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

APPENDIX
SOURCE CODE IN PYTHON

```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")


##############################################################################
#                          Utility functions                                 #
##############################################################################

def flatten2d(list2d):
    return itertools.chain.from_iterable(list2d)

def flatten3d(list3d):
    return itertools.chain.from_iterable(flatten2d(list3d))

def map2d(f, docs):
    for doc in docs:
        yield [f(sent) for sent in doc]

def map3d(f, docs):
    for doc in docs:
        yield [[f(word) for word in sent] for sent in doc]

##############################################################################
#              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()))))

################################################################################
#               Codes for topic mining                                         #
################################################################################

def tokenise(dataset):
    '''
    consider all the blogs from one person as a document

    Returns
    ---------
    docs: list of list of list
        a list of documents, each of which is a list of sentences,
        each of which is a list of words.
    '''

    print('tokenising the text dataset...')
    docs = []
    for rec in dataset:
        doc = []
        for post in rec.posts:
            # print(post)
            for sent in nltk.sent_tokenize(post.text):
                doc.append([w.lower() for w in nltk.word_tokenize(sent)])
            # print(doc)
        docs.append(doc)
        # print(doc)
        # return(docs)
    return docs

def get_things(docs):
    things = filter(lambda wp: wp[1] == 'NN', flatten3d(docs))
    tf = nltk.FreqDist(things)
    print(tf.most_common(50))

#TODO: remove stop words
def count_word(dataset):
    docs = tokenise(dataset)

    print('POS tagging...')
    tagged_docs = list(map2d(lambda s: nltk.pos_tag(s), docs))
    print(tagged_docs[0][0])
    # print(sorted(set(p for w, p in flatten3d(tagged_docs))))

    things = get_things(tagged_docs)

    print('counting word frequencies...')
```

```
    tf = nltk.FreqDist(flatten3d(tagged_docs))
    print(tf.most_common(50))

def main():
    # read_blogs('blogs')
    # 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()
```