 1:
web - media - users - content - production - sites
-
A cultural and political economy of Web 2.0
Academics and Wikipedia: reframing Web 2.0+as a disruptor of ...
Applications of semantic web methodologies and techniques to social ...
Classifying tags using open content resources
A Wikipedia matching approach to contextual advertising
==========================================================================
COMPONENT 2:
question - classification - answer - systems - answering - method
-
A semantic approach for question classification using WordNet and Wikipedia
```

Graph mining with NetworkX

Use the NetworkX package for storing and analyzing graphs and the matplotlib package for plotting

```
# Import a graph library and plotting library
import networkx as nx
import matplotlib.pyplot as plt

# Generate coauthor graph
coauthor_graph = nx.Graph()
for paper in papers:
    coauthors = paper['Author(s)'].split(',')
    for n in range(len(coauthors)-1):
        for m in range(n, len(coauthors)):
            coauthor_graph.add_edge(coauthors[n], coauthors[m])
```

Plotting a part of the graph

There are numerous graph mining functions in NetworkX. Here is one that extracts the connected components in the coauthor graph:

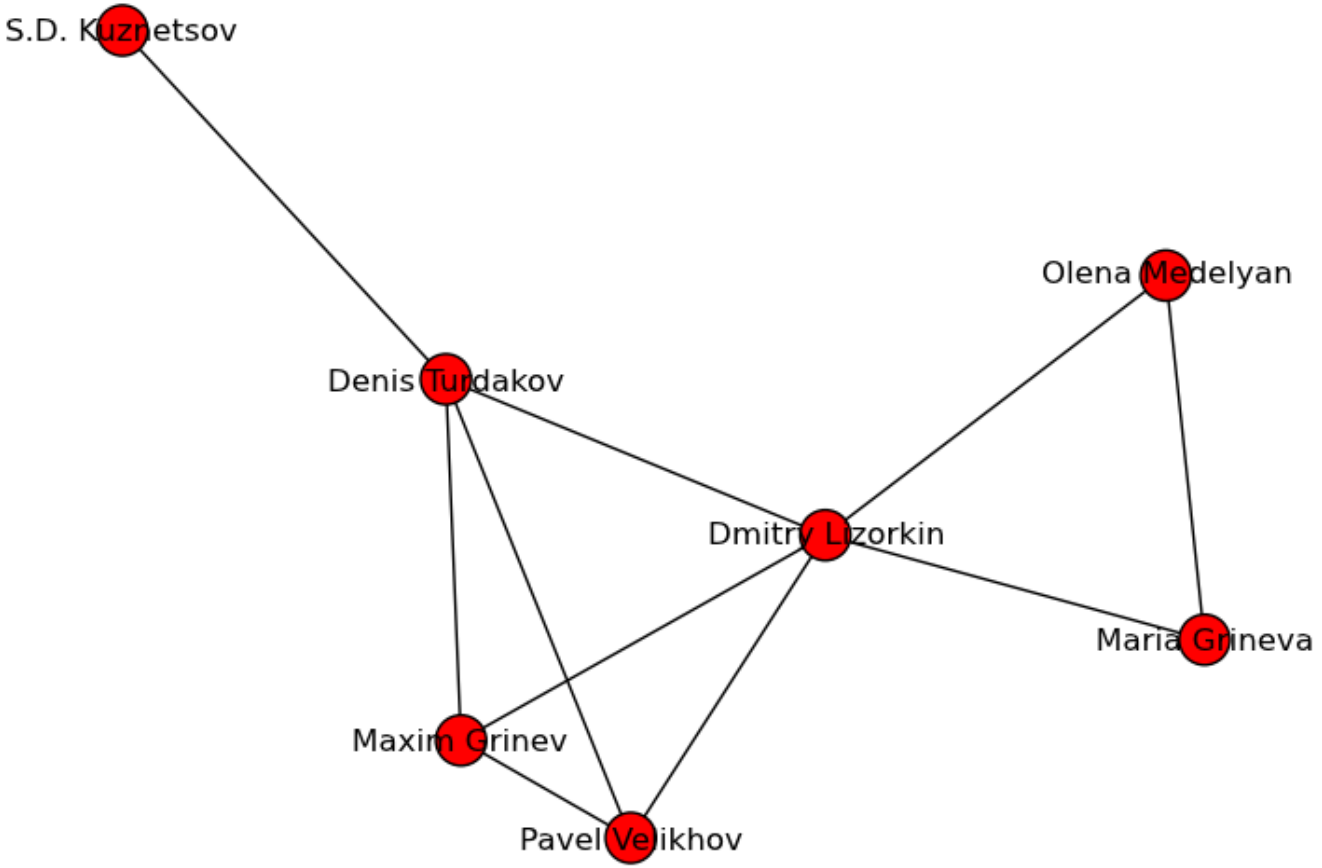
```
# Extract a subgraph
author_communities = nx.connected_component_subgraphs(coauthor_graph)
```

Take and plot the biggest connected component:

