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, Counter
import pandas as pd
from bs4 import BeautifulSoup

import nltk
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer

_DEBUG = True

sw = stopwords.words("english")


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

def len2d(iter2d):
    return sum(len(d) for d in iter2d)

def list2d(iter2d):
    return [[x for x in inner] for inner in iter2d]

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

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

def mapbar(f, seq, desc):
    for e in tqdm(seq, desc):
        yield f(e)

def map2d(f, docs):
    with tqdm(total=len2d(docs)) as pbar:
        def _helper(sent):
            pbar.update(1)
            return f(sent)

        for doc in docs:
            yield map(_helper, doc)

def map3d(f, docs):
    with tqdm(total=len2d(docs)) as pbar:
        def _helper(sent):
            pbar.update(1)
            return [f(word) for word in sent]

        for doc in docs:
            yield map(_helper, doc)

def foreach3d(f, docs):
    with tqdm(total=len2d(docs)) as pbar:
        for doc in docs:
```

```
            for sent in doc:
                for word in sent:
                    f(word)
                pbar.update(1)

def foreach2d(f, docs):
    with tqdm(total=len2d(docs)) as pbar:
        for doc in docs:
            for sent in doc:
                f(sent)
                pbar.update(1)

################################################################################
#               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])

def read_blog_file(fpath):
    try:
        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):
        if c.name != state:
            print('Warning: inconsistent format in file {}'.format(fpath))
        if state == 'date':
            try:
                date_str = c.text.strip()
                date = date_str
            except ValueError:
                print('Warning: invalid date {} in file {}' \
                        .format(c.text, fpath))
            state = 'post'
        else:
            text = c.text.strip()
            state = 'date'
            posts.append(Post(date, text))
    posts.sort(key=lambda p: p.date)
    return posts

def read_blogs(path, force=False, cache_file='blogs.pkl'):
    if not force and cache_file is not None 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
    if cache_file is not None:
        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 = []

    if _DEBUG:  # use small files for fast debugging
        files = [os.path.join(path, fname) for fname in ['3998465.male.17.indUnk.Gemini.xml',
            '3949642.male.25.indUnk.Leo.xml', '3924311.male.27.HumanResources.Gemini.xml']]
    else:
        files = glob(os.path.join(path, '*'))

    for fpath in  tqdm(files):
        fname = os.path.basename(fpath)
        meta_data = parse_meta_data(fname)
        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.describe(include='all'))
    print('{} possible values for "gender": {}'.format(
            len(df.gender.unique()), ', '.join(sorted(df.gender.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 = []
    vocab = set()
    with tqdm(total=sum(len(rec.posts) for rec in dataset)) as pbar:
        for rec in dataset:
            doc = []
            for post in rec.posts:
                for sent_str in nltk.sent_tokenize(post.text):
                    sent = [w.lower() for w in nltk.word_tokenize(sent_str)]
                    doc.append(sent)
```

```
                    vocab.update(sent)
                pbar.update(1)
            docs.append(doc)
    return sorted(vocab), docs

def get_things(docs, n=5):
    things = filter(lambda wp: wp[1] == 'NN', flatten3d(docs))
    tf = nltk.FreqDist(things)
    return tf.most_common(n)


def get_surroundings(words, docs, n=2):
    '''expand the topic to be 2 verb/noun before and 2 verb/noun after the topic
    '''

    print('get surrounding {} nouns/verbs for words {}'.format(n, words))

    sur = {w: Counter() for w in words}

    target_pos_tags = ('NN', 'NNS', 'VB', 'VBP', 'VBD', 'VBN')

    def _helper(sent):
        for w in words:
            try:
                idx = sent.index(w)
            except ValueError:
                continue

            after = 0
            for (wi, pi) in sent[(idx+1):]:
                if pi in target_pos_tags:
                    sur[w][(wi, pi)] += 1
                    after += 1
                if after == n:
                    break

            before = 0
            for (wi, pi) in reversed(sent[:idx]):
                if pi in target_pos_tags:
                    sur[w][(wi, pi)] += 1
                    before += 1
                if before == n:
                    break

    foreach2d(_helper, docs)
    return sur

def mine_topic_by_freq(dataset):
    vocab, docs = tokenise(dataset)
    print('Size of vocabulary: {}'.format(len(vocab)))
    print(vocab[:100])

    print('POS tagging...')
    tagged_docs = list2d(map2d(lambda s: nltk.pos_tag(s), docs))

    things = get_things(tagged_docs)
    print('things: ', things)


    thing_words = [(w, pos) for ((w, pos), c) in things]
    keywords = get_surroundings(thing_words, tagged_docs, n=2)
    print(keywords)


def main():
    if _DEBUG:
        dataset = read_blogs('blogs', cache_file=None)
```

```
266     else:
267         dataset = read_blogs('blogs')
268 
269     mine_topic_by_freq(dataset)
270     return
271 
272 if __name__ == '__main__':
273     main()
```