```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
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Perceptron

import numpy as np
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('Keyphrases_experimental'):
    os.makedirs('Keyphrases_experimental')
exprSet = 'Keyphrases_experimental/'

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

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

goldenSetTrain = 'Goldenset_train/'
goldenSetTest = 'Goldenset_test/'
trainSet = 'Train_Set_Preprocessed/'
testSet = 'Test_Set_Preprocessed/'

################################################################################
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):
            if ngram[1] not in stop_words:
                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 = 0
    if len(precisions) != 0:
        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()
        self.CandidatesAsVectors = None
        self.CandidatesResultVector = None
        self.grammar_candidates = None

    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))

    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 predict_perceptron(self, perceptron):
        for idx, candidate in enumerate(self.CandidatesAsVectors):

            a = perceptron.predict(candidate.reshape(1,-1))
            if a[0] == 1:
                self.keyphrases.append((self.ngrams[idx][1], a))

        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 self.keyphrases:
                out.write(i[0] + '\n')

    def vectorize_candidates(self, ds_idf, dataset):
        vecs = []
        res = []
        gs = []

        with open(dataset + os.path.basename(self.file), 'r', encoding='utf-8') as golden:
            for line in golden:
                gs.append(line)
        for i in self.ngrams:
            v = np.zeros(3)
            tf = i[0]
            idf = ds_idf[i[1]]
            tfidf_with_len = tf * idf * len(i[1].split())
            v[0] = tfidf_with_len
            v[1] = len(i[1].split())

            if i[1] in self.get_preprocessed_text():
                v[2] = self.get_preprocessed_text().index(i[1])
```

```
            else: v[2] = -1

            vecs.append(v)

            res.append(1) if i[1]+"\n" in gs else res.append(0)

        self.CandidatesAsVectors = vecs
        self.CandidatesResultVector = res


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

def main():
    ds = Dataset(preprocFullDS)
    ds.from_files_to_list()
    ds.compute_idf(1, 3)

    training_vectors = []
    labels_l = []

    for file in glob.glob(trainSet + '*.txt'):
        doc = Document(file)
        doc.compute_ngrams(1, 3)
        doc.vectorize_candidates(ds.idf, goldenSetTrain)
        training_vectors += doc.CandidatesAsVectors
        labels_l += doc.CandidatesResultVector

    labels = np.array(labels_l)
    arr = np.vstack(training_vectors)

    ppn = Perceptron(max_iter=10000, eta0=0.01, random_state=0)
    ppn.fit(arr, labels)

    for file in glob.glob(testSet + '*.txt'):
        doc = Document(file)
        doc.compute_ngrams(1, 3)
        doc.vectorize_candidates(ds.idf, goldenSetTest)
        doc.predict_perceptron(ppn)


    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(goldenSetTest + '*.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()
```