```
#!/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 os
import glob
import string
import re

#############################################################################
#some global vars such as file nanes and directories

preprocessedFile = "preprocessed.txt"


keyphrasesFile = "keyphrases.txt"

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

outFileDSBase = "Dataset/DS_preproc_"

inFile = "original.txt"

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

train = fetch_20newsgroups(subset='train', remove=('headers', 'footers',
'quotes')).data

test = fetch_20newsgroups(subset='test', remove=('headers', 'footers',
'quotes')).data

full_ds = train + test

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

def preprocess_file(docname, outfile_name):
    #given a file docname, preprocesses and saves it at file 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):
    #given a dataset like 20 newsgroup, preprocess it and save it on files.
    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):
    #preprocess a sentece: lowercase, remove numbers, stopwords, punctuation.
    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]
    if(sentence[-1] == '\n'): return  " ".join(filtered_words) + " "
    return " ".join(filtered_words)


#############################################################################
def ngrams(docname, low, high):
    #given a file, get ngrams from range low to 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, candidates):
    #given a dataset of files, compute the idf score for words in list candidates,
    #ngrams ranging from low to high.
    vectorizer = TfidfVectorizer(strip_accents='unicode', ngram_range=(low, high),
vocabulary = candidates)

    vectorizer.fit_transform(dataset)

    idf = vectorizer.idf_

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

#############################################################################
class Dataset:
    # models a dataset. assuming files already preprocessed
    def __init__(self, directory):
        self.directory = directory #directory of the files
        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:
```

```
                string = document.read()
            self.asList = self.asList + [string]

    def compute_idf(self, preprocessed_file, low, high, candidates):
        self.idf = idf(self.asList + [preprocessed_file], low, high, candidates)

    def get_idf(self):
        return self.idf


class Document:
    #models a document. assuming the document is already preprocessed.
    def __init__(self, filename):
        self.file = filename
        self.ngrams = None
        self.keyphrases = list()

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

    def find_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))

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

    def get_ngrams_without_tf(self):
        ngrams = list()
        for i in self.ngrams:
            ngrams.append(i[1])
        return ngrams

    def print_keyphrases(self, n):
        kp = sorted(self.keyphrases, key = lambda x: x[1], reverse = True)[:n]

        open(keyphrasesFile, 'w', encoding='utf-8').close()
        with open(keyphrasesFile, 'a+', encoding='utf-8') as file:
            for phrase in kp:
                file.write(phrase[0] + '\n')
        print(kp)


################################################################################
def main():
    #preprocess_list(full_ds, outFileDSBase) #run this only once
    #preprocess_file(inFile, preprocessedFile) #run this only once

    ds = Dataset(preprocDS)
    ds.from_files_to_list()

    doc = Document(preprocessedFile)
    doc.compute_ngrams(1, 3)
    ds.compute_idf(doc.get_preprocessed_text(), 1, 3, doc.get_ngrams_without_tf())

    doc.find_keyphrases(ds.get_idf())
    doc.print_keyphrases(5)

if __name__ == "__main__":
    main()


```