```
# Plot the graph
nx.draw(author_communities[0])
plt.show()

# plt.savefig("coauthorgraph.png")
```

Result


S.D. Kuznetsov
Olena Medelyan
Denis Turdakov
Dmitry Lizorkin
Maria Grineva
Maxim Grinev
Pavel Velikhov

IPython Notebook

IPython Notebook version

Yet another case (does not work as it queries a Semantic MediaWiki that has been taken down)

Downloading a Web page

Read information from the Web (Martelli et al., 2005, p. 489)

```
# Import the 'urllib' library for Web page retrieval
from urllib import urlopen

url = 'http://rb.imm.dtu.dk/w/index.php/' + \
      'Special:Ask/-5B-5Bfeed::+-5D-5D/-3FFeed/' + \
      'sort=/order=ASC/format=csv/sep=,/limit=100'

help('urllib')

# Get and read the web page
doc = urlopen(url).read()  # Object from urlopen has read function

print(doc)
```

Output from the print function

```
"Autoblog Green",http://feeds.autoblog.com/weblogsinc/autoblog
"Brussels Sunshine",http://blog.brusselssunshine.eu/feeds/posts/default
"Capital Eye",http://www.opensecrets.org/news/atom.xml
Causecast,http://feeds.feedburner.com/causecast/latest_news.rss
"Clean Fuels Blog",http://feeds.feedburner.com/CFDC?format=xml
"Close Concerns Weblog",http://closeconcerns.typepad.com/close_ ...
"Corporate Eye Corporate social responsibility",http://feeds. ...
"Corporate social responsibility (guardian)",http://www.guardian. ...
"Corpwatch Blog",http://www.corpwatch.org/rss.php
"Crane and Matten blog",http://craneandmatten.blogspot.com/feeds ...
"Dax Mahoney (blog)",http://daxmahoney.blogspot.com/feeds/posts/default
...
```

Reading the comma separated values

```
# Import a CSV reader/writer library.
import csv

web = urlopen(url)
# 'web' is now a file-like handle

blogs = csv.reader(web, delimiter=',', quotechar='"')
# 'blogs' is now an object that can be iterated over

# Iterate over 'blogs'
for blog in blogs:
    print(blog)
```

Outout from the print function

```
['Autoblog Green', 'http://feeds.autoblog.com/weblogsinc/autoblog']
['Brussels Sunshine', 'http://blog.brusselssunshine.eu/fe ...
['Capital Eye', 'http://www.opensecrets.org/news/atom.xml']
['Causecast', 'http://feeds.feedburner.com/causecast/latest_ ...
['Clean Fuels Blog', 'http://feeds.feedburner.com/CFDC?format=xml']
['Close Concerns Weblog', 'http://closeconcerns.typepad. ...
['Corporate Eye Corporate social responsibility', 'http:// ...
['Corporate social responsibility (guardian)', 'http://www ...
['Corpwatch Blog', 'http://www.corpwatch.org/rss.php']
['Crane and Matten blog', 'http://craneandmatten.blogspot.c ...
['Dax Mahoney (blog)', 'http://daxmahoney.blogspot.com/feed ...
['Dgoodr', 'http://feeds.feedburner.com/dgoodr?format=xml']
...
```

Adding the URLs to a Python list

```
# Each row is of a Python 'list' type
isinstance(blog, list)   # or type(blog)

blogs = csv.reader(urlopen(url), delimiter=',', quotechar='"')

# Create empty list
urls = []

for blog in blogs:
    # Python indexes from 0: '1' is the second column
    feed = blog[1]
    if len(feed):
        urls.append(feed)
```

The feeds

```
>>> urls[0]
'http://feeds.autoblog.com/weblogsinc/autoblog'

>>> doc = urlopen(urls[0]).read()
>>> print(doc[0:600])
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" media="screen" href="/~d/ ...
<channel>
<title>Autoblog</title>
<link>http://www.autoblog.com</link>
<description>Autoblog</description>
<image>
<url>http://www.blogsmithmedia.com/www.autoblog.com/media/feedlogo.
```

Reading feeds

Reading feeds from, e.g., blogs (Segaran, 2007, p. 229)

