# summarize.py

from \_\_future\_\_ import print\_function

from collections import Counter

from sklearn.feature\_extraction.text import TfidfVectorizer

from nltk.corpus import reuters

from nltk.stem.snowball import SnowballStemmer

from nltk.corpus import stopwords

import nltk.data

import math

import re

DOC\_ROOT = 'docs/'

DEBUG = False

SUMMARY\_LENGTH = 5

stop\_words = stopwords.words('english')

ideal\_sent\_length = 20.0

stemmer = SnowballStemmer("english")

class Summarizer3():

def \_\_init\_\_(self, articles,compression):

self.\_articles = []

self.compression\_prcnt=compression

for doc in articles:

with open(DOC\_ROOT + doc) as f:

headline = f.readline()

url = f.readline()

f.readline()

body = f.read().replace('\n', ' ')

if not self.valid\_input(headline, body):

self.\_articles.append((None, None))

continue

self.\_articles.append((headline, body))

def valid\_input(self, headline, article\_text):

return headline != '' and article\_text != ''

def tokenize\_and\_stem(self, text):

tokens = [word for sent in nltk.sent\_tokenize(text) for word in nltk.word\_tokenize(sent)]

filtered = []

for token in tokens:

if re.search('[a-zA-Z]', token):

filtered.append(token)

stems = [stemmer.stem(t) for t in filtered]

return stems

def score(self, article):

headline = article[0]

sentences = self.split\_into\_sentences(article[1])

print("dividing the text document into sentences")

print(sentences)

frequency\_scores = self.frequency\_scores(article[1])

j=1

r=[]

for i, s in enumerate(sentences):

headline\_score = self.headline\_score(headline, s, j) \* 1.5

length\_score = self.length\_score(self.split\_into\_words(s)) \* 1.0

print("length score for sent %d" % j)

print(length\_score)

position\_score = self.position\_score(float(i+1), len(sentences)) \* 1.0

print("position score for sent %d" % j)

print(position\_score)

frequency\_score = frequency\_scores[i] \* 4

print("frequency score for sent %d" % j)

print(frequency\_score/4)

score = (headline\_score + frequency\_score + length\_score + position\_score) / 4.0

r.append([score,j])

print("final score for sent %d" % j)

print(j,score,sep = ' - ')

j=j+1

self.\_scores[s] = score

n=sorted(r,reverse=True)

print(n)

def generate\_summaries(self):

total\_num\_sentences = 0

for article in self.\_articles:

total\_num\_sentences += len(self.split\_into\_sentences(article[1]))

print("total sent %d" % total\_num\_sentences)

SUMMARY\_LENGTH=((self.compression\_prcnt\*total\_num\_sentences)/100)

print("summary length %d" % SUMMARY\_LENGTH)

if total\_num\_sentences <= SUMMARY\_LENGTH:

return [x[1] for x in self.\_articles]

self.build\_TFIDF\_model()

self.\_scores = Counter()

for article in self.\_articles:

self.score(article)

highest\_scoring = self.\_scores.most\_common(SUMMARY\_LENGTH)

#if DEBUG:

# print(highest\_scoring)

#print("## Headlines: ")

headlines="## Headlines: \n"

for article in self.\_articles:

#print("- " + article[0])

headlines+="- " + article[0]+"\n"

return headlines,' '.join([sent[0] for sent in highest\_scoring])

def split\_into\_words(self, text):

""" Split a sentence string into an array of words """

try:

text = re.sub(r'[^\w ]', '', text) # remove non-words

return [w.strip('.').lower() for w in text.split()]

except TypeError:

return None

def remove\_smart\_quotes(self, text):

return text.decode('utf-8').strip().replace(u"\u201c","").replace( u"\u2014","").replace(u"\u201d", "").replace(u"\u2019", "").replace(u"\u2018", "").replace( u"\u20ac","")

def split\_into\_sentences(self, text):

tok = nltk.data.load('tokenizers/punkt/english.pickle')

sentences = tok.tokenize(self.remove\_smart\_quotes(text))

sentences = [sent.replace('\n', '') for sent in sentences if len(sent) > 10]

return sentences

## ----- CALCULATING WEIGHTS FOR EACH FEATURE -----

def headline\_score(self, headline, sentence, num):

""" Gives sentence a score between (0,1) based on percentage of words common to the headline. """

title\_stems = [stemmer.stem(w) for w in headline if w not in stop\_words]

sentence\_stems = [stemmer.stem(w) for w in sentence if w not in stop\_words]

count = 0.0

print("headline score of sent %d" % num)

for word in sentence\_stems:

if word in title\_stems:

count += 1.0

score = count / len(title\_stems)

print(score)

return score

def length\_score(self, sentence):

""" Gives sentence score between (0,1) based on how close sentence's length is to the ideal length."""

len\_diff = math.fabs(ideal\_sent\_length - len(sentence))

return len\_diff / ideal\_sent\_length

def position\_score(self, i, size):

""" Yields a value between (0,1), corresponding to sentence's position in the article.

Assuming that sentences at the very beginning and ends of the article have a higher weight.

"""

relative\_position = i / size

if 0 < relative\_position <= 0.1:

return 0.17

elif 0.1 < relative\_position <= 0.2:

return 0.23

elif 0.2 < relative\_position <= 0.3:

return 0.14

elif 0.3 < relative\_position <= 0.4:

return 0.08

elif 0.4 < relative\_position <= 0.5:

return 0.05

elif 0.5 < relative\_position <= 0.6:

return 0.04

elif 0.6 < relative\_position <= 0.7:

return 0.06

elif 0.7 < relative\_position <= 0.8:

return 0.04

elif 0.8 < relative\_position <= 0.9:

return 0.04

elif 0.9 < relative\_position <= 1.0:

return 0.15

else:

return 0

def build\_TFIDF\_model(self):

token\_dict = {}

for article in reuters.fileids():

token\_dict[article] = reuters.raw(article)

self.\_tfidf = TfidfVectorizer(tokenizer=self.tokenize\_and\_stem, stop\_words='english', decode\_error='ignore')

tdm = self.\_tfidf.fit\_transform(token\_dict.values()) # Term-document matrix

def frequency\_scores(self, article\_text):

response = self.\_tfidf.transform([article\_text])

feature\_names = self.\_tfidf.get\_feature\_names() # stemmed words

word\_prob = {} # TF-IDF individual word probabilities

for col in response.nonzero()[1]:

word\_prob[feature\_names[col]] = response[0, col]

if DEBUG:

print(word\_prob)

sent\_scores = []

print("words remained after performing stopword removal")

print(word\_prob)

m=1

for sentence in self.split\_into\_sentences(article\_text):

score = 0

sent\_tokens = self.tokenize\_and\_stem(sentence)

print("after tokenization and stemming for sent %d" % m)

print(sent\_tokens)

for token in (t for t in sent\_tokens if t in word\_prob):

score += word\_prob[token]

print(token,word\_prob[token],sep = ' - ')

print("sentence score for sentence %d before dividing with sent length" % m)

print(m,score,sep = ' - ')

print("sent lenght %d" % len(sent\_tokens))

m=m+1

# Normalize score by length of sentence

sent\_scores.append(score / len(sent\_tokens))

print("frequency score obtained after dividing score with length of sentence")

p=1

for n in sent\_scores:

print(p,n,sep = ' - ')

p=p+1

return sent\_scores