Sentence Recommender with Python

It is not uncommon for article writers to experience what is called writer's block. We have written one or two sentences and suddenly we do not have any idea what to say in the next sentence. The following program is trying to address this issue. It suggests next sentences to write based on the previous one. A collection of articles are needed for this purpose.

Here is the complete code:

```
import os
import nltk
import time

def get_filenames():
    for dir_path, dir_names, file_names in os.walk('textfiles'):
        for file_name in file_names:
            yield open(os.path.join(dir_path,file_name),'r',encoding='utf-8')

def get_text(file_path):
    for file in file_path:
        with open(file.name,'r') as f:
            yield f.read()
            f.close()

def get_sentences(text):
    sentences = nltk.sent_tokenize(text)
    for s in sentences:
        s = sentence_cleaning(s)
```

```
        yield s

def sentence_cleaning(sentence):
    mark_list = ['[',']','!',''','#','$','%','%26;','\'','(',')','*','+',',','-','.',
                 '/',':',';','<','=','>','?','@','[','\','^','_','\`','{','}','|',
                 '~']
    sentence = sentence.lower()
    sentence = sentence.strip()
    sentence = ''.join(c for c in sentence if c not in mark_list)
    return sentence

#vectorizing + cosine similarity
def calculate_score(set1,set2):
    rvector = set1.union(set2)
    list1 = []
    list2 = []
    for w in rvector:
        if w in set1:
            list1.append(1)
        else:
            list1.append(0)
        if w in set2:
            list2.append(1)
        else:
            list2.append(0)
    #cosine similarity
    c = 0
    for i in range(len(rvector)):
        c += list1[i] * list2[i]
    score = c / float((sum(list1) * sum(list2)) ** 0.5)
    return score

def similarity_score(sentence1,sentence2):
    list1 = nltk.word_tokenize(sentence1)
    list2 = nltk.word_tokenize(sentence2)
    set1 = {w for w in list1}
    set2 = {w for w in list2}
    return calculate_score(set1,set2)

def get_idx_similar(sentence_input):      #get the index of similar sentences
    idx_dict = {}
    sentence_input = sentence_cleaning(sentence_input)
    file_path = get_filenames()
    text = get_text(file_path)
    text_idx = 0
    for t in text:
        sentence_list = get_sentences(t)
        sentence_idx = 0
        for sentence in sentence_list:
            score = similarity_score(sentence_input,sentence)
            key = '{}-{}'.format(str(text_idx),str(sentence_idx))
            idx_dict[key] = score
            sentence_idx += 1
        text_idx += 1
    return idx_dict

def get_next_sentences(idx_dict):
    idx_tuples = sorted(idx_dict.items(), key=lambda x: x[1])
    next_dict = {}
    file_path = get_filenames()
    text = get_text(file_path)
    text_idx = 0
    for t in text:
        sentence_list = get_sentences(t)
        sentence_idx = 0
        for sentence in sentence_list:
            for elem in idx_tuples:
                elem1,elem2 = elem[0].split('-')
                elem2 = int(elem2) + 1
                if int(elem1) == text_idx and int(elem2) == sentence_idx:
                    next_dict[sentence] = elem[1]
```

```
            sentence_idx += 1
        text_idx += 1
        time.sleep(0.2)         #so the program does not consume too much CPU.
    return next_dict

sentence_input = 'Machine learning algorithm can only read numerical values.'

idx_dict = get_idx_similar(sentence_input)
next_dict = get_next_sentences(idx_dict)
next_tuples = sorted(next_dict.items(), reverse=True, key=lambda x: x[1])

counter = 1
for elem in next_tuples:
    print(str(counter)+' : '+elem[0])
    if counter == 10:
        break
    counter += 1
```

Explanations

```
def get_filenames():
    for dir_path, dir_names, file_names in os.walk('textfiles'):
        for file_name in file_names:
            yield open(os.path.join(dir_path,file_name),'r',encoding='utf-8')
```

As mentioned before, a collection of articles are needed for this program to work. Collect a number of articles and store them in a directory. In the above code, you can see that a generator is used to collect the filenames of all of the articles in the directory.

```
def get_text(file_path):
    for file in file_path:
        with open(file.name,'r') as f:
            yield f.read()
            f.close()
```

In the above code, a generator is used to grab the content of the articles. The object of the previous generator *(get_filenames)* is passed to it.

```
def get_sentences(text):
    sentences = nltk.sent_tokenize(text)
    for s in sentences:
        s = sentence_cleaning(s)
        yield s
```

