Session 17

Text as Data 2 - Information Extraction and Unsupervised Methods

Snorre Ralund

Enriching your raw data

Unless you find a nice backdoor to a "hidden" API: scraping your own data and building a dataset means that you need to process and enrich the raw data.

Examples include:

- Looking up Geographical entities and merging it with latitude and longitude or other *Geodata*.
- Locating entities mention in text using *Named Entity Recognition* methods.
- Inferring *gender* and *ethnicity* from names in your data.

Today we are gonna look into method for doing this.

Extracting information from text using dictionary approaches.

Examples include:

- This could be applying a pretrained (on reviews) sentiment classifier on social media expressions on twitter.
- Applying *Lexical* and *Rulebased* Sentiment Analyzers on News Articles about Climate Change or Immigration.
- Measuring saliency of topics using Curated Concepts

Using unsupervised methods to Explore your text corpus

- Topic modelling (a widely used clustering method for text) large text corpuses and producing prototypical (i.e. unvalidated) measurements.

Appropriating results from the field of Natural Language Processing

More or less generalized AI means the ability to parse sentences. This is something the natural language processing community has worked a lot on. Stanford CoreNLP: http://corenlp.run/ (http://corenlp.run/)

- Named-entity-recognition
- Subject - Verb - Object.
- Discovering Events: Event detection and Entity resolution methods.

Agenda

- Name based methods for Inferring Gender and Ethnicity
  - utilizing python `set` operations
- Sentiment Analysis using Lexical Approaches.
  - more `set` operations.
- Unsupervised Learning for Text.
  - Word2Vec and Topic Modelling using Gensim.

Name-based methods Inferring Gender and Ethnicity

- Looking up names from (goverment) curated lists.
- Scraping data from Wikipedia and Building a character-based for model predicting gender and ethnicity "Name-ethnicity classification from open sources" (Ambekar et al. 2009)
- or having an even nicer (and larger) dataset to extract features from using unsupervised learning:
  - "Nationality Classification using Name Embeddings" Ye et al. 2017
  - http://www.name-prism.com/api (http://www.name-prism.com/api)

In [10]:
```
#df.iplot(x='x',y='y',categories='gender',text='name')
```

Using an unsupervised methods on 1 billion Facebook Likes I could construct a gender classifier with .99 accuracy. Have not tried it for ethnicity but it looks promising from the visualization.

Getting Curated list of Names and Gender

In [16]:
```
# scraping names from ankestyrelsen

import pandas as pd
# links found looking here: https://ast.dk/born-familie/navne/navnelister/godkendte-forn
avne
urls = {'female': 'https://ast.dk/_namesdb/export/names?format=xls&gendermask=1',
        'male': 'https://ast.dk/_namesdb/export/names?format=xls&gendermask=2',
        'unisex': 'https://ast.dk/_namesdb/export/names?format=xls&gendermask=3'}
dfs = {}
for name,url in urls.items():
    df = pd.read_excel(url,header)
    dfs[name] = df
```

In [4]:
```
# scrape ethnicity:
# Link located by looking in the network monitor for activity on this page:
# https://ast.dk/born-familie/navne/navnelister/udenlandske-navne
import requests
url = 'https://ast.dk/_namesdb/namecultures?published=1&page=1&pagesize=270'
response = requests.get(url)
```

In [15]:
```
import requests
url = 'https://ast.dk/_namesdb/namecultures?published=1&page=1&pagesize=270'
response = requests.get(url)
d = response.json()
import tqdm
import pandas as pd
data = []
for num in tqdm.tqdm(range(1,d['pages']+1)):
    url = 'https://ast.dk/_namesdb/namecultures?published=1&page=%d&pagesize=270'%(num)
    response = requests.get(url)
    d = response.json()
    data+=d['names']
df_eth = pd.DataFrame(data)
print(len(df_eth))
```

```
100%|████████████████████████████████████████████████████████████████████
██| 28/28 [00:15<00:00,  1.81it/s]

7441
```

In [19]:
```
df_eth.head()
```

Out[19]:

	culture	culture_id	female	male	name	name_id
0	afrikansk	1	False	True	Abadi	38385
1	muslimsk	2	False	True	Abadi	38385
2	muslimsk	2	False	True	Aban	5074
3	muslimsk	2	False	True	Abane	409
4	muslimsk	2	False	True	Abaq	5078

