Popular topics analysis

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

I. Metrics

xx

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

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

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

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

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, '*')))[:10]:
        # 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):
    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()))))

def main():
    read_blogs('blogs')
    return

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