پیاده‌سازی Query Optimization به دو روش کوتاه ترین لیست و کوچکترین عنصر در لیست ایندکس ها

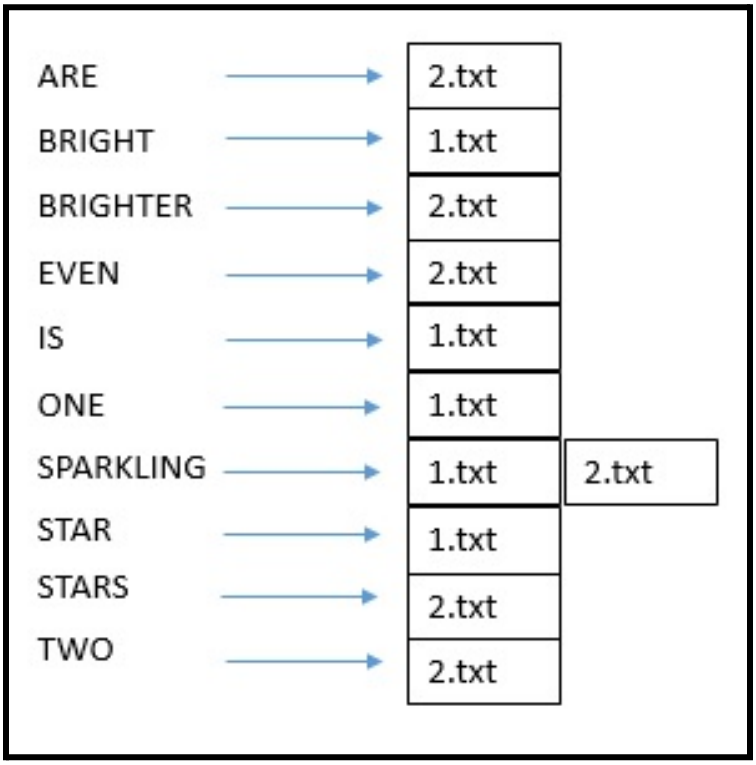


Fig1: codefying.com

بازیابی اطلاعات

استاد تاج بخش

کیانا نصیری

پاییز ۱۴۰۴

فهرست

۱. چکیده

در این تمرین، مسئله بهینه‌سازی پرس‌وجو (Query Optimization) در سیستم‌های بازیابی اطلاعات مورد تحلیل و بررسی قرار گرفته است. هدف اصلی این پروژه، پیاده‌سازی یک سیستم جستجوی کارامد براساس Inverted Index و مقایسه روش های مختلف برای افزایش سرعت پاسخ‌دهی به کویری‌های چندواژه‌ای می‌باشد. جهت اجرای آزمایش‌ها، از مجموعه داده استاندارد ۲۰ گروه خبری (20newsgroups) استفاده شده است. فرآیند کار شامل مراحل پیش‌پردازش متنی دقیق (شامل ریشه‌یابی و حذف کلمات توقف)، ساخت Inverted Index با بیش از ۱۱۱,۰۰۰ واژه منحصربه‌فرد و در نهایت پیاده‌سازی دو متد جستجوی «پردازش لیست‌های کوتاه‌تر در اولویت» و «کوچک‌ترین عنصر در موقعیت آخر» است. عملکرد این دو روش از منظر زمان اجرا و سرعت پردازش مورد ارزیابی قرار گرفته و نتایج حاصله جهت تعیین بهینه‌ترین رویکرد در بخش پایانی گزارش ارائه شده است.

۲. مقدمه

۱.۲. صورت مسئله

بازیابی اطلاعات و بهینه‌سازی پرس‌وجو از جمله مباحث بنیادین در حوزه علوم کامپیوتر و هوش مصنوعی هستند که نقش حیاتی در مدیریت و استخراج دانش از حجم عظیم داده‌های متنی ایفا می‌کنند. با گسترش روز افزون داده‌های غیرساختار یافته، استفاده از الگوریتم‌هایی که توانایی مدیریت فضای حالت و هدایت بهینه جستجو را داشته باشند، اهمیتی دوچندان یافته است. Inverted Index به عنوان ساختار داده‌ای اصلی در موتورهای جستجو، امکان دسترسی سریع به اسناد را فراهم می‌سازد، اما چالش اصلی در چگونگی ترکیب این لیست‌ها در پرس‌وجوهای پیچیده نهفته است.