In [98]:

```
name2eth_gender = {}
for name,culture,sex, sex2 in df_eth[['name','culture','female','male']].values:
    if name in name2eth_gender:

        name2eth_gender[name]['culture'] = culture
        if sex&sex2:
            name2eth_gender[name]['sex'] = 'unisex'
            continue
        elif sex:
            name2eth_gender[name]['sex'] = 'female'
        elif sex2:
            name2eth_gender[name]['sex'] = 'male'
        else:
            print('eo')
    else:
        name2eth_gender[name] = {'culture':culture}
        if sex&sex2:
            name2eth_gender[name]['sex'] = 'unisex'
            continue
        elif sex:
            name2eth_gender[name]['sex'] = 'female'
        elif sex2:
            name2eth_gender[name]['sex'] = 'male'
        else:
            del name2eth_gender[name]
            #print(name,sex,sex2,'missing info')
```

In [99]:
```
names_used = set()
for sex,df in dfs.items():
    names = [df.columns[0]]+list(df.values[:,0])
    if sex=='unisex':
        continue
    print(sex,len(names),len(set(names)&set(name2eth_gender)))
    for name in names:
        if name in name2eth_gender:
            if name in names_used:
                name2eth_gender[name]['sex'] = 'unisex'
            continue
            name2eth_gender[name]['sex'].append(sex)
        else:
            name2eth_gender[name] = {'sex':sex}
            names_used.add(name)
```

```
female 21470 623
male 17571 2525
```

In [100]:
```
from collections import Counter
Counter([i['culture'] for i in name2eth_gender.values() if 'culture' in i]),Counter([i[
'sex'] for i in name2eth_gender.values()])
```

Out[100]:
```
(Counter({'Burma': 17,
          'afrikansk': 365,
          'indisk': 51,
          'muslimsk': 5456,
          'tamilsk': 1528}),
 Counter({'female': 21495, 'male': 20846, 'unisex': 969}))
```

In [101]:
```
import json
json.dump(name2eth_gender,open('name2eth_gender_json','w'))
```

Lookup method

Lookups in Sets (and dictionaries) are really fast, and Lists are slow! So for lookups, do not use Lists.

In [30]:
```
female = set([name for name,d in name2eth_gender.items() if d['sex']=='female'])
male = set([name for name,d in name2eth_gender.items() if d['sex']=='male'])
male_l = list(male)
print('Anders' in male)
```

```
True
```

In [36]:
```
%timeit 'Anders' in male
%timeit 'Anders' in male_l
```

```
52.5 ns ± 3.2 ns per loop (mean ± std. dev. of 7 runs, 10000000 loops each)
332 µs ± 18.6 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

Now lets us apply this technique for inferring gender to our Review Dataset

In [3]:
```
import pandas as pd
df = pd.read_csv('https://raw.githubusercontent.com/snorreralund/scraping_seminar/master/english_review_sample.csv')
```

In [38]:
```
def infer_gender(name):
    if name in name2eth_gender:
        return name2eth_gender[name]['sex']
    return 'unknown'
infer_gender('Anders')
```

Out[38]:
```
'male'
```

In [81]:
```
print(infer_gender('Lars Ole'))
from collections import Counter
Counter([len(i.split()) for i in name2eth_gender])
```

