بازیابی پیشرفته اطلاعات – جلسه ۱– ۲۰ اسفند

به لطف سامان کفشدوزی

مقدمه درس موتور های جستجو (کار با متن و **text)**

بازیابی اطلاعات

Information retrieval

تعریف : پیدا کردن یک سری متریال (موجدیت ها ، اقلام) که این متریال ها از سند هستند (**document**) و غیر ساخت یافته هم هستند ، یعنی سند ها **exel** نیستند. حاوی تصویر نیستند فقط متن هستند که یک نیاز اطلاعاتی را در میان حجم زیادی از اسناد براورده میکنند .

نیاز های اطلاعاتی :

جستجو در ایمیل ،جستجو در لپتاب ،جستجو در **knowledge base** ،بازیابی اطلاعات قانونی

نکته : جستجو در بحث ما فقط شامل **web** و فایل هانیست بلکه **search** لایبری میتونه باشه مثلا یک کتابخونه ی لایبری داریم و میخواهیم بگردیم

صفحه ۴

نمودار نمایان گر این است که داده های ساختار نیافته از داده های ساختار یافته پیشی میگرند . در ابتدا **DB** های رابطه ای بودن که متن کم بود ولی الان با **social network** متن زیاد شده است.

نقطه : فرض میکنیم مجموعه اسنادمون ثابتند (این فرض درست نیست)

صفحه ۵

در **network** اسناد همیشه در حال تغییر هستند .

هدف ما : ما سند هایی رو میخواهیم پیدا کنیم که مرتبطند (**relevant**)به نیاز اطلاعاتی ما.

صفحه ۶

Preasion: دقت

Recall: به خاطر آوردن

صفحه ۱۰

در **linux**:

greb برای **search**

در متن : **string** بر میگردوند و برای این کار به درد نخوره : سرعت پایین (حجم زیاد هست)

صفحه ۱۱و۱۲

ماتریس **term-document**

تمام مکالمات که در سند وجود دارند را لیست کردند . یک ماتریس **boolean** برای مقدار **Brutus And Caesar But not Colpurnia** کافی است مقدار **Brutus** و **Caesar** را در هر سند (ستون) باهم **and** کنیم و بعد در هر سند (ستون) مقدار **colpurnia not** را با مقدار قبلی ، **And** کنیم .

به این ماتریس (کار) میگن **Pre-process** که باعث میشود هی سراغ اسناد نمیرویم ولی برای جواب دادن به یک **query** است.

صفحه ۱۴

این ساختار چند ایراد دارد : حافظه ی زیاد لازم دارد .

ماتریس پر از صفر هست.

صفحه ۱۶

Inverted index

Sparse matrix

صفحه ۱۸و۱۷

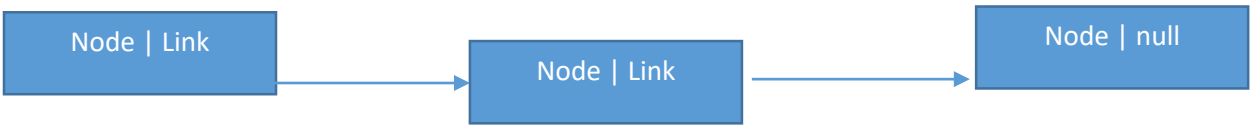
به جای ماتریس **term** و به ازای هر **term** (**brutus**) بگیم که در چه اسنادی وجود دارد

برای انجام این کار به اسناد یک شماره ی صحیح میدهیم **identifier**، از اسم اسناد استفاده نمیکنیم

شماره اسناد را میتوانیم تو یک آرایه ذخیره کنیم .

لیست میتونه یک لیست پیوندی باشد، کلاس باشد یا و ...

یاد آوری لیست پیوندی



خاصیت مشترک **term-material / inverted- index**:

برای جستجو در لینک لیست از **search** معمولی استفاده میکنیم دونه به دونه جلو میریم.

Posting list : sorted by id و **Dictionary** هر عدد یک **posting** است .

چرا این عدد ها **sort** هستند ؟ : برای این که با سرعت بالاتری **merge** کنیم و اشتراکات را پیدا کنیم **sort** انجام میشود

2=>4=>8=>16

1=>2=>3=>5=>8

اشاره گر اگه روی عدد کوچیکتر باشد جلو میرود اگر مساوی باشند هر دو اشاره گر جلو میروند . از **o(n+m)** هست.

