Popular Topic Mining from Blogs

Stone Fang (Student ID: 19049045)
Computers and Information Sciences
Auckland University of Technology
Auckland, New Zealand
fnk7060@autuni.ac.nz

I. Metrics

xx

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import sys
import os.path
from glob import glob
from tqdm import tqdm
import pickle

import itertools
from collections import namedtuple
import pandas as pd
from xml.etree import ElementTree
from xml.etree.ElementTree import ParseError
from bs4 import BeautifulSoup
from datetime import datetime

import nltk
from nltk.corpus import stopwords
sw = stopwords.words("english")

################################################################################
#              Codes for data reading & transformation                         #
################################################################################

Record = namedtuple('Record', ['meta', 'posts'])
Post = namedtuple('Post', ['date', 'text'])
MetaData = namedtuple('MetaData', ['id', 'gender', 'age', 'category', 'zodiac'])

def parse_meta_data(meta_data_str):
    arr = meta_data_str.strip().split('.')
    return MetaData(arr[0], arr[1], int(arr[2]), arr[3], arr[4])

# _parser = ElementTree.XMLParser(encoding="utf-8")
def read_blog_file(fpath):
    try:
        # tree = ElementTree.parse(fpath, parser=_parser)
        with open(fpath, encoding='utf-8', errors='ignore') as f:
            soup = BeautifulSoup(f.read(), "xml")
        blog = soup.Blog
    except ParseError:
        print('Error: invalid xml file {}'.format(fpath))
        raise
        return []

    posts = []
    state = 'date'
    for c in blog.find_all(recursive=False):
        # print(c)
        # print(c.text)
        # check the <date> and <post> tags appear alternately
        if c.name != state:
            print('Warning: inconsistent format in file {}'.format(fpath))
        if state == 'date':
            try:
                date_str = c.text.strip()
                # date_str = date_str.replace('janvier', 'january') \
                #     .replace('mars', 'march') \
                #     .replace('avril', 'april') \
                #     .replace('mai', 'may') \
                #     .replace('juin', 'june') \
                #     .replace('juillet', 'july')
                # date = pd.to_datetime(date_str, format='%d,%B,%Y')
                date = date_str
            except ValueError:
```

```
                print('Warning: invalid date {} in file {}' \
                        .format(c.text, fpath))
                date = datetime.fromtimestamp(0)
            state = 'post'
        else:
            text = c.text.strip()
            state = 'date'
            posts.append(Post(date, text))
        # print(c, c.text)
    posts.sort(key=lambda p: p.date)
    # print(posts)
    # print(pd.DataFrame(posts))
    # sys.exit()
    return posts

def read_blogs(path, force=False, cache_file='blogs.pkl'):
    if not force and os.path.exists(cache_file):
        print('load dataset from cached pickle file ' + cache_file)
        with open(cache_file, 'rb') as f:
            dataset = pickle.load(f)
        return dataset

    dataset = read_blogs_xml(path)

    # save to pickle file for fast loading next time
    with open(cache_file, 'wb') as f:
        print('save dataset to pickle file ' + cache_file)
        pickle.dump(dataset, f)

    return dataset

def read_blogs_xml(path):
    print('reading all data files from directory {} ...'.format(path))
    dataset = []
    # for fpath in tqdm(glob(os.path.join(path, '*'))):
    for fpath in list(glob(os.path.join(path, '*')))[:3]:
        # print(fpath)
        fname = os.path.basename(fpath)
        meta_data = parse_meta_data(fname)
        # print(meta_data)
        posts = read_blog_file(fpath)
        rec = Record(meta_data, posts)
        dataset.append(rec)
    return dataset

def show_summary(dataset):
    '''This function describes the summary of dataset or human inspection.
    It's not necessary for the mining process.

    Parameters
    --------------
    dataset : list of Record
        The blog dataset
    '''

    df = pd.DataFrame([d.meta for d in dataset])
    df['blog_count'] = [len(d.posts) for d in dataset]
    # print(df)
    print(df.describe(include='all'))
    print('{} possible values for "gender": {}'.format(
            len(df.gender.unique()), ', '.join(sorted(df.gender.unique()))))
    # print('{} possible values for "{}": {}'.format(
    #         len(df.age.unique()), ', '.join(sorted(df.age.unique()))))
    print('{} possible values for category: {}'.format(
            len(df.category.unique()), ', '.join(sorted(df.category.unique()))))
    print('{} possible values for zodiac: {}'.format(
            len(df.zodiac.unique()), ', '.join(sorted(df.zodiac.unique()))))
```

```
133
134 ##############################################################################
135 #              Codes for topic mining                                        #
136 ##############################################################################
137
138 def tokenise(dataset):
139     '''
140     consider all the blogs from one person as a document
141
142     Returns
143     ---------
144     docs: list of list of list
145         a list of documents, each of which is a list of sentences,
146         each of which is a list of words.
147     '''
148
149     print('tokenising the text dataset...')
150     docs = []
151     for rec in dataset:
152         doc = []
153         for post in rec.posts:
154             # print(post)
155             for sent in nltk.sent_tokenize(post.text):
156                 doc.append([w.lower() for w in nltk.word_tokenize(sent)])
157             # print(doc)
158         docs.append(doc)
159         # print(doc)
160         # return(docs)
161     return docs
162
163 def flatten2d(list2d):
164     return itertools.chain.from_iterable(list2d)
165
166 def flatten3d(list3d):
167     return itertools.chain.from_iterable(flatten2d(list3d))
168
169 def map2d(f, docs):
170     for doc in docs:
171         yield [f(sent) for sent in doc]
172
173 def map3d(f, docs):
174     for doc in docs:
175         yield [[f(word) for word in sent] for sent in doc]
176
177 def get_things(docs):
178     things = filter(lambda wp: wp[1] == 'NN', flatten3d(docs))
179     tf = nltk.FreqDist(things)
180     print(tf.most_common(50))
181
182 #TODO: remove stop words
183 def count_word(dataset):
184     docs = tokenise(dataset)
185
186     print('POS tagging...')
187     tagged_docs = list(map2d(lambda s: nltk.pos_tag(s), docs))
188     print(tagged_docs[0][0])
189     # print(sorted(set(p for w, p in flatten3d(tagged_docs))))
190
191     things = get_things(tagged_docs)
192
193     print('counting word frequencies...')
194     tf = nltk.FreqDist(flatten3d(tagged_docs))
195     print(tf.most_common(50))
196
197 def main():
198     # read_blogs('blogs')
199     # read_blogs('blogs', force=True, cache_file='blogs-10.pkl')
```

```
    # dataset = read_blogs('.', cache_file='blogs-10.pkl')
    dataset = read_blogs('.', cache_file='blogs-3.pkl')

    count_word(dataset)
    return

if __name__ == '__main__':
    main()
```