```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

from math import log

import xml.etree.ElementTree as ET

import json

import os
import glob
import string
import re
import itertools, nltk, string

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

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 select_candidates(text):
    grammar = r'KT: {(<JJ>* <NN.*>+ <IN>)? <JJ>* <NN.*>+}'

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

    chunker = nltk.chunk.regexp.RegexpParser(grammar)
    tagged_sents = nltk.pos_tag_sents(nltk.word_tokenize(sent) for sent in
nltk.sent_tokenize(text))

    all_chunks =
list(itertools.chain.from_iterable(nltk.chunk.tree2conlltags(chunker.parse(tagged_sen
t))
                                                        for tagged_sent in tagged_sents))

    candidates = [' '.join(word for word, pos, chunk in group).lower()
                  for key, group in itertools.groupby(all_chunks, lambda
wordposchunk: wordposchunk[2] != 'O') if key]

    result = [cand for cand in candidates
            if cand not in stop_words and len(cand.split()) < 4]

    result = list(dict.fromkeys(result))

    #if len(result) < 10:
        #grammar = r'KT: {<DT>? <JJ>* (<NN>|<NP>|<PRN>)+}'

        #chunker = nltk.chunk.regexp.RegexpParser(grammar)

        #all_chunks =
list(itertools.chain.from_iterable(nltk.chunk.tree2conlltags(chunker.parse(tagged_sen
t))
                                                                #for tagged_sent in
tagged_sents))
        #candidates = [' '.join(word for word, pos, chunk in group).lower()
                      #for key, group in itertools.groupby(all_chunks, lambda
wordposchunk: wordposchunk[2] != 'O') if key]

        #result = [cand for cand in candidates
                #if cand not in stop_words and len(cand.split()) < 4 and not all(char
in punct for char in cand)]

    return result

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 n_t(list1, list2):
    c=[]
    for i in range(0,len(list1)):
        n=0
        for j in range(0,len(list2)):
            if (list1[i] in list2[j]) == True:
                n=n+1
        c.append((n,list1[i]))
    return c

def frequency(list1, doc1):
    c=[]
    for i in range(0,len(list1)):
        c.append(doc1.count(list1[i]))
    return c
```

```
231
232 def avgdl(list1):
233     s=0
234     for i in range(0,len(list1)):
235         s = s+len(list1[i])
236     total = s/len(list1)
237     return total
238
239 def bm25(candidates, dataset, doc):
240     N = len(dataset)
241     D = len(doc)
242     k1 = 1.2
243     b = 0.75
244     avgdl1 = avgdl(dataset)
245     frequency1 = frequency(candidates, doc)
246     n_t1 = n_t(candidates, dataset)
247     final = dict()
248     for i in range(0,len(candidates)):
249         score = log((N - n_t1[i][0] + 0.5)/(n_t1[i][0] + 0.5))*((frequency1[i]*
(k1+1))/(frequency1[i]+(k1*(1-b+(b*(D/avgdl1))))))
250         final[candidates[i]] = score
251     return final
252
253 ##############################################################################
254 def rates(gs, exp):
255     tp = 0
256     fp = 0
257     fn = 0
258     tp_at_5 = 0
259
260     for i in exp:
261         if i in gs:
262             tp += 1
263         if i not in gs:
264             fp += 1
265
266     for i in gs:
267         if i not in exp:
268             fn += 1
269
270     for i in exp[:5]:
271         if i in gs:
272             tp_at_5 += 1
273
274     return tp, fp, fn, tp_at_5
275
276 def precision(tp, fp):
277     return 0 if tp + fp == 0 else tp/(tp+fp)
278
279 def recall(tp, fn):
280     return 0 if tp + fn == 0 else tp/(tp+fn)
281
282 def f_one(r, p):
283     return 0 if r + p == 0 else 2 * r * p / (r + p)
284
285 def precision_at_n(tp_at_n, n):
286     return tp_at_n/n
287
288 def mean_average_precision(golden, experimental):
289     precisions = []
290     for i in range(1,len(experimental)+1):
291         tp, fp, _, _ = rates(golden, experimental[:i])
292         precisions.append(precision(tp, fp))
293     meavp = sum(precisions)/len(precisions)
294     return meavp
295
296 def metrics(golden, experimental):
297     with open(golden, 'r', encoding = 'utf-8') as gs_file, open(experimental, 'r',
encoding = 'utf-8') as exp_file:
298         gs = []
299         exp = []
300         for line in gs_file:
301             gs.append(line)
302         for line in exp_file:
303             exp.append(line)
304
305         tp, fp, fn, p_at_5 = rates(gs, exp)
306
307         prec = precision(tp, fp)
308
309         rec = recall(tp, fn)
310
311         p_at_5 = p_at_5/5
312
313         f1 = f_one(rec, prec)
314
315         mav = mean_average_precision(gs, exp)
316
317         file = metricsF + os.path.basename(golden)
318         s1 = "Precision: " + str(prec) + "\n" + "Recall: " + str(rec) + "\n" + "F1: "
+ str(f1) + "\n" + "P@5: " + str(p_at_5) + "\n" + "MAV: " + str(mav)
319
320         open(file, 'w', encoding = 'utf-8').close()
321         with open(file, 'w', encoding = 'utf-8') as f:
322             f.write(s1)
323
324
325     return prec, rec, f1, p_at_5, mav
326
327
328
329 ##############################################################################
330 class Dataset:
331     # models a dataset
332     def __init__(self, directory):
333         self.directory = directory
334         self.asList = list()
335         self.bm25 = None
336
337     def from_files_to_list(self):
338         for filename in glob.glob(self.directory + '*.txt'):
339             with open(filename, 'r', encoding='utf-8') as document:  # utf-8 is
breaking at (spanish n)
340                 string = document.read()
341             self.asList = self.asList + [string]
342
343 class Document:
344     # models a processed doc
345     def __init__(self, filename):
346         self.file = filename
```

```
        self.ngrams = None
        self.bm25 = None
        self.grammar_candidates = None
        self.keyphrases_idf = list()
        self.keyphrases_bm25 = list()

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

    def get_grammar_candidates(self):
        self.grammar_candidates = select_candidates(self.get_preprocessed_text())

    def get_keyphrases_bm25(self):
        #for phrase in self.ngrams:
        for phrase in self.grammar_candidates:
            score = self.bm25[phrase]
            #score_len = 0.5*score + 0.5*len(phrase.split())
            self.keyphrases_bm25.append((phrase, score))

    def compute_bm25(self, dataset):
        self.bm25 = bm25(self.grammar_candidates, dataset, self.get_preprocessed_text())


    def write_keyphrases_bm25(self, n):
        keyphrases = sorted(self.keyphrases_bm25, 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_ngrams_without_tf(self):
        ngrams = list()
        for i in self.ngrams:
            ngrams.append(i[1])
        return ngrams

    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()

    for file in glob.glob(preprocDS + '*.txt'):
        doc = Document(file)
        doc.compute_ngrams(1, 3)
        doc.get_grammar_candidates()
        doc.compute_bm25(ds.asList)
        doc.get_keyphrases_bm25()
        doc.write_keyphrases_bm25(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()
```