به نام خدا

فاز سوم پروژه بازیابی پیشرفته اطلاعات

مدرس : مهدیه سلیمانی و سیده فاطمه سیدصالحی

تهیه‌کنندگان : حسین ابراهیمی
نیما فتحی
سینا توکلی

نیمسال دوم سال تحصیلی ۰۰-۹۹
دانشکده مهندسی کامپیوتر
دانشگاه صنعتی شریف

۱ پیاده‌سازی خزنده، واکشی اطلاعات مقالات

در این بخش قصد داریم برای سایت Microsoft Academic یک خزنده پیاده‌سازی کرده و با استفاده از آن اطلاعات تعدادی مقاله را واکشی کنیم.
اطلاعاتی که می‌خواهیم از مقالات واکشی کنیم:

۱. آیدی مقاله
۲. عنوان مقاله
۳. چکیده مقاله
۴. سال انتشار مقاله
۵. نویسندگان مقاله
۶. موضوعات مرتبط
۷. تعداد استنادهای مقاله (citations)
۸. تعداد ارجاعات مقاله (references)
۹. ارجاعات مقاله (تنها ده ارجاع اول که در صفحه مقاله سایت Microsoft Academic قرار دارد کافی است و نیازی به واکشی تمامی ارجاعات نیست.)

برای آغاز کار از تعدادی مقاله که در فایل start.txt وجود دارد استفاده می‌کنیم. این مقالات در صف خزش قرار می‌گیرند و ۲۰۰۰ مقاله را ذخیره می‌کنیم (پارامتر تعداد مقالات را به عنوان ورودی در نظر بگیرید). پس از خزش هر مقاله آدرس ۱۰ مرجع اول (در صورت کمتر از ۱۰ هر تعدادی که دارد) در صف خزش قرار می‌گیرد. نکته مهم این است که برخی مقالات چند نسخه منتشر شده در وبسایت‌های مختلف است و یکی از این نسخه‌ها باید ذخیره شود و هیچ مقاله نباید بیش ای یکبار ذخیره شود.
اطلاعات واکشی شده باید به صورت یک فایل json ذخیره شود. یک نمونه از اطلاعات ذخیره شده در فایل نمونه با نام example.json قرار داده شده است. شما در عمل باید تعداد ۲۰۰۰ مورد مانند اطلاعات موجود در فایل example.json را در یک فایل با نام CrawledPapers.json ذخیره کنید (یعنی این فایل حاوی لیستی از اطلاعات ۲۰۰۰ مورد مقاله واکشی شده باشد).

نکات مهم:

- ممکن است لازم باشد برای واکشی اطلاعات میان هر واکشی یک مقدار delay قرار دهید (مثلا چهار ثانیه کافی است).
- برای اینکار باید از زبان پایتون استفاده کنید پیشنهاد ما استفاده از فریم‌ورک‌های scrapy یا selenium می‌باشد.
- شما باید تمام ارجاعات درون صف را واکشی کنید اگر به هر دلیلی خزنده شما متوقف شد باید دوباره آن را راه‌اندازی کرده تا رسیدن به تعداد درخواست شده ادامه دهید (تنها وقتی صف شما تمام شده باشد و به تعداد ۲۰۰۰ عدد نرسیده باشید مشکلی ندارد که البته ممکن هم نیست!).
- برای آنکه از نسخه‌های متفاوت مقاله بیش از یکبار واکشی رخ ندهد از ID آن استفاده کنید این ID در بخش href مراجع و url خود مقاله یکتا مقاله را مشخص می‌شود. یعنی در لینک مقاله آیدی آن:

  https://academic.microsoft.com/paper/(id)

  همان بخش id می‌باشد. (برای رفرنس‌ها هم با مشاهده الگو خودتان متوجه می‌شوید!)
- در آخر پیشنهاد ما استفاده از scrapy می‌باشد و سعی کنید از CSS Selector ها استفاده بهینه کنید.

۲ ارزش گذاری مقالات با PageRank

در این قسمت شما باید مقدار PageRank مقالات واکشی شده در فایل CrawledPapers.json را محاسبه کرده و در یک فایل با نام PageRank.json با فرمت زیر ذخیره کنید.

```
{
"Paper_id" :"page rank value",
"Paper_id": "page rank value",
...
}
```

ورودی

- فایل اطلاعات CrawledPapers.json
- مقدار α

خروجی

- فایل نهایی مقادیر PageRank مقالات با نام CrawledPapers.json.