بازیابی پیشرفته اطلاعات - جلسه ۲- ۲۷ اسفند

به لطف اکرم دیودل

گفتیم که ساختمان اصلی در داخل موتور جست وجو همین Inverted indexمی باشد که از دو بخش ساخته شده است الف:دیکشنری .که همین کلمات نیز می بیاشند ب:postings lists :برای ساخت Inverted index ما یکسری سند داریم که سند های ورودی سیستم ما نیز می باشند و باید این سند ها را یکی یکی باز کنیم و از ابتدا این index ها را بخوانیم تا جایی که به فاصله یا مکانمایی یا نقطه ای برسیم یک token را تشکیل دادیم که به این حرکت میگویند Tokenizer که کار مارا باید اینTokenizer باید انجام بدهد بعد streamباید جریانیاز tokenizer هایی که تولید کردیم داخل سندها و یکسری محاسبات را انجام می دهیم.indexها یکسری از کلمات یا tokenهاست که این مجموعه از کلمات میتواند دریک ساختمان داده مناسب ذخیره بشوند یعنی ما tokenهایی را در داخل ساختمان داده خود نگهداری میکنیم به همراه یک اشاره گر برای friend لیست اسنادی را نشان می دهد که friendدر داخل آنها است و Romanلیست اسنادی را نشان میدهد که Romanداخل آنهاست و به همین روال تا آخر.پس ما یک اشاره گر داریم که لیست پیوندی یا یک لیست است وکلا بستگی دارد به زبان برنامه نویسی ما که با چه زبانی برنامه نویسی می کنیم مثلا از پایتون .جاوا استفاده میکنیم مثلا جاوا یک ساختمان داده دارد به نام Vectorکه میتوانیم از این استفاده بکنیم که شبیه به لیست پیوندی و آرایه نیز میباشد از این جهت که سریع میتوانیم داده ای خود را در آن پیدا کنیم مثلا با indexمیتوانیم داده شماره یک یا داده شماره چهار و یا اینکه از این جهت مثل آرایه است.واز یک جهت میتوانیم ار vectorتا جایی که میتوانیم به آن داده اضافه کنیم مثل لیست پیوندی است حالا ساختار خاص و جایی دادرد که چطور ما میتوانیم یک ساختمان داده داشته باشیم که هم خاصیت آرایه داشته باشد و داده ها را نزدیک هم نگه داری کند و هم خاصیت لیست پیوندی را داشته باشد که ما اگر به هر تعدادی که بخواهیم به آن داده در آن ساختمان داده دخیره کنیم Normalization .

Initial Stages of Text Processing

مراحلی که اطلاعات داخل یک indexمی خواهند بشوند چه پردازش هایی روی آنها انجام میگیرد .

Tikenizetion: یا کلمه جمع کردن این stringها است

Normalization: همان حذف کردن حروف و تبدیل حروف به بزرگ و کوچک است

Stemming: کلماتی هستند که ریشه یکسان دارند که می تواند ریشه یکسان یک کلمه را پیدا کرد

Stop word: کلمات اضافی مانند ofو theو toو aو...که شاید معنی خاصی به ما ندهد در مورد محتوای سند می توان آنها را حذف کرد و در نهایت می توان indexرا ساخت

Indexer steps: Token sequence

حالا نحوه نگاه کردن indexهایی که ما ساختیم در عمل چگونه ساخته می شوند .فرض میکنیم index ی به ما دادن که یکسری سند در داخلindex ما قرار دارد که اگر سند ها را بگیریم چگونه می توانیم این indexرا بسازیم اگر هر کدام از ما یه نمونه متفاوت برای آن بزنیم یک راه خاص است که ما میتوانیم از این روش این indexرا بسازیم برای سند های توزیع شده اگر ار یک کامپیوتر استفاده میکنیم این راه حل بدرد میخورد. سند ها یکی یکی خوانده میشوند ولی این indexها اعمال می شوند Tikenizetion Normalizationو Stemmingو Stop word میشوند حال برای حل این index در هر مرحله یک شماره به آن اختصاص داده میشود که به این شماره ها میگوییم doc IDو وقتی که داریم سند ها را پردازش میکنیم که مثلا آن سند شماره اول است یا سند شماره Nیعنی سندی که در حال پردازش است ما شماره آن سند را میدانیم و بعد شروع میکنیم به TOKENکردن این سند .وقتی که یکی یکی این TOKEN ها را تشخیص میدهیم جلوی این سند شماره سند را هم می آوریم و در داخل یک لیستی آن را ذخیره میکنیم حالا فرض کنید داخل یه آرایه آنها را ذخیره میکنیم حالا این کار را انجام میدهیم تا اینکه سند شماره اول تمتم شود و می رویم طرف سند شماره دوم حالا همه این اتفاقات را برای سند شماره دوم رخ میدهد و شماره سند را بر روی آن مینویسیم و بعد می رویم به مرحله Sort وقتی ما این لیست را sortکردیم آنهایی که شبیه همدیگر هستند کنار یکدیگر قرار میگیرند فارغ ار آنجایی که داخل چه سندی هستند طبق مرحله sortاگر یک کلمه ای داخل چندین سند باشد به ترتیب هر جا شماره سندات به صورت کم یا زیاد مرتب میشوند ما یک sort را داخل ساختمان داده خوانده ایم که به آن Stable sort نیز میگویند .

Indexer steps:Dictionary and Postings

در اینجا چندین Termتقسیم می شوند به چندین دیکشنری و postingکه فقط اینجا یک documentو fre quency هم در اینجا برای ما اضافه کرده است یعنی سند هایی که داخل آنها چندین کلمه وجود دارد که تکرار در آنها مهم نیست فقط مهم اینجاست که یک کلمه در چند سند وجود دادر مثلا در دو سند رخ داده باشد .