```
import feedparser

f = feedparser.parse(urls[0])
# Now 'f' is a kind of Python dictionary

>>> f.keys()
['feed', 'status', 'updated', 'version', 'encoding', 'bozo',
'headers', 'etag', 'href', 'namespaces', 'entries']

>>> f['entries'][0].keys()
['summary_detail', 'author', 'links', 'title', 'feedburner_origlink',
'tags', 'updated', 'comments', 'summary', 'guidislink',
'title_detail', 'link', 'id', 'updated_parsed']
```

Examining the feeds

The 7th tag of the 1st entry in the downloaded feed

```
>>> f['entries'][0]['tags'][6]['term']
u'obama administration'
```

All tags for all posts in one particular feed:

```
tags = [];
for e in f['entries']:
    for t in e['tags']:
        tags.append(t['term']);
```

Getting all feeds

```
from BeautifulSoup import BeautifulSoup          # HTML reading library
fs = [];
for url in urls:
 fs.append(feedparser.parse(url))
 fs[-1]['wordlist'] = []
 for e in fs[-1]['entries']:
  if e.has_key('summary'):
   fs[-1]['wordlist'].extend(''.join(BeautifulSoup( \
    e.summary).findAll(text=True)).split());
 print(url)

allwords = [word for f in fs for word in f['wordlist']]
```

Some statistics

```
float(len(allwords))/len(set(allwords))

wordcount = dict([ [t, allwords.count(t)] for t in set(allwords) ])

wordcount = {}
for term in allwords:
    wordcount[term] = wordcount.get(term, 0) + 1

items = [(v, k) for k, v in wordcount.items()]
items.sort()
items.reverse()    # or items.sort(reverse=True)

for n in range(0,2000):
    print('%3d: %4d %s' % (n+1, items[n][0], items[n][1]))

  1: 5205 the
  2: 3211 to
  3: 2922 of
```

Determine “interesting” words

Reading a stopword list

```
stopwords = [ line.strip() for line in open('stop_english1.txt', 'r') ]

wordcount = {}
for term in allwords:
    if term.lower() not in stopwords:
        wordcount[term] = wordcount.get(term, 1) + 1

items = [(v, k) for k, v in wordcount.items()]  # 'Inverting' the dict
items.sort()
items.reverse()

terms = []
for n in range(0,500):
    terms.append(items[n][1])
    print('%3d: %4d %s' % (n+1, items[n][0], items[n][1]))
```

Making a matrix

To make numerical processing in Python import the `numpy` module and then initialize the elements

```
import numpy                    # Import of the numerical module

M = numpy.matrix(numpy.zeros([len(fs), len(terms)]))
for n in range(len(fs)):
    for m in range(len(terms)):
        M[n,m] = fs[n]['wordlist'].count(terms[m])
```

The `M` matrix has the size feeds-by-terms and each element is set to the number of times a term (i.e., a word) occurs in a feed.

Multivariate analysis algorithm:

```
def nmf(M, components=5, iterations=5000):
    import random
    W = numpy.matrix(numpy.zeros([M.shape[0], components]))
    H = numpy.matrix(numpy.zeros([components, M.shape[1]]))
    for n in range(M.shape[0]):
        for m in range(components):
            W[n,m] = random.random()
    for n in range(components):
        for m in range(M.shape[1]):
            H[n,m] = random.random()
    for n in range(0, iterations):
        H = numpy.multiply(H, (W.T * M) / (W.T * W * H + 0.001))
        W = numpy.multiply(W, (M * H.T) / (W * (H * H.T) + 0.001))
        print "%d/%d" % (n, iterations)
    return (W, H)
```

Using the algorithm on the matrix

```
(W, H) = nmf(M)
```

. . . And then display the results:

```
for n in range(4):
    print n, "-----"
    w = []; h = []
    for m in range(W.shape[0]):
        w.append((W[m,n], fs[m]))
    for m in range(H.shape[1]):
        h.append((H[n,m], terms[m]))
    h.sort()
    h.reverse()
    w.sort()
    w.reverse()
    for n in range(0,20):
      print "%5.3f %10s %s" % (h[n][0], h[n][1], w[n][1]['feed']['title'])
```

References

Martelli, A., Ravenscroft, A. M., and Ascher, D., editors (2005). *Python Cookbook*. O'Reilly, Sebastopol, California, 2nd edition.

Segaran, T. (2007). *Programming Collective Intelligence*. O'Reilly, Sebastopol, California.