۲.۲. هدف تمرین

هدف از انجام این تمرین، طراحی و پیاده‌سازی یک سیستم بازیابی اطلاعات خودکار و بررسی تأثیر رویکردهای مختلف بهینه‌سازی بر کارایی جستجو است. در این راستا، اهداف زیر دنبال شده است:

- پیاده‌سازی متدهای پیش‌پردازش متنی جهت استاندارد سازی داده
- ساخت Inverted Index برای مدیریت کارآمد واژگان
- ارزیابی و مقایسه دو الگوریتم جستجوی متفاوت جهت شناسایی تاثیر ترتیب ادغام لیست‌ها بر سرعت
- تحلیل آماری زمان اجرای پرس‌وجوها برای درک رفتار الگوریتم‌ها در مواجهه با کلمات با فراوانی‌های مختلف

۳. توضیحات پروژه

در این بخش، مراحل مختلف پیاده‌سازی سیستم بازیابی اطلاعات، از تحلیل داده‌های خام تا اجرای فرآیندهای بهینه‌سازی، مورد بررسی قرار می‌گیرد.

۱.۳. بررسی مجموعه داده

جهت اجرای آزمایش‌ها و ارزیابی کارایی سیستم با توجه به نمایه ۱، از مجموعه داده استاندارد *20newsgroups* استفاده شده است. این مجموعه شامل حدود ۱۱۳۱۴ سند متنی در دسته‌بندی‌های مختلف خبری است. در مرحله نخست، ساختار داده‌ها تحلیل شده و توزیع کلمات در اسناد مورد ارزیابی اولیه قرار گرفته است تا بستری مناسب برای ساخت نمایه معکوس فراهم گردد. همچنین یک مورد نمونه از داده مورد بررسی قرار گرفته که در تصویر ۱ مشاهده می‌شود.

Exploring Dataset

```
from sklearn.datasets import fetch_20newsgroups
newsgroups_train = fetch_20newsgroups(subset='train')
```
[1] ✓ 2.0s

```
print(len(newsgroups_train['data']))
```
[2] ✓ 0.0s

```
11314
```

```
print(newsgroups_train['data'][0])
```
[3] ✓ 0.0s

```
From: lerxst@wam.umd.edu (where's my thing)
Subject: WHAT car is this!?
Nntp-Posting-Host: rac3.wam.umd.edu
Organization: University of Maryland, College Park
Lines: 15

 I was wondering if anyone out there could enlighten me on this car I saw
the other day. It was a 2-door sports car, looked to be from the late 60s/
early 70s. It was called a Bricklin. The doors were really small. In addition,
the front bumper was separate from the rest of the body. This is
all I know. If anyone can tellme a model name, engine specs, years
of production, where this car is made, history, or whatever info you
have on this funky looking car, please e-mail.

Thanks,
- IL
   ---- brought to you by your neighborhood Lerxst ----
```

تصویر ۱: بخشی از کد

۲.۳ مراحل پیش‌پردازش

به منظور افزایش دقت در بازیابی و کاهش حجم محاسبات، فرآیند پیش‌پردازش متنی در چندین مرحله بر روی تمامی اسناد اجرا شده است که در تصویر ۲ مشاهده میشود. این مراحل عبارتند از:

- **پاک‌سازی متنی:** حذف کاراکترهای غیر الفبایی و علائم نگارشی.
- **Normalizing:** تبدیل تمامی حروف به حالت کوچک (Lowercasing).
- **حذف کلمات توقف:** حذف کلمات پرتکرار و فاقد ارزش معنایی (مانند "is" ,"and" ,"the").