Query processing : And

Queryهایی که داخل سیستم booleanمی آیند Querybooleanبه آنها میگویند یعنی بین کلمات یا andیا orیا not وجود دارد مثلا queryبه سیستیم داده ایم مراحلی که باید طی بشود این است که Brutusرا در داخل دیکشنری خود پیدا کنیم (دیکشنری یعنی مجموعه ای از کلمات که آیا دیکشنری ما یک درخت دودویی است یا یک آرایه است یا یک لیست پیوندی است که لیست پیوندی اینجا توصیه نمیشود)وقتی Brutusرا پیدا کردیم باید به اشاره گر آن دقت کنیم که به کدام لیست اشاره میکند و به همین روال کلمه Caesar را در داخل دیکشنری خود پیدا میکنیم

پس ما خود در اینجا دوتا لیست مرتب داریم که می آییم داخل این دوتا لیست اشتراک ها را پیدا کنیم ما دنبال سندی میگردیم که هر کلمهBrutus و هر کلمه Caesar در داخل آن باشد یعنی بین اعداد مشترک بین این دو لیست هستیم اگر ما ۳ تا andداشته دباشیم باید داخل هر ۳ تا لیست اعداد مشترک را پیدا کنیم اگر طول لیست Brutusبرابر با mباشد و طول لیست Caesar برابر باشد با nبا چه پیچیدگی زمانی می توانیم اشتراک ها اینها را پیدا کنیم O(M+N)

The merge

الگوریتم پیدا کردن اشتراک مثل الگوریتم mergeداخل merge sort است که همگی merge sortرا دیده ایم ولی هدف آنها یک چیز دیگر بود در آنجا دو تا لیست را باهم ادغام میکردند ولی شاید آنجا اعداد تکراری را حذف نمی کردند ولی اینجا ادعام نیست بلکه پیدا کردن اشتراک هاست ولی نکته جالب اینجاست که این merge به درد ما میخورد برای بازیابی Boolean

Boolean queries: exact match

می خواهیم در مورد مدل بازیابی صحبت بکنیم که مدل بازیابی booleanمیگوید همه چیز را باید با چشم Set میبینیم یعنی برای من مهم است که در دنیایی متن ها و بازیابی اطلاعات آن اسناد را بیابیم و بگیریم و انها را تبدیل بکنیم به یکسری موضوعیت ریاضی که در ریاضی چیزهای مثل مجکوعه یا set که من به محض اینکه سند خود را بگیرم وآن را به set تبدیل بکنم آن موقع می توانم محاسبات ریاضی را روی آنها انجام بدهم .این سیستم Boolean اسناد را به چشم set میبیند و آنها را تبدیل به set میکند که به این روش نیز bag of word نیز میگویند یعنی سندی از کلمات .یعنی سند را میگیریم و کلمات را تیکه تیکه میکنیم و همه آنه را می ریزیم داخل یک سند یعنی جای آنها داخل سند مهم نیست یعنی کلمات می توانند جا به جا شوند

Merging

یک وقت ما می بینیم query ما ترکیبی هستند یک جایی or داریم یه جایی NOT داریم و باید به آنها جواب بدهیم نکته ای که اینجا وجود دارد این است که من اگر دوتا پرانتز داشته باشم می گوییم بهتر است آن پرانتزی که طول اسناد داخل آن کوچکتر است را پیدا بکنم

مثال : X and Y and Z میگوییم بهتر است از آن کلمه ای استفاده بکنم که لسیت کوته تری دارند پس نکته ای که اینجا مهم است این است که حتی اگر در QUERE که Notوorو And در داخل آن داشته باشیم به فکر آن باشیم از آن لسیت های کوته تر شروع بکنیم اگر دوتا پرانتز داشته باشیم (X or Z) and (y and t) میگوییم آن پرانتزی که لیست کوتاه تری دارد از آن استفاده میکنیم

بازیابی پیشرفته اطلاعات - جلسه ۳- ۱۸ فروردین

به لطف مهین طاهری

Boolean queries: Exact match

- The Boolean retrieval model is being able to ask a query that is a Boolean expression:
  - Boolean Queries are queries using *AND, OR* and *NOT* to join query terms
    - Views each document as a set of words
    - Is precise: document matches condition or not.
  - Perhaps the simplest model to build an IR system on
- Primary commercial retrieval tool for 3 decades.
- Many search systems you still use are Boolean:
  - Email, library catalog, Mac OS X Spotlight

30

تا الان یک مدل بازیابی اطلاعات رو توضیح دادیم یعنی اگر یکسری سند به شما بدهند شما میدانید چطور یک سیستمی رو تهیه کنید تا بتوانید اطلاعات را از توی اون اسناد بازیابی کنید.این مدل بازیابی اطلاعات رو Boolean model یک مدلی بولی گویند.دلیل اینکه بهش مدل Boolean می کویند بخاطر query ها هست چون queryها بصورت بولی نوشته می شوند به این مدل میگویند مدل بولی.

Query booly یعنی چی؟! یعنی query هایی که عملگر های منطقی (or,not,and) در آنها هست.

XOY یعنی سندهایی که هم x هم y در آنها هست. X or Y یعنی سندهایی میخوام که یا x یا y در آنها وجود داشته باشد و not هم یعنی اسنادی که که در اون x نباشد.البته از ترکیبی از این عملیات ها همونطور که در منطقی خواندیم می توان یک query (بولی) منطقی پیچیده هم ساخت .

برای اینکه به این queryهای بولی پاسخ دهیم نیاز به یک ساختمان داده داریم که آن ساختمان داده inverted index (شاخص معکوس)هست.

نکته : این ساختمان داده در تمام موتورهای جستجو و سیستم های بازیابی اطلاعات وجود دارد .

