```python
#!/usr/bin/python
# -*- coding: utf-8 -*-
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.tokenize import RegexpTokenizer
from nltk.tokenize import sent_tokenize

import xml.etree.ElementTree as ET

import json

import os
import glob
import string
import re

##############################################################################

stop_words = set(stopwords.words('english'))

if not os.path.exists('Dataset_original'):
    os.makedirs('Dataset_original')
originalDS = 'Dataset_original/'

if not os.path.exists('Dataset_as_txt'):
    os.makedirs('Dataset_as_txt')
asTextDS = 'Dataset_as_txt/'

if not os.path.exists('Dataset_preprocessed'):
    os.makedirs('Dataset_preprocessed')
preprocDS = 'Dataset_preprocessed/'

if not os.path.exists('Keyphrases_golden_set'):
    os.makedirs('Keyphrases_golden_set')
goldenSet = 'Keyphrases_golden_set/'

if not os.path.exists('Keyphrases_experimental'):
    os.makedirs('Keyphrases_experimental')
exprSet = 'Keyphrases_experimental/'

if not os.path.exists('Metrics'):
    os.makedirs('Metrics')
metricsF = 'Metrics/'

##############################################################################
#
# XML/TREE OPERATIONS
#
##############################################################################

def get_root_from_xml(file):
    return ET.parse(file).getroot()


def from_root_get_document(root):
    return list(root)[0]


def from_document_get_sentences(doc):
    return list(doc)[0]


def from_sentences_get_nth_sentence(sentences, n):
    return list(sentences)[n]


def from_sentence_get_tokens(sentence):
    return list(sentence)[0]


def from_tokens_get_full_sentence(tokens, n):
    sentence = []
    for i in tokens:
        sentence.append(i[0].text)
    return sentence


##############################################################################
def from_xml_to_txt():
    for filename in glob.glob(originalDS + '*.xml'):
        with open(filename, 'r', encoding='utf-8') as document:
            noPath = os.path.basename(filename)
            asTxt = os.path.splitext(noPath)[0] + '.txt'

            open(asTextDS + asTxt, 'w', encoding='utf-8').close()
            with open(asTextDS + asTxt, 'a+', encoding='utf-8') as out:
                root = get_root_from_xml(document)
                doc = from_root_get_document(root)
                sentences = from_document_get_sentences(doc)
                for i in range(len(sentences)):
                    sentence = \
                        from_sentences_get_nth_sentence(sentences, i)
                    tokens = from_sentence_get_tokens(sentence)
                    for j in range(len(tokens)):
                        full_sentence = \
                            from_tokens_get_full_sentence(tokens, j)
                    for u in full_sentence:
                        if u != '.':
                            out.write(u + ' ')
                        else:
                            out.write(u + '\n')

def preproc_all_txt():
    for filename in glob.glob(asTextDS + '*.txt'):
        noPath = os.path.basename(filename)
        out = preprocDS + noPath

        preprocess_file(filename, out)

def parse_json():
    with open("test.reader.json", 'r', encoding='utf-8') as fjs:
        d = json.load(fjs)
        for key, value in d.items():
            open(goldenSet + key + ".txt", 'w', encoding='utf-8').close()
            with open(goldenSet + key + ".txt", 'a+', encoding='utf-8') as gkps:
```

```
                for i in value:
                    if len([word for word in i[0].split()]) in [1, 2, 3]:
                        gkps.write(i[0] + "\n")

###############################################################################
def preprocess_file(docname, outfile_name):
    with open(docname, 'r', encoding='utf-8') as file:
        outfile = open(outfile_name, 'w', encoding='utf-8')

        for line in file:
            print(preprocess_sentence(line), end='', file=outfile)
        outfile.close()

    return outfile_name


def preprocess_list(inList, outfile_name):
    # only run this once for 20newsgroup to save the preprocessed DS in memory
    i = 1
    for article in inList:
        sentences = sent_tokenize(article)
        outfile = open(outfile_name + str(i) + '.txt', 'w',
                       encoding='utf-8')
        for sentence in sentences:
            preprocessed_sentence = preprocess_sentence(sentence)
            print(preprocessed_sentence, end='', file=outfile)
        outfile.close()
        i += 1
    return (outfile_name, i)


def preprocess_sentence(sentence):
    processed = sentence.lower()
    processed = re.sub(r'\d+', '', processed)
    tokenizer = RegexpTokenizer(r'\w+')
    tokens = tokenizer.tokenize(processed)
    #filtered_words = [w for w in tokens if not w in stop_words]
    filtered_words = [w for w in tokens]
    if sentence[-1] == '\n':
        return ' '.join(filtered_words) + ' '
    return ' '.join(filtered_words)

###############################################################################
def ngrams(docname, low, high):
    with open(docname, 'r', encoding='utf-8') as document:
        result = []

        c_vec = CountVectorizer(ngram_range=(low, high))
        ngrams = c_vec.fit_transform(document)
        vocab = c_vec.vocabulary_
        count_values = ngrams.toarray().sum(axis=0)