After you get the content of the articles, each of them are turned into list of sentences using **nltk** module. Note that with the module, capitalization is taken into account to determine next sentence.

```
def calculate_score(set1,set2):
    rvector = set1.union(set2)
    list1 = []
    list2 = []
    for w in rvector:
        if w in set1:
            list1.append(1)
        else:
            list1.append(0)
        if w in set2:
            list2.append(1)
        else:
            list2.append(0)
    #cosine similarity
    c = 0
    for i in range(len(rvector)):
        c += list1[i] * list2[i]
    score = c / float((sum(list1) * sum(list2)) ** 0.5)
    return score
```

The above function produces similarity score. It turns two sentences, your input sentence and one sentence in the articles, into vectors. Each element in the vector is a binary value (1 or 0). All the words in the both sentences are first inserted into a variable *(rvector)*. Then based on whether or not the corresponding word appeared in the variable, the vectors for the two sentences are created. Cosine similarity function then calculates the similarity score. The bigger the number (score), the more similar are the sentences.

```
def get_idx_similar(sentence_input):      #get the index of similar sentences
    idx_dict = {}
    sentence_input = sentence_cleaning(sentence_input)
    file_path = get_filenames()
    text = get_text(file_path)
    text_idx = 0
    for t in text:
        sentence_list = get_sentences(t)
        sentence_idx = 0
        for sentence in sentence_list:
            score = similarity_score(sentence_input,sentence)
            key = '{}-{}'.format(str(text_idx),str(sentence_idx))
            idx_dict[key] = score
            sentence_idx += 1
        text_idx += 1
    return idx_dict
```

In the above function, a dictionary *(idx_dict)* is created. It stores the index and the similarity score of each sentence in the articles. The index is created based on the order of the corresponding article in the directory *(text_idx)* and the position of the sentence in the article *(sentence_idx)*. This function also calls another function *(sentence_cleaning)* to remove unwanted characters and convert all the words to lowercase.

```
def get_next_sentences(idx_dict):
    idx_tuples = sorted(idx_dict.items(), key=lambda x: x[1])
    next_dict = {}
    file_path = get_filenames()
    text = get_text(file_path)
    text_idx = 0
    for t in text:
        sentence_list = get_sentences(t)
        sentence_idx = 0
        for sentence in sentence_list:
            for elem in idx_tuples:
                elem1,elem2 = elem[0].split('-')
                elem2 = int(elem2) + 1
                if int(elem1) == text_idx and int(elem2) == sentence_idx:
                    next_dict[sentence] = elem[1]
            sentence_idx += 1
        text_idx += 1
        time.sleep(0.2)         #so the program does not consume too much CPU.
    return next_dict
```

The *get_next_sentences* is the function that grab the next sentences. The results are stored in the *next_dict* dictionary. In the function, the *idx_dict* dictionary is first sorted based on the similarity score. The results are stored in the tuple *idx_tuples*.

```
sentence_input = 'Machine learning algorithm can only read numerical values.'
```

The variable *sentence_input*, as the name implies, stores your input or the last sentence you have written in your unfinished article. Change the value as you like.

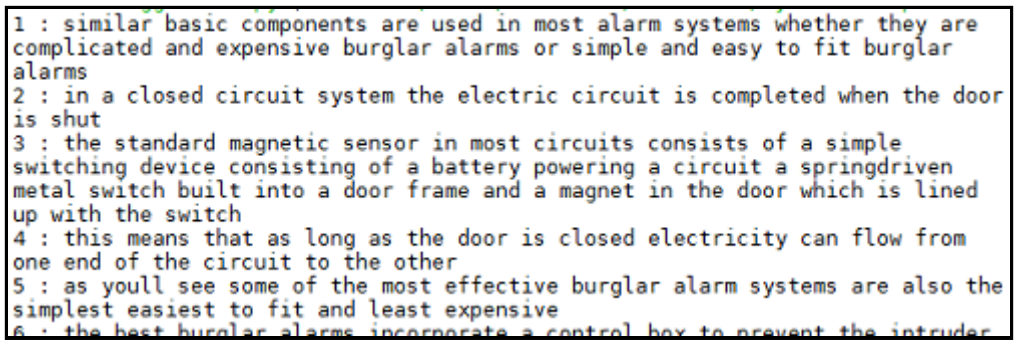
```
idx_dict = get_idx_similar(sentence_input)
next_dict = get_next_sentences(idx_dict)
next_tuples = sorted(next_dict.items(), reverse=True, key=lambda x: x[1])
```

In the above piece of code, you can see the big picture of the program. First, the index of each sentence in the article collection is collected. Then, based on it, the next sentences are picked. The result is then

reordered based on the similarity score of each sentence.

```
counter = 1
for elem in next_tuples:
    print(str(counter)+' : '+elem[0])
    if counter == 10:
        break
    counter += 1
```

Display the results. Change the counter *(if counter == 10:)* to get more sentences.



This is just a simple sentence recommendation program. It does not take into account grammar or word order. Create a better program by joining a machine learning course.