```
unknown
```

Out[81]:
```
Counter({1: 43090, 2: 216, 3: 4})
```

Problem: We need to design a lookup scheme that take into account, that we need to ectract the first name from the full name, while also handling that some names are actually spanning more than one token - ***name-grams?***.

In [ ]:

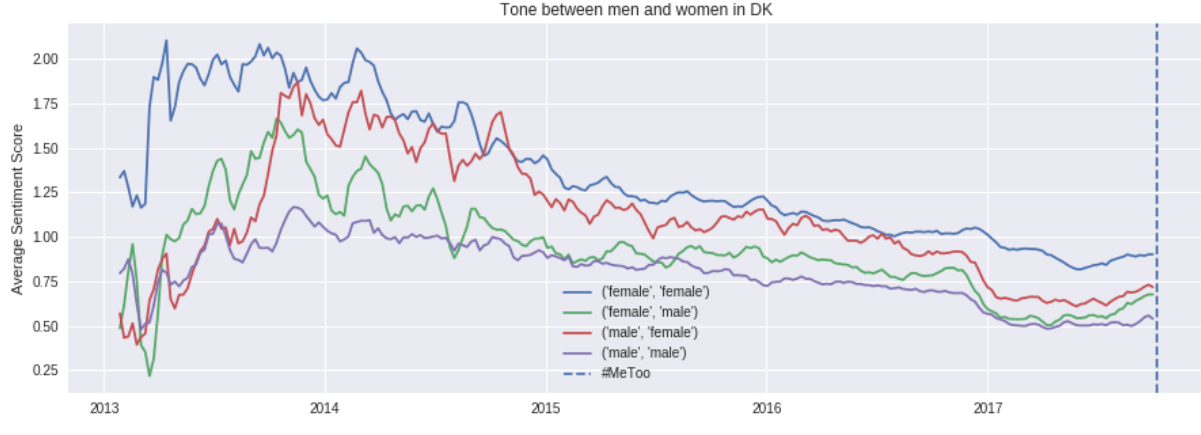
```
def infer_gender_ngram(full_name):
    names = full_name.split()
    firstname = names[0]
    if firstname in name2eth_gender:
        return name2eth_gender[firstname]['sex']
    n_names = len(names)-1
    if n_names>=2:
        second_name = names[1]
        if second_name in name2eth_gender:
            return name2eth_gender[second_name]['sex']
        combinations = zip(*[names[i:] for i in range(n_names)])
        for comb in combinations:
            if comb in name2eth_gender:
                name = ' '.join(comb)
                return name2eth_gender[name]['sex']
    return 'unknown'
print(infer_gender_ngram('Coolio Snorre Ralund'))
```

Lexical Based Sentiment Analysis

Lexical Based Sentiment Analysis (2)

Pros

- Comes with very low cost.
- Fast and scalable.
- Good for prototyping results.



Lexical Based Sentiment Analysis (2)

Cons

- You never really know the performance for your case.
- Performance can be really bad in some cases.

Lexical Based Sentiment Analysis (3)

Here we are gonna be comparing a few off-the-shelf lexical and rulebased approaches:

- Afinn (is danish!) (http://neuro.imm.dtu.dk/wiki/AFINN),
- Liu Hu (lexical) (http://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html) and
- Vader (lexical and rulebased) (https://github.com/cjhutto/vaderSentiment).
- **Purely Lexical** Naively Matching positive words. "*You are beautiful.*"
- **Rule-based** Can Adopt hard-coded rules to counter more or less simple negations. "*You are not particularly beautiful.*"

.. still lacks the ability to understand that the connotations: "*You are extremely beautiful on the outside.*"

- **Modelbased** Supervised Models trained to adopt many different expression of emotions: for some exciting ones lookup **DeepMoji**, or the so-called **sentiment neuron**.

Unsupervised Learning on Text Data

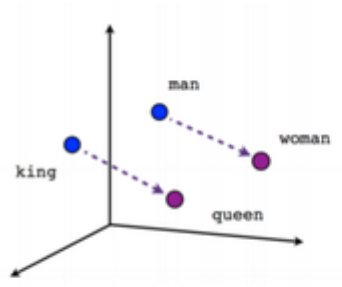
Word2Vec

Mikolov et. al 2013: *"Distributed Representations of Words and Phrases and their Compositionality"*

GOAL: Find a set of latent representations of N dimensions that maximize the dot products between each word and its context (i.e. the latent representation of words surrounding the target word).

Optimize Use gradient descent to search for a good solution. Run it large corpuses with billions of words, and context examples.

Result Major breakthroughs in the Field of Natural Language Processing



Male-Female

Verb tense

Country-Capital

Topic Modelling

Blei 2003: *"Latent Dirichlet Allocation"*

Goal Assume text and meaning is understood as a bag of words. Assume that the text is generated according to three concepts: a set of topics defined by word probabilities, a set of documents defined by their distribution of topics, and words described by a set of topic probabilities. No meaning only colocation.

Optimize Initialize clusters randomly. Initialize all probability distribution drawing multinomial distribution from model defined dirichlet priors. Optimize using Gibbs Sampling.

Result A surprisingly effective clustering method widely used accross disciplines (from humanities, to medicine, engineering, and social science) discovering and "measuring" topic proportions in texts.

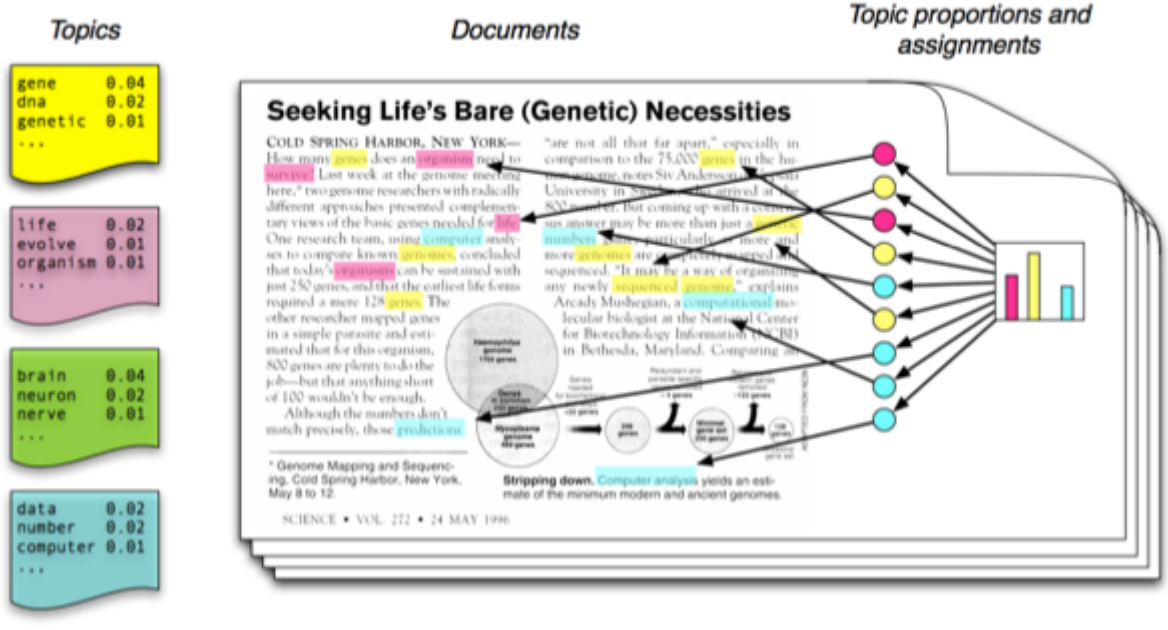


Figure source: Blei, D. M. (2012). Probabilistic topic models. *Communications of the ACM*, *55*(4), 77-84.

Appendix: Implementations

Running a Word2Vec model using Gensim