        for ngram in sorted([[count_values[i], k] for (k, i) in vocab.items()],
reverse=True):
            result.append(ngram)

    return result


def idf(dataset, low, high):
    # assuming dataset is already a list (like train.data)

    vectorizer = TfidfVectorizer(strip_accents='unicode', ngram_range=(low, high))

    vectorizer.fit_transform(dataset)

    idf = vectorizer.idf_

    return dict(zip(vectorizer.get_feature_names(), idf))

###############################################################################

def rates(gs, exp):
    tp = 0
    fp = 0
    fn = 0
    tp_at_5 = 0

    for i in exp:
        if i in gs:
            tp += 1
        if i not in gs:
            fp += 1

    for i in gs:
        if i not in exp:
            fn += 1

    for i in exp[:5]:
        if i in gs:
            tp_at_5 += 1

    return tp, fp, fn, tp_at_5

def precision(tp, fp):
    return 0 if tp + fp == 0 else tp/(tp+fp)

def recall(tp, fn):
    return 0 if tp + fn == 0 else tp/(tp+fn)

def f_one(r, p):
    return 0 if r + p == 0 else 2 * r * p / (r + p)

def precision_at_n(tp_at_n, n):
    return tp_at_n/n

def mean_average_precision(golden, experimental):
    precisions = []
    for i in range(1,len(experimental)+1):
        tp, fp, _, _ = rates(golden, experimental[:i])
        precisions.append(precision(tp, fp))
    meavp = sum(precisions)/len(precisions)
    return meavp

def metrics(golden, experimental):
    with open(golden, 'r', encoding = 'utf-8') as gs_file, open(experimental, 'r',
encoding = 'utf-8') as exp_file:
        gs = []
        exp = []
        for line in gs_file:
```

```
            gs.append(line)
        for line in exp_file:
            exp.append(line)

        tp, fp, fn, p_at_5 = rates(gs, exp)

        prec = precision(tp, fp)

        rec = recall(tp, fn)

        p_at_5 = p_at_5/5

        f1 = f_one(rec, prec)

        mav = mean_average_precision(gs, exp)

        file = metricsF + os.path.basename(golden)
        s1 = "Precision: " + str(prec) + "\n" + "Recall: " + str(rec) + "\n" + "F1: "
+ str(f1) + "\n" + "P@5: " + str(p_at_5) + "\n" + "MAV: " + str(mav)

        open(file, 'w', encoding = 'utf-8').close()
        with open(file, 'w', encoding = 'utf-8') as f:
            f.write(s1)


    return prec, rec, f1, p_at_5, mav



################################################################################
class Dataset:
    # models a dataset
    def __init__(self, directory):
        self.directory = directory
        self.asList = list()
        self.idf = None

    def from_files_to_list(self):
        for filename in glob.glob(self.directory + '*.txt'):
            with open(filename, 'r', encoding='utf-8') as document:  # utf-8 is
breaking at (spanish n)
                string = document.read()
            self.asList = self.asList + [string]

    def compute_idf(self, low, high):
        self.idf = idf(self.asList, low, high)



class Document:
    # models a processed doc
    def __init__(self, filename):
        self.file = filename
        self.ngrams = None  # candidates
        self.keyphrases = list()

    def compute_ngrams(self, low, high):
        self.ngrams = ngrams(self.file, low, high)
```

```
    def get_keyphrases(self, ds_idf):
        for phrase in self.ngrams:
            tf = phrase[0]
            idf = ds_idf[phrase[1]]
            tfidf_with_len = tf * idf * len(phrase[1].split())
            self.keyphrases.append((phrase[1], tfidf_with_len))

    # Class helpers
    def write_keyphrases(self, n):
        keyphrases = sorted(self.keyphrases, key=lambda x: x[1], reverse=True)[:n]
        open(exprSet + os.path.basename(self.file), 'w', encoding = 'utf-8').close()
        with open(exprSet + os.path.basename(self.file), 'a+', encoding = 'utf-8') as
out:
            for i in keyphrases:
                out.write(i[0] + '\n')


    def get_preprocessed_text(self):
        with open(self.file, 'r', encoding='utf-8') as document:
            string = document.read()
        return string

################################################################################

def main():
    #test.reader.json and dataset_original needed, others are created
    #from_xml_to_txt() #only #1 time
    #preproc_all_txt() #only #1 time
    #parse_json() #only #1 time

    ds = Dataset(preprocDS)
    ds.from_files_to_list()
    ds.compute_idf(1, 3)
    for file in glob.glob(preprocDS + '*.txt'):
        doc = Document(file)
        doc.compute_ngrams(1, 3)
        doc.get_keyphrases(ds.idf)
        doc.write_keyphrases(10)

    precision_sum = 0
    recall_sum = 0
    f1_sum = 0
    p_at_5_sum = 0
    mav_sum = 0
    total_files = 0

    for file in glob.glob(goldenSet + '*.txt'):
        total_files += 1
        p, r , f1, p_at_5, mav = metrics(file, exprSet + os.path.basename(file))
        precision_sum += p
        recall_sum += r
        f1_sum += f1
        p_at_5_sum += p_at_5
        mav_sum += mav


    print("Average Precision: " + str(precision_sum/total_files))
    print("Average Recall: " + str(recall_sum/total_files))
    print("Average F1: " + str(f1_sum/total_files))
    print("Average P@5: " + str(p_at_5_sum/total_files))
```

```
    print("Average MAV: " + str(mav_sum/total_files))


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