۳ رتبه‌بندی نویسندگان بر اساس الگوریتم HITS

در بخش‌های پیشین آموختید که چگونه ارجاعات میان مقاله‌ها را مدل کنید و با استفاده از معیار PageRank مقاله‌ها را به صورت مرتب‌شده جستجو کنید.
حال مفهموم جدیدی را برای این بخش معرفی می‌کنیم. می‌گوییم فرد X به فرد Y ارجاع دارد اگر فرد X مقاله‌ای نوشته باشد که به یکی از مقالات فرد Y ارجاع داشته باشد و در چنین حالتی یک یال از X به Y در نظر گرفته خواهد شد.
با این تعریف جدید می‌خواهیم با استفاده از روش HITS و محاسبه‌ی شاخص‌های hub و authority نویسندگان سایت را رتبه‌بندی کنیم. برای این کار لازم است از معیار authority برای امتیازدهی به افراد و تشخیص افراد شاخص (n نفر برتر) استفاده کنید.

ورودی

- فایل اطلاعات CrawledPapers.json
- تعداد نویسندگان برتر مورد نظر (n)

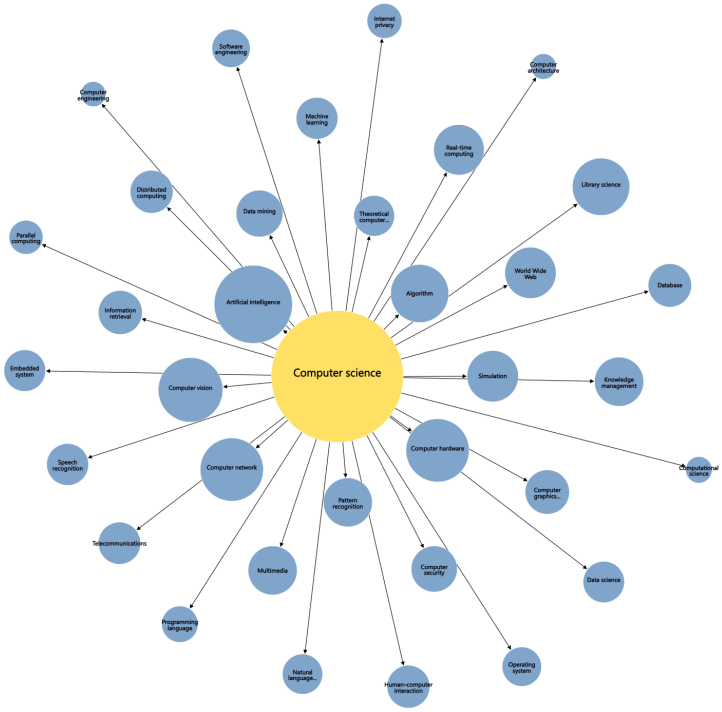
خروجی

- لیست n نویسنده‌ی برتر به همراه مقدار معیار authority برای هر کدام.

نکات: برای پیاده‌سازی روش HITS عدد ۵ را به عنوان تعداد اجرای حلقه در نظر بگیرید.

۴ پیاده‌سازی یک Recommender System

در این قسمت بر اساس داده‌ای که از خوانندگان Microsoft Academic در اختیار داریم قصد داریم یک سیستم پیشنهاد دهنده از بین مقالات واکشی شده برای کاربران طراحی کنیم. مجموعه دادگان data.csv از ماتریس $M_{n\times m}$ تشکیل شده است که سطرهای آن کاربران و ستون‌هایش موضوعات مربوط به علوم کامپیوتر است. هر درایه M_{ij} نشان‌دهنده‌ی نسبت تعداد مقالات خوانده شده کاربر i در موضوع j به تعداد کل مقاله‌های خوانده شده توسط او است. به طور مثال اگر یک کاربر در کل ۴۰ مقاله خوانده باشد و ۱۰ تا از آن‌ها مرتبط به موضوع یادگیری ماشین باشد، مقدار M_{ij} برابر با ۰٫۲۵ خواهد بود.



۱.۴ روش Content-based

در این روش ایده آن است که آیتم‌هایی را به کاربر پیشنهاد بدهیم که قبلا به مشابه‌های آن امتیاز بالایی داده است. حال در مسئله‌ی ما، امتیاز را می‌توان تعداد مقالات خوانده شده در یک موضوع توسط کاربر تعبیر کرد.
ابتدا نیاز است که برای هر مقاله پروفایلی همانند کاربران ساخته شود. برای این کار نیاز است که بردار v برای هر مقاله به شکلی ساخته شود که اگر مقاله به موضوع i مرتبط باشد مقدار درایه آن (v_i) ۱ و در غیر این صورت صفر در نظر گرفته شود (موضوعات مربوطه از صفحه‌ی مقاله در سایت واکشی شده است). سپس با استفاده از ضرب داخلی بردار v با پروفایل کاربر، میزان ارتباط مقاله به موضوعات مطالعه شده توسط کاربر بدست آورده و بر اساس آن پیشنهادات داده می‌شود.
حال با استفاده از بردارهای بدست آمده، تابعی بنویسید که با گرفتن پروفایل کاربر، ۱۰ مقاله‌ی مرتبط به آن کاربر را خروجی دهد.