```
################################################
# Train the Word2Vec Model using Gensim.       #
################################################

### Import modules ###
import gensim
import gensim.parsing.preprocessing
from gensim.parsing.preprocessing import preprocess_string

### APPLY Your Favorite Preprocessing + Tokenization + PostProcessing Scheme ###
filters = gensim.parsing.preprocessing.DEFAULT_FILTERS
filters = [i for i in filters[0:2]+filters[4:] if not i.__name__ =='strip_short']
corpus = [preprocess_string(text,filters) for text in df.text.values]
### Import model and logging function ###
import logging
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.I
NFO)
from gensim.models.word2vec import Word2Vec
### Define parameters for the model ###
size=300 # Size of Embedding.
workers=4 # Number CPU Cores to use for training in Parallel.
iter_=10 # Depending on the size of your data you might want to run through your data m
ore than once.
window=6 # How much Context
min_count=5 # Number of Occurrences to be kept in the Vocabulary
### Initialize model and start training ###
model = Word2Vec(corpus,size=size,workers=workers,iter=iter_,window=window,min_count=mi
n_count)
```

Big Data implementation: Streaming data instead of keeping it in memory.

```
class DocumentStream(object):
    # Defines a class for streaming data from disk.
    # This specific one reads lines from a list of files, so this should adopted to you
r data(-base) setup.
    def __init__(self, files):
        #self.filename = filename
        self.files = files
    def __iter__(self):
        for filename in self.files:
            with codecs.open(filename,'r','utf-8') as f:
                for line in f:
                    yield preprocess_string(line,filters) # or apply your favorite prep
roccesing scheme
path2files = '/'
from os import listdir
files = [path2files+filename for filename in listdir(files)]
corpus = DocumentStream(files)
model = Word2Vec(corpus,size=size,workers=workers,iter=iter_,window=window,min_count=mi
n_count)
```

Running a Topic Model using Gensim

```
################################################
# Run latent dirichlet alloaction using Gensim. #
################################################

### Import modules ###
import gensim
import gensim.parsing.preprocessing
from gensim.parsing.preprocessing import preprocess_string

### APPLY Your Favorite Preprocessing + Tokenization + PostProcessing Scheme ###
filters = gensim.parsing.preprocessing.DEFAULT_FILTERS
filters = [i for i in filters[0:2]+filters[4:] if not i.__name__ =='strip_short']
processed_docs = [preprocess_string(text,filters) for text in df.text.values]
### Create Index using gensims own method
dictionary = gensim.corpora.Dictionary(processed_docs)
### OPTIONAL: Filter Number of Dimensions ###
dictionary.filter_extremes(no_below=15, no_above=0.5, keep_n=100000)
### Convert Tokenized Documents to BoWs using gensims own method ###
bow_corpus = [dictionary.doc2bow(doc) for doc in processed_docs]
### Run model with K topics and define number of Cores for multiprocessing ###
n_cores = 8
k = 50
lda_model = gensim.models.LdaMulticore(bow_corpus, num_topics=k, id2word=dictionary, pa
sses=2, workers=n_cores)
### WAIT ###
```

Simple Counting Operations