آیا می شود یک سیستم بازیابی اطلاعات داشت که inverted index در آن نباشد؟ خیر

کدام ساختمان داده در تمام سیستم های بازیابی اطلاعات وجود دارد؟ inverted index

Phrase queries

- We want to be able to answer queries such as ***"stanford university"*** – as a phrase
- Thus the sentence *"I went to university at Stanford"* is not a match.
  - The concept of phrase queries has proven easily understood by users; one of the few "advanced search" ideas that works
  - Many more queries are *implicit phrase queries*
- For this, it no longer suffices to store only <*term* : *docs*> entries

چرا به آن inverted index می گویند؟ چون از term به docs (سند) میرسیم.

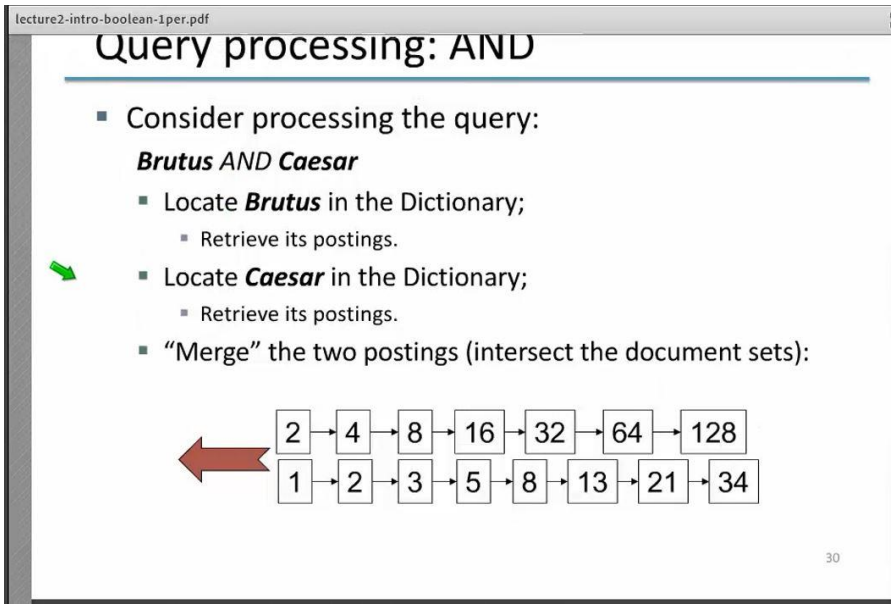
عموما سندهامون شامل term ها و کلمات هستند .وقتی میگوییم سند شامل چه کلماتی است نگاه میکنیم به سند که چه کلماتی داخلش هست.

اینکه یک کلمه را بدهیم و بگوییم این کلمه در کدام سند رخ داده است این نگاه وارونه و واژه رو به دنیاست تا الان می گفتیم یک سند شامل چه کلماتی است الان میگوییم یک کلمه در چه اسنادی رخ داده است. از این جهت بهش inverted (معکوس)می گویند .

اینکه می گویند موتور جستجوگر گوگل این تعداد صفحه رو شاخص گذاری کرده منظورش اینه که توی inverted index ش اضافه کرده است.

گفتیم که inverted index شامل دو قسمت است قسمت dictionary که شامل کلمات است و قسمت posting list لیست پیوندی که نشان دهنده ی اون id است .

وقتی از inverted index صحبت کردیم گفتیم که inverted index لیست ها رو مرتب نگه داری میکنه (شماره اسناد).اگر فرض کنیم دو تا posting list هستند شماره اسناد باید در inverted index مرتب باشندو به صورت مرتب ذخیره می شوند به این دلیل که بتوان محاسبات سریع تر انجام شوند مثل محاسبات AND مثلا query زیر سندهایی پیدا کنید که هم کلمه brutus و هم کلمه Caesar در آن باشد.



باید بریم لیست پیوندی یا posting list مربوط به Caesar را دربیاریم و همچنین posting list مربوط به brutus بعد اشتراک این دوتا لیست را محاسبه کنیم اشتراک ین دوتا لیست می شود اون سندی که هر دو کلمه داخلش هست .مثلا سند شماره۲ هم brutus و هم Caesar در اون وجود دارد برای اینکه اشتراک سریع تر محاسبه کنیم این ها مرتب میکنیم.

سو ا ل: اگر دوتا لیست مرتب نباشند پیچیدگی ما برای پیدا کردن اشتراک چقدر میشود؟(فرض کنید طول یک لیست m و طول دیگری n ست)؟ O(m*n)

اگر مرتب باشند؟ O(m+n)

به همین دلیل مرتب می شوند تا اشتراکشون را سریع تر محاسبه کنیم .