ورودی

- پروفایل کاربر x
- بردارهای بدست آمده برای مقالات

خروجی

- ۱۰ مقاله‌ی مشابه به پروفایل کاربر x (برای تخمین شباهت میان کاربر و مقالات از cosine-similarity استفاده کنید)

۲.۴ روش Collaborative Filtering

در این راهکار سعی می‌کنیم که سایر علایق کاربر که اطلاعاتی از آن در داده‌ی ما نیست را از طریق پروفایل‌های مشابه بدست آوریم. به طور مثال اگر کاربر x تعداد زیادی مقاله در رابطه با موضوع بینایی ماشین مطالعه کرده است، به احتمال زیاد به موضوعات یادگیری ماشین و هوش مصنوعی نیز علاقه خواهد داشت. حال سعی بر آن است که این اطلاعات از طریق کاربران دیگر با علایق مشابه بازیابی شود.
در این روش ابتدا باید N کاربر دیگر که پروفایل مشابه با x دارند را یافت و سپس از طریق آن‌ها مقادیر ناموجود کاربر x را تخمین زد و در انتها بر اساس پروفایل جدید پیشنهادهایی به کاربر داد.

ورودی

- پروفایل کاربر x
- عدد N

خروجی

- نرمالایز شده پروفایل کاربر x
- ۱۰ مقاله‌ی مشابه به پروفایل جدید کاربر x

۳.۴ کامل کردن ماتریس ورودی (امتیازی)

یکی از مشکلاتی که در سیستم‌های پیشنهاده دهنده با آن رو به رو هستیم، به شدت sparse بودن داده‌ی ورودی است که باعث می‌شود اطلاعات کاملی از کاربران و علایق آن‌ها نداشته باشیم. یکی از روش‌هایی که به ما کمک می‌کند این missing values ها را بازیابی کنیم، مدل‌های Latent Factor هستند. در این روش‌ها فرض می‌شود که ماتریس امتیازات low-rank است و توسط چندین بردار قابل بازیابی است.
فرض کنید ماتریس ورودی ما $M_{n\times m}$ است و می‌خواهیم آن را توسط ضرب دو ماتریس $P_{n\times k}$ و $Q_{k\times m}$ تخمین بزنیم به شکلی که $M \approx P \times Q$. در این صورت خواهیم داشت:

$$M_{ij} = p_i \cdot q_j = \sum_{t=1}^{k} p_{it} \cdot q_{tj}, \qquad p_i, q_j \in \mathbb{R}^k \tag{۱}$$

حال اگر ماتریس‌های P و Q را داشته باشیم، آن وقت missing values ها از طریق رابطه‌ی (۱) قابل تخمین هستند.
برای بدست آوردن این ماتریس‌ها از مقادیر موجود در M یا همان مشاهدات استفاده کرده و سعی می‌کنیم اختلاف بین مقادیر پیش‌بینی شده و مشاهدات را کمینه کنیم. (Sum of Squared Errors)

$$\min_{P,Q} \sum_{i,j\ observed} (M_{ij} - p_i \cdot q_j)^2 \tag{۲}$$

- نشان دهید که اگر به جای مقادیر ناموجود (missing values) صفر داشتیم، تجزیه‌ی SVD تابع هدف بالا را کمینه می‌کرد.

Algorithm 1 Gradient descent for finding P and Q

1: Initialize P and Q (using SVD, pretend missing values are 0), and $k = 2$
2: **for** s steps **do**
3: Update P: $\forall t = 1, \cdots, k, \forall i = 1, \cdots, n$

$$p_{it} \leftarrow p_{it} - \eta \cdot 2 \sum_{j} (M_{ij} - p_i \cdot q_j) p_{it}$$

4: Update Q: $\forall t = 1, \cdots, k, \forall j = 1, \cdots, m$

$$q_{tj} \leftarrow q_{tj} - \eta \cdot 2 \sum_{i} (M_{ij} - p_i \cdot q_j) q_{tj}$$

5: **end for**

- ابتدا 80% داده را به عنوان داده‌ی آموزشی و باقی را برای تست مدل در نظر بگیرید. سپس با استفاده از الگوریتم گرادیان کاهشی زیر تابع هدف را بر روی داده‌ی آموزشی کمینه کرده و خطا بر روی داده‌ی تست گزارش کنید.

در الگوریتم بالا، η ضریب یادگیری و s تعداد مراحل اجرای حلقه تا همگرایی است.

۵ رابط کاربری

شما باید یک واسط کاربری ساده تحت کنسول برای اجرای تعاملی بخش‌های مختلف سیستم و همچنین مشاهده نتایج پیاده‌سازی کنید. در صورت پیاده سازی زیبا و بهتر رابط کاربری تا ده نمره نمره امتیازی نیز در نظر گرفته خواهد شد.