به نام یزدان پاک

دانشگاه صنعتی امیرکبیر

دانشکده مهندسی کامپیوتر

درس بازیابی اطلاعات

گزارش پروژه

استاد راهنما:

دکتر نیک‌آبادی

تهیه‌کننده:

آیلار صدائی

فاز اول

پرسش اول:

در گام پیش‌پردازش ابتدا از طریق کتابخانه json، محتویات فایل json را می‌خوانیم و در آرایه ذخیره می‌کنیم. هر عضو آرایه، متعلق به یک سند است. یعنی یک دیکشنری که شامل اطلاعات مختلف سندها از جمله عنوان، محتوا، آدرس اینترنتی، برچسب‌ها و... است.

در این پروژه ما فقط با استفاده از محتوای سند بازیابی را انجام می‌دهیم، بنابراین محتواها را به صورت جدا در یک آرایه ذخیره می‌کنیم.

سپس تابع preprocess را روی این آرایه اعمال می‌کنیم. این تابع آرایه‌ای از رشته‌ها که هر یک نشان‌دهنده محتوای یک سند هستند را گرفته و در خروجی، به ما لیستی می‌دهد که در آن به ازای هر سند، یک لیست از token ها وجود دارد.

این توکن‌ها با استفاده از کتابخانه پردازش زبان فارسی «هضم» به دست آمده اند.

ابتدا با استفاده از normalize، متن را از نظر حذف حروفی که چند شکل دارند (از جمله ک و ی، که فرم عربی آنها کاراکتر متفاوتی در utf-8 است) را یکسان‌سازی می‌کند.

سپس رشته محتوا را به تابع word_tokenize می‌دهیم و این تابع، لیستی از توکن‌ها به ما تحویل می‌دهد. سپس چک می‌کنیم توکن‌ها در لیست حروف اضافه (stopwords) و یا علائم نگارشی (punctuations) نباشند. سپس توکن‌ها را با استفاده از lemmatizer ریشه‌یابی می‌کنیم و به لیست نهایی توکن‌های آن سند اضافه می‌کنیم. بعد از ساخته شدن لیست هر سند، آن را به لیستی که در نهایت می‌خواهیم به عنوان خروجی تابع برگردانیم، اضافه می‌کنیم.

```python
def preprocess(documents):
  documents_tokens_list = []
  for document in documents:
    tokens_list = []
    normalized_document = normalizer.normalize(document)
    tokens = word_tokenize(normalized_document)
    punctuation = list (string.punctuation) + ['https://', 'http://', 'ء' ,'،' ,'؟','/','\\' ,'//:' ,'انتهای پیام/', '«', '»']
    for token in tokens:
      if ((token not in stopwords_list()) and (token not in punctuation)):
        tokens_list.append(lemmatizer.lemmatize(token))
    documents_tokens_list.append(tokens_list)

  # print (len(documents_tokens_list))
  return documents_tokens_list
```

برای مثال، ما روی ورودی زیر تابع preprocess را اجرا می‌کنیم:

«عمری دگر بِباید، بعد از فِراق ما را... کاین عمر صرف کردیم اندر امیـدواری.»

«ترسم نماز صوفی با صحبت خیالت / باطل بُود که صورت بر قبله می‌نگاری...»

خروجی به صورت زیر نمایش داده می‌شود:

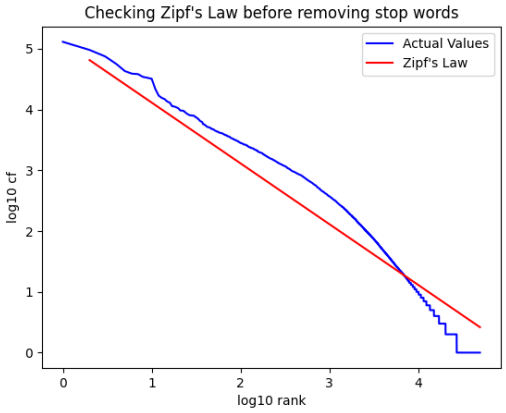
```
strings = ['«...عمری دگر بِباید، بعد از فِراق ما را... کاین عمر صرف کردیم اندر امیـدواری.» ','«ترسم نماز صوفی با صحبت خیالت / باطل بُود که صورت بر قبله می‌نگاری»']
stings_tokens = preprocess (strings)
for x in stings_tokens:
  print (x)
```

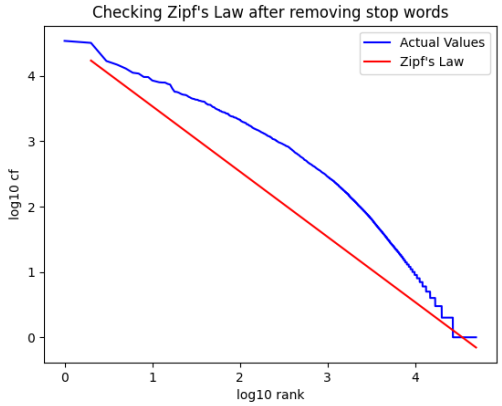
```
['عمر', 'دگر', 'بباید', 'فراق', '...', 'کاین', 'عمر', 'کرد#کن', 'اندر', 'امیدوار']
['ترسید#ترس', 'نماز', 'صوف', 'صحبت', 'خیال', 'باطل', 'صورت', 'قبله', 'می', 'نگاری\u200c', '...']
```

پرسش دوم:

تصویر سمت چپ، منحنی قبل از حذف کلمات stopwords_list، را نشان می‌دهد و تصویر سمت راست، بعد از حذف. خط قرمز هم نشان‌دهنده قانون Zipf است. (نمودار لگاریتمی رسم شده است.)

همانطور که مشاهده می‌شود، هر دو منحنی تقریباً با قانون zipf انطباق دارند.





پرسش سوم:

قبل از ریشه‌یابی:

تعداد اسناد	500	1000	1500	2000
تعداد توکن‌ها	6599	9509	11308	12824
تعداد کل کلمات	142400	292131	442653	581805

پرسش چهارم:

چالش اول: تابع stemmer کتابخانه هضم، ریشه‌یابی را به طوری انجام می‌دهد که ضمایر را از اسم‌ها حذف می‌کند. این موضوع باعث بروز خطا در پیش‌پردازش متون می‌شود. برای مثال کلمه «گزارش»، خودش یک اسم است و نیاز به پردازش ندارد. اما با دادن آن به stemmer کتابخانه هضم، این کلمه به «گزار» تبدیل می‌شود چون حرف «ش» آن ضمیر تشخیص داده شده است. برای همین در پیش‌پردازش از این تابع استفاده نکردم. اما استفاده نکردن از این تابع هم مشکلاتی دارد. مثلاً وقتی دو کلمه فقط در ضمایر متصلشان تفاوت دارند، این دو کلمه را متفاوت تشخیص می‌دهد و برای آن‌ها دو توکن مختلف در نظر می‌گیرد.

چالش دوم: همانظور که گفته شد استفاده از stemmer با خطاهایی روبروست. بنابراین برای ریشه‌یابی کلمات از lemmatizer استفاده کردم که یک تابع دیگر در کتابخانه هضم است. با اینکه این تابع خطاهایی که در بالا گفته شد را ندارد اما باز هم عملکردش به نسبت کتابخانه stemmer از سرعت کمتری برخوردار است.