- **ریشه‌یابی (Stemming):** استفاده از الگوریتم Porter Stemmer جهت کاهش کلمات به ریشه اصلی آن‌ها (به عنوان مثال تبدیل running به run).

```
def preprocess_text(text):
    # Convert to lowercase
    text = text.lower()
    # Remove email addresses
    text = re.sub(r'\S+@\S+', '', text)
    # Remove URLs
    text = re.sub(r'http\S+|www\.\S+', '', text)
    # Remove numbers
    text = re.sub(r'\d+', '', text)
    # Remove punctuation and special characters (but preserve spaces)
    text = re.sub(r'[^\w\s]', ' ', text)
    # Remove extra whitespace
    text = re.sub(r'\s+', ' ', text).strip()
    # Tokenize
    words = text.split()
    # Remove stopwords and very short words (less than 2 characters)
    words = [word for word in words if word not in stop_words and len(word) > 2]
    # Lemmatize
    words = [lemmatizer.lemmatize(word) for word in words]
    return ' '.join(words)

# Apply preprocessing to all training data
preprocessed_data = []
print("Preprocessing training data...")
for doc in tqdm[Any](newsgroups_train['data']):
    preprocessed_data.append(preprocess_text(doc))

print(f"Preprocessing complete. Sample of first preprocessed document:")
print(preprocessed_data[0][:200] + "..." if len(preprocessed_data[0]) > 200 else preprocessed_data[0])
print(f"Original document count: {len(newsgroups_train['data'])}, Preprocessed document count: {len(preprocessed_data)}")
```

```
Preprocessing training data...
100%|██████████| 11314/11314 [00:13<00:00, 817.01it/s]
Preprocessing complete. Sample of first preprocessed document:
thing subject car nntp posting host rac wam umd edu organization university maryland college park line wondering anyone could enlig
```

تصویر ۲: بخشی از کد preprocessing

نمونه‌ای از داده قبل و بعد از پیش پردازش:

```
From: lerxst@wam.umd.edu (where's my thing)
Subject: WHAT car is this!?
Nntp-Posting-Host: rac3.wam.umd.edu
Organization: University of Maryland, College Park
Lines: 15

 I was wondering if anyone out there could enlighten me on this car I saw
the other day. It was a 2-door sports car, looked to be from the late 60s/
early 70s. It was called a Bricklin. The doors were really small. In addition,
the front bumper was separate from the rest of the body. This is
all I know. If anyone can tellme a model name, engine specs, years
of production, where this car is made, history, or whatever info you
have on this funky looking car, please e-mail.

Thanks,
- IL
   ---- brought to you by your neighborhood Lerxst ----




---------After Preprocessing---------
lerxstwamumdedu wheres thing subject car nntppostinghost racwamumdedu organization university maryland college park line wondering anyo
--------------------------------------
```

تصویر ۳: نمونه از داده قبل و بعد بیش پردازش

۳.۳. روش‌های جستجو و بهینه‌سازی (Search Methods)

جهت پاسخ‌دهی به پرس‌وجوهایی مانند ترکیب Boolean AND Queries، دو متد متفاوت مورد بحث در کلاس از نظر زمان اجرا و مقایسه شده‌اند:

- **روش اول: پردازش لیست‌های کوتاه‌تر در اولویت Shorter Posting Lists First**

در این روش، واژگان موجود در پرس‌وجو بر اساس طول Posting List آن‌ها به صورت صعودی مرتب‌سازی میشوند. منطق حاکم بر این روش محدود سازی سریع از همان ابتدا است.

در این رویکرد تصویر ۴، با انتخاب کوتاه‌ترین لیست به عنوان نقطه شروع مجموعه اسناد مربوطه به کمترین تعداد ممکن تقلیل می‌یابد. در مراحل بعدی، به جای پیمایش کامل لیست‌های طولانی، تنها وجود اسناد معدود به‌دست‌آمده از مرحله اول در این لیست‌های بزرگ بررسی می‌شود. این فرآیند به عنوان یک فیلتر زودهنگام عمل کرده و مانع از پردازش حجم عظیمی از داده‌هایی می‌گردد که در نهایت در نتیجه نهایی حضور نخواهند داشت.