برای محاسبه اشتراکشون گفتیم که از ابتدای دو تا لیست شروع میکنیم دوتا مقدار را با هم مقایسه میکنیم اون لیستی که کمتر باشه را pointerش را جلو میبریم الان ۱ از ۲ کمتر است.۱ رو یکی می بریم جلو چون با هر

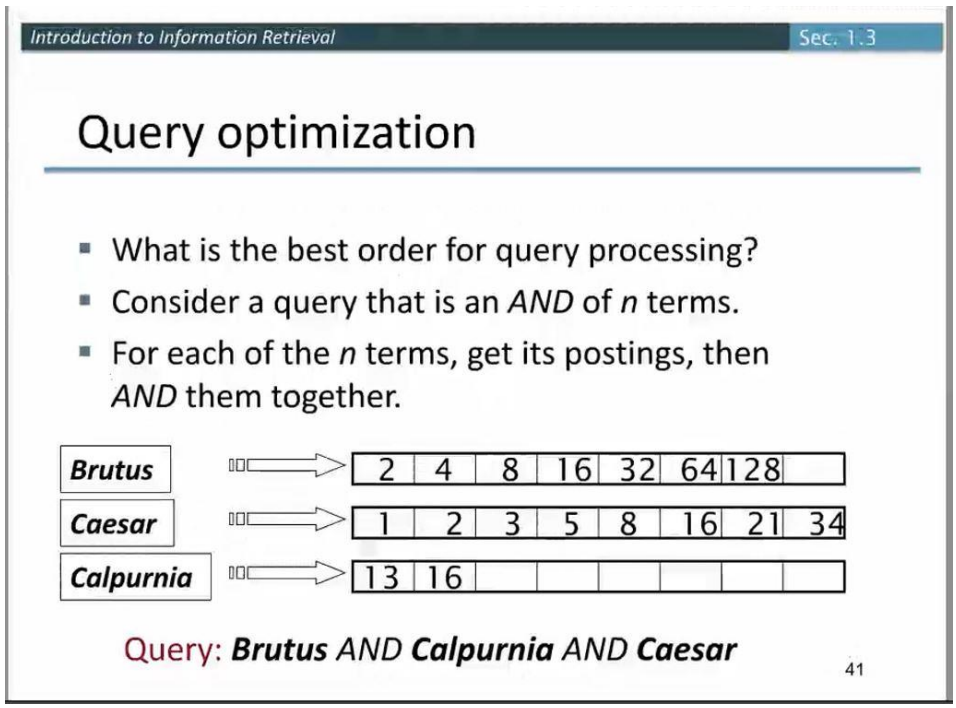
مقایسه یکی از عناصر این دوتا لیست کم می شود حالا یا از بالا یا از پایین یا بعضی وقتا هر دوتا مساوی باشد با هر مقایسه هر دوتا کم می شود .پس با جمع دوتا لیست مقایسه اشتراک را می توانیم حساب کنیم .

Intersecting two postings lists (a "merge" algorithm)

```
INTERSECT(p1, p2)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then ADD(answer, docID(p1))
 5             p1 ← next(p1)
 6             p2 ← next(p2)
 7        else if docID(p1) < docID(p2)
 8                then p1 ← next(p1)
 9                else p2 ← next(p2)
10  return answer
```

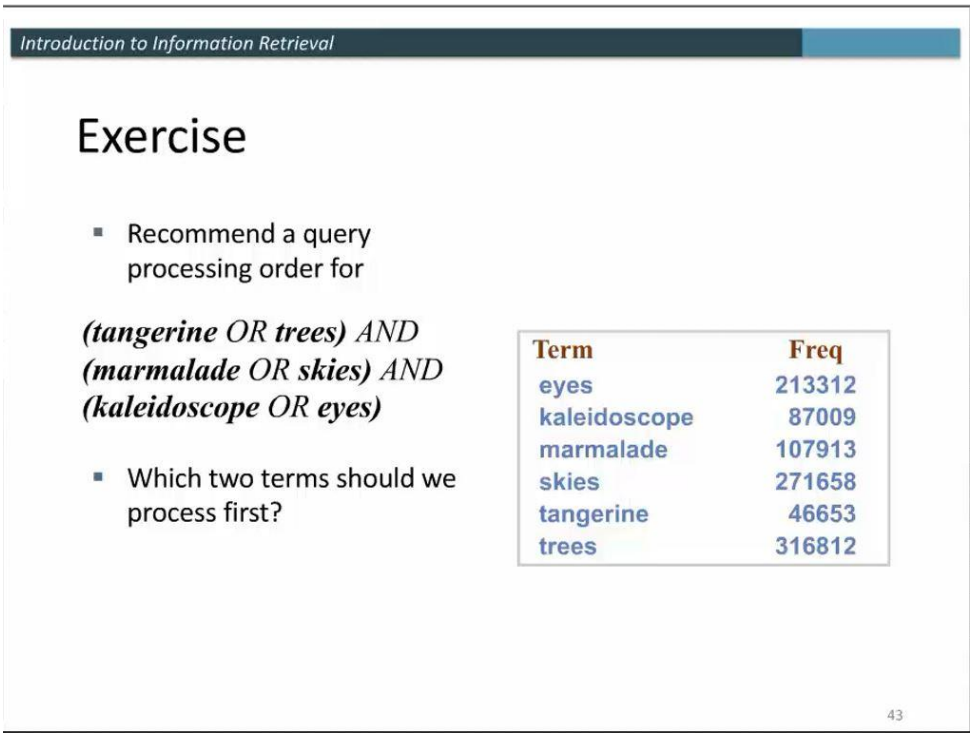
28

اسلاید بالا یک سودو کد برای پیدا کردن اشتراک دوتا لیست که AND می شود اگر or را بخواهید حساب کنید.باید اجتماع دوتا لیست را پیدا کنید.برای اجتماع هم حتما باید اشتراکات را حذف کنید.

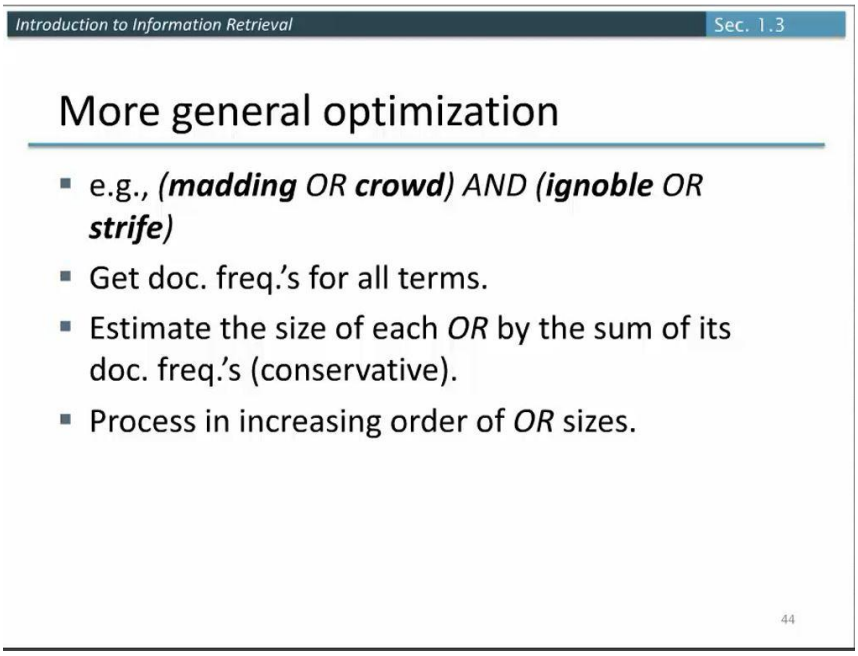


Query optimization کار خاصی انجام نمیدهد اگر زمانی شما ۳ تا کلمه دارید مثل caesar، brutus، calpurina و and اینها رو میخواهید حساب کنید اول از لیست هایی شروع کنید که طول کمتری دارند مثلا brutus و calpurina اول AND شون رو محاسبه کنید چرا به چه علت ؟ o(m+n) سرعت بیشتر می شود.

اول دو تا لیستی که از همه کوتاهترند را باهم merge کنید که از همه کوتاهترند .کد الگوریمی شبیه الگوریتم اینجا داشتیم که هافمن میگفتند. مثلا امیدواریم که brutus و calpurina اشتراک شون تهی باشد ،یه وقت calpurina که دوتا عنصر دارد با brutus که بیشتر دارد اشتراک شون از ۲ تا بیشتر که نمی شود زیر ۲ تاست .باز هم لیست حاصله لیست خیلی کوچکتری از این دو خواهد بود و اشتراک شون ه سریع تر گرفته خواهد شد .این یک اثبات شهودی است فقط بصورت زبانی اثبات کردیم بصورت ریاضی اثبات نشد که حدس میزنیم برای شما قابل فهم باشد.

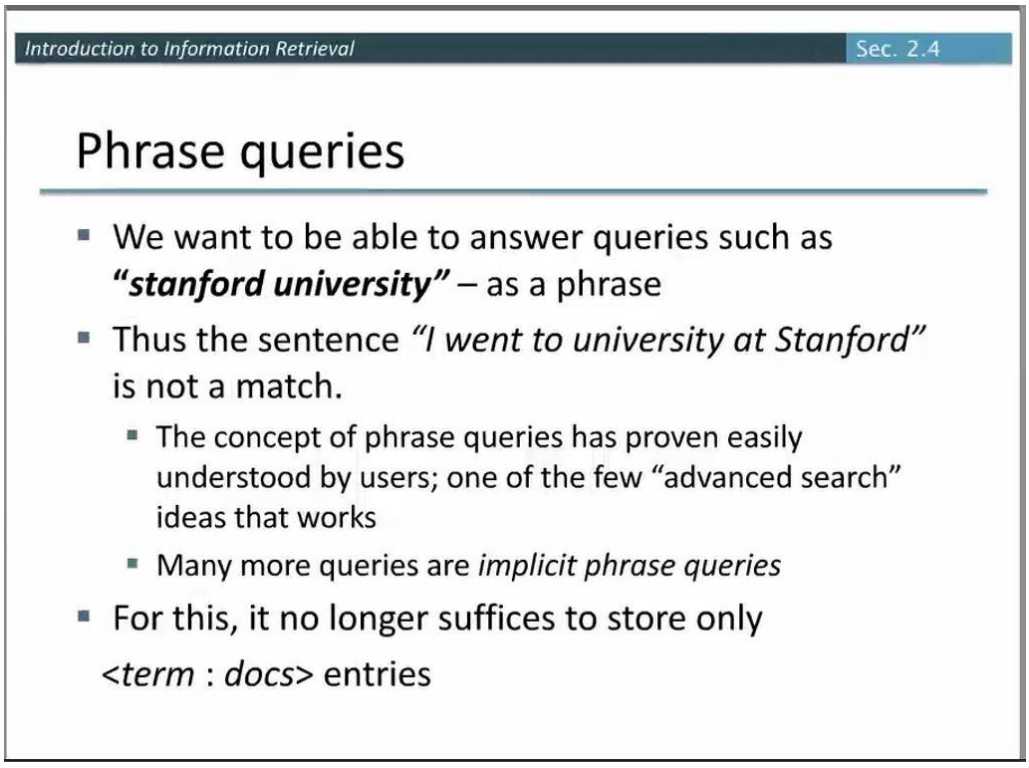


بر همین مبنا میگوید که recom بدهید که کدام or ها را حساب کنیم .دوتا AND داریم بین ۳تا پرانتز که Freq آنها مشخص است یعنی طول لیست شون معلوم است ...skies,trees, بنظر شما بهتر است کدام اول محاسبه شود؟



یکسری hint داده که اینجا میتوانید از این hint ها استفاده بکنید Freq هم بگیریداگر or داخل پرانتز هست فرض کنید طول کل لیست حاصل جمع دوتا می شود یک تخمین است estimate میزنید مثلا tangerine ۳۱۶۸۱۲ trees و ۴۶۶۶۵۳ بار رخ داده اند با هم جمع میکنیم می شود طول لیست اول ،لیست دوم را هم محاسبه میکنیم .از دوتا کمترها شروع می کنید به محاسبه کردن .

نحوه ساخت invert index را گفتیم قرار شد فقط یک بار اسناد متنی را بخوانیم هر خط فقط یکبار خوانده می‌شود هر متن یک بار ،در نتیجه یک بار کل اسناد را خواندید invert index را باید تحویل بدهید روش اینورتر ایندکس را اینطوری به شما گفتیم فقط یکبار خوانده می شود.



phrase queries

بعضی وقت ها query هایی هستند که در " " می گذاریم می خواهیم دو کلمه حتما کنار هم باشند مثل Stanford university ، information retrieval ، مثل search engine ما به اینها عبارت می گوییم حالا موتور جستجو بهش phrase query ، query های عبارتی می گوید.می خواهیم موتور جستجو را به صورتی تغییر بدهیم که بتوانیم phrase query هم پاسخ دهیم.بنظر شما چکار باید بکنیم ؟؟

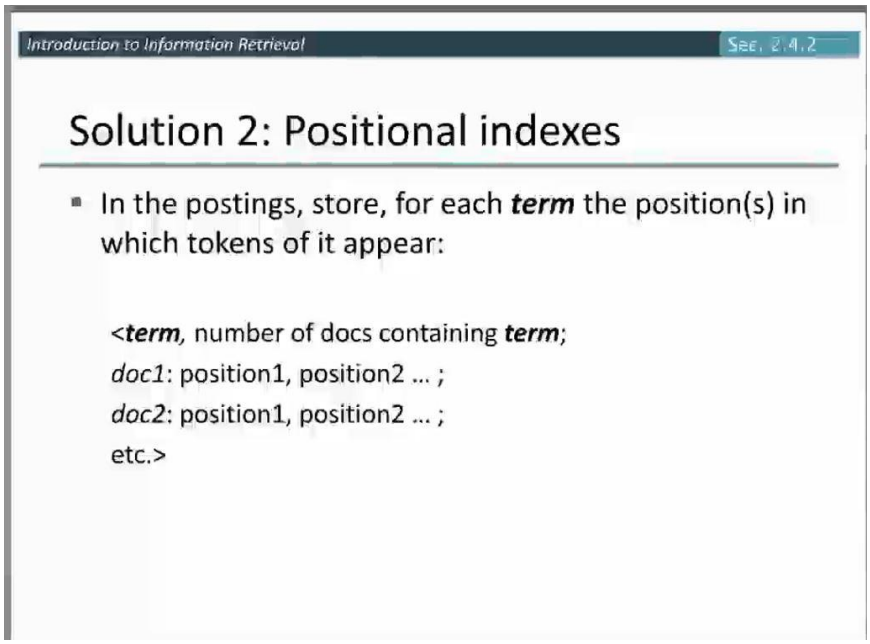
چطور از اون تکنیک قبلی استفاده بکنیم تا phrase query هم پاسخ بدیم؟

از ایتدایی و ساده ترین راه حل شروع میکنیم می توان بجای اینکه کلمات تکی رو مثل , we want answer query, want to,to be,be able و ... هر دو کلمه یی که کنار هم می آیند را توی شاخص مون ارائه کنیم (اولین راه حل)

چندتا ایراد دارد ۱- حجم ایندکس خیلی زیاد می شود ۲- با فرض اینکه بتوان phrase دو کلمه یی را حساب کرد با phrase های سه تایی چکار کنیم؟ آیا باید we want to، want to be ، to be able و .... هم نگهداری هم نگهداری دارد

این روش را می توان بهبود داد بگوییم اون دو کلمه هایی که خیلی معتبرن و معروفن و خیلی استفاده می شوند اینها را بتوان پیدا کرد (که توضیحات خاص خودش را دارد)پس زوش اول اینست که دنبال phrase هایی بگردیم که خیلی معتبر هستند index اضافه کنیم تا سریع تر بتوان به آنها پاسخ بدهیم دنبال محاسبات دیگر نرویم.

روش دوم : استفاده از همان مدلی که تا الان گفته شده است مدل Boolean . phrase را سرچ نمی کند Stanfordرا جدا سرچ میکند university هم جدا سرچ میکندسند هایی هم که هم Stanford و هم university در اونها هست را پیدا میکند فیلتر میکند بقیه را حذف میکند بعد اون سندهایی که Stanford و university در اونها هست را یکی یکی صفر تا صد بررسی میکند این روش هم یک مقدار وقت گیر هست .



روش دیگر استفاده از positional invert index است

Invert index که ما توضیح دادیم را باید ارتقا داد علاوه بر اینکه بگوییم یک کلمه در کدام سند ها هست باید بگوییم یک کلمه در کجاهای سندها وجود دارد فقط وجود داشتن یا نداشتن کافی نیست باید جاهای تکرارش هم به ما بگوید

اگر Invert index را طوری عوض کنید که شامل اینها باشد این solution هایی که گفتیم این solution خودش Invert index است شما توی Invert index تعداد ساعت هایی که شامل این term ها هستند هم می آید مثلاً این term داخل ۵ تا سند رخ داده یا ۲تا ID، اون سندها می‌آید

Doc2، Doc1 یعنی id اون سند .Doc2 یعنی سند شماره ۲ آیدی هرچی که هست بعد پوزیشن ها هم می آید شما یک مقدار دیگر هم می توانستید به این ساختار اضافه کنید جلوی Doc1 می‌توانستید tf1 بزارید یعنی بگوید در ساعت یک چند بار هم رخ داده است تعداد رخدادش را بنویسید بعد مکان هایش را بنویسید.

Positional index example

<*be*: 993427;
1: 7, 18, 33, 72, 86, 231;
2: 3, 149;
4: 17, 191, 291, 430, 434;
5: 363, 367, …>

Which of docs 1,2,4,5 could contain "*to be or not to be*"?

- For phrase queries, we use a merge algorithm recursively at the document level
- But we now need to deal with more than just equality

همون ساختاری که بالا گفتیم یک مثالش اینجاست ترم be در ۹۹۳۴۲۷ ظاهر شده است از جکله سند های ۱و۲و۴و۵ در گجای سند شماره ۱ ظاهر شده : 7,18,33,72,86,231 کجاهای سند ۲ و سند ۳ و الی آخر .

حالا که این position index را داریم می توانیم query هایی مثل to be or not to be یا phrase query ها رو پاسخ بدهیم با این position index هر phrase یی با هر طولی قابل محاسبه است.

TF : tem frequency است تعداد رخ داد یک کلمه در یک سند است .مثال be در سند ۱ چندبار رخ داده است

Processing a phrase query

- Extract inverted index entries for each distinct term: ***to, be, or, not.***
- Merge their *doc:position* lists to enumerate all positions with "***to be or not to be***".
  - ***to:***
    - *2*:1,17,74,222,551; *4*:8,16,190,429,433; *7*:13,23,191; ...
  - ***be:***
    - *1*:17,19; *4*:17,191,291,430,434; *5*:14,19,101; ...
- Same general method for proximity searches

حالا که این **positional invert index** را دارم میتوانم phrase یی مثل to be را پاسخ بدهم هم to را دارم هم be مثل حالت قبلی است در قبل هم می‌بایست and ها را پیدا می‌کردم این **;** را دقت کنید وقتی میاد یعنی سند بعدی این عدد اول یعنی تعداد رخداد عدد بعدی یعنی پوزیشن ها و موقعیت ها.

دقیقاً همون سیستم قبلی است باید اول از همه سندها یی پیدا کنیم که هم to در آنها باشد هم be با موقعیت ها کاری ندارید تا بریم سراغ مرحله بعد .پس ما حالت قبلی یک با دو مقایسه می‌شود یک با دو مساوی نیست یک میره جلو یک میشه ۴، ۴ با دو مقایسه می‌شود مساوی نیستند دو که کوچکتره میره جلو چهار با چهار مساوی است اینجا امیدواریم که این سند چهار پاسخ ما باشد از اینجا به بعد باید بریم سراغ پوزیشن ها اگر (پوزیشن to) x = باشد پوزیشن be باید x+1 باشد.

پس اگر to در موقعیت x باشد be باید در موقعیت x+1 باشد.

سوال اگر to در موقعیت x باشد be در موقعیت y باشد چه رابطه‌ای با باید بین x و y باشد تا بتوانیم بگوییم یک موقعیت مناسب پیدا کردیم ؟ جواب : y=x+1

موقعیت to + ۱ = y اگر شد می‌تواند یک to be باشد به محض اینکه ۴ دیدیم میری پوزیشنها را نگاه می کنیم

17> 8+1=9 مساوی نیستند

۱۶+۱=۱۷ =۱۷ با ۱۷ پایین برابر است به این معنی است که یک to be پیدا کردیم.

یعنی در سند شماره ۴ در موقعیت ۱۶ to دارد در موقعیت ۱۷ be دارد امیدواریم در موقعیت ۲۰ هم to داشته باشد که متاسفانه این اتفاق نیفتاده است از موقعیت ۱۶ پریده ۱۱۹ پس to be اینجا اتفاق نیفتاده است .

۱۹۱=۱+۱۹۰

۴ تا position جلوتر هم دوباره باید to ببینیم ۱۹۴ هم باید باشدکه متاسفانه نیست پس موقعیت ۱۹۰ موقعیت مورد نظر ما نیست .

۴۳۰=۱+۴۲۹

یک موقعیت احتمال هست جواب باشد چهارتا کاراکتر جلوتر ۴۳