- **روش دوم: کمترین تکرار در موقعیت آخر Fewer in Last Position**

در این رویکرد، اولویت‌بندی بر اساس ساختار توزیع شناسه‌ها در انتهای لیست‌های نمایه صورت می‌گیرد. در این متد، واژگانی که در موقعیت آخر لیست خود دارای کمترین تکرار مشترک با سایر واژگان هستند، در اولویت پردازش قرار می‌گیرند. هدف این روش، شناسایی واژگانی است که احتمال همپوشانی کمتری با کل مجموعه داشته و می‌توانند با احتمال بیشتری منجر به حذف سریع اسناد غیر مرتبط شوند.

```python
# Function to search using the first approach: process terms with shorter posting lists first
def search_shorter_posting_lists(query_terms, inverted_index):
    if not query_terms:
        return []
    # Sort terms by length of posting lists (ascending)
    sorted_terms = sorted(query_terms, key=lambda term: len(inverted_index[term]))
    # Start with the posting list of the first term
    result = set[Any](inverted_index[sorted_terms[0]])
    # Intersect with remaining terms
    for term in sorted_terms[1:]:
        result = result.intersection(set[Any](inverted_index[term]))
        if not result:  # Early termination if intersection is empty
            break
    return list[Any](result)

⌘L to chat, ⌘K to generate
# Function to search using the second approach: process terms with fewer documents in the last position
def search_fewer_last_position(query_terms, inverted_index):
    if not query_terms:
        return []
    # Sort terms by the number of documents in the last position
    def count_last_position(term):
        posting_list = inverted_index[term]
        if not posting_list:
            return 0
        last_doc_id = posting_list[-1]

        # Pre-compute last document IDs for all query terms to avoid repeated lookups
        last_doc_ids = {t: inverted_index[t][-1] if inverted_index[t] else None for t in query_terms}

        # Count terms with matching last document ID
        count = sum(1 for t_id in last_doc_ids.values() if t_id == last_doc_id)
        return count

    # Sort once and reuse the result
    sorted_terms = sorted(query_terms, key=count_last_position)
    # Start with the posting list of the first term
    result = set[Any](inverted_index[sorted_terms[0]])

    # Intersect with remaining terms
    for term in sorted_terms[1:]:
        result = result.intersection(set[Any](inverted_index[term]))
        if not result:  # Early termination if intersection is empty
            break

    return list[Any](result)
```

تصویر ۴: بخشی از کد two search methods

۴. نتایج

۱.۴. ارزیابی عملکرد متدها

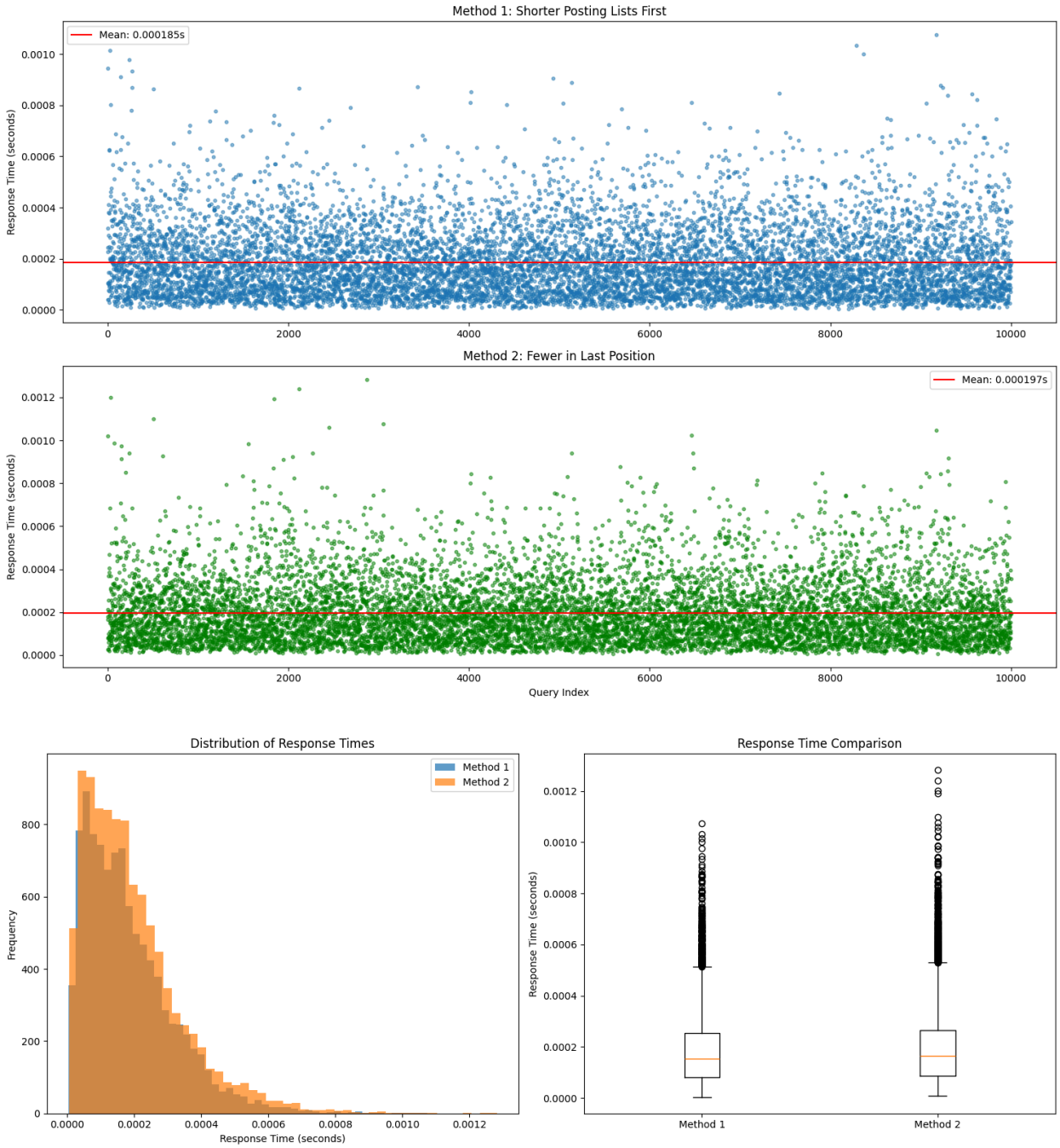
- **روش اول Shorter Posting Lists First:** این متد به طور میانگین با سرعت بالاتری داشت. در اکثر پرس‌وجوها، زمان اجرای این روش حدود ۱.۱۱ برابر سریع‌تر از روش دوم است.
- **روش دوم Fewer in Last Position:** اگرچه این روش در برخی سناریوهای خاص عملکرد خوبی داشت، اما به دلیل پیچیدگی در محاسبات مربوط به موقعیت آخر و عدم تمرکز مستقیم بر حجم داده‌های ورودی، کندتر بود.

۲.۴. مثال هایی از رکوردها

در جدول زیر، چهار نمونه از پرس‌وجوهایی که در آن‌ها روش اول عملکرد بهتری نسبت به روش دوم (Fewer in Last Position) داشته است، آورده شده است:

شماره کوئری	واژگان پرس‌وجو	میزان کندتر بودن روش دوم	علت برتری روش اول
۱۰۰۵	['apt', 'finnegan', 'never']	**۷.۳۹ برابر**	واژه 'never' لیست طویلی دارد که روش ۱ با اولویت دادن به واژگان دیگر، آن را خنثی کرد.
۷۸۶۱	['string', 'day', 'bashing']	**۷.۳۳ برابر**	تفاوت فاحش در طول لیست 'day' نسبت به 'bashing'.
۸۲۰۳	cordoba', 'bouhdiba','] ["jaca	**۶.۶۵ برابر**	وجود کلمات بسیار خاص که فضای جستجو را در گام اول به شدت محدود کردند.
۸۸۴۱	newsreader', 'used','] ["trons', 'fort	**۵.۷۲ برابر**	واژه 'used' به عنوان یک کلمه توقف عمل کرده و روش ۱ از پیمایش لیست آن جلوگیری کرد.

نمودار مدت زمان پاسخ ۱۰۰۰۰ کویری و میانگین دو روش که در این نمونه تست, روش اول با میانگین 0.000185 ثانیه سریعتر از مدل دوم که با میانگین 0.000197 ثانیه بود.



۵. تحلیل

۱.۵. برتری روش اول در داده‌های واقعی

نهایتا حین تست با بالغ بر ۱۰ دسته ی ۱۰۰۰۰ تایی از کویری های ۳ تا ۵ کلمه ای، این دو روش بر روی داده مورد بررسی بسیار نزدیک بهم بودند، و در یک مورد فقط روش دوم (Fewer in Last Position) به اندازه ی 0.57x سریعتر بود. روش Shortest List First به طور میانگین بین ۰.۹ تا ۱.۲ برابر سریع‌تر از روش دوم بود که این نتیجه را در محیط‌های عملیاتی و داده‌های واقعی می‌توان از چند منظر زیر تحلیل نمود:

1. **کاهش فضای جستجو در گام نخست:** در داده‌های واقعی، تعداد کمی از کلمات بسیار پرتکرار و حجم عظیمی از کلمات بسیار کم‌تکرار هستند. روش اول کوچک‌ترین مجموعه ممکن را به عنوان پایه انتخاب می‌کند، که همان طور که در کلاس صحبت شد، این مورد باعث کاهش فضای جست و جو می‌شود که لیست های طولانی چند بار trace نشوند.
2. **سربار محاسباتی کمتر:** روش دوم سعی می‌کند بر اساس توزیع ایندکس ها در انتهای لیست تصمیم‌گیری کند. در داده‌های واقعی، این استراتژی همواره تضمین‌کننده کاهش حجم محاسبات نیست، زیرا ممکن است توزیع ایندکس‌ها در انتهای لیست‌ها الگوی منظمی نداشته باشد. در مقابل، روش اول بر مبنای طول لیست عمل می‌کند که هیچ‌گونه سربار محاسباتی اضافی به سیستم تحمیل نمی کند.
3. **محدود بودن تنوع داده در مجموعه‌ی فعلی:**
   نکته‌ی قابل توجه دیگر این است که در مجموعه‌ی فعلی شامل حدود ۱۱۰۰۰ لیست رشته‌ای، با وجود حجم حدودی ۲۰۰ کلمه در هر نمونه، همچنان با یک خوشه‌ی کوچک و نسبتاً همگن از داده‌ها مواجه هستیم. این موضوع باعث می‌شود تفاوت‌های بالقوه‌ی دو روش به‌طور کامل بروز پیدا نکند و عملکرد آن‌ها بسیار به هم نزدیک باقی بماند. در چنین شرایطی، اختلاف‌های چشمگیر در زمان اجرا مشاهده نشد.
4. **مقاومت در برابر کلمات توقف:** کلماتی مانند never یا used که لیست‌های ایندکس بسیار طولانی دارند، در صورت عدم حذف کامل در پیش‌پردازش، می‌توانند گلوگاه ایجاد کنند. روش اول با انتقال این کلمات به انتهای زنجیره پردازش، باعث میشوند در اخر پردازش شوند؛ چرا که این لیست‌های عظیم تنها به عنوان فیلتر نهایی بر روی تعداد بسیار محدودی از اسناد (که از مراحل قبل به دست آمده‌اند) اعمال می‌شوند.

۶. منابع

- Claude 4.5
- Gemini 3 Thinking mode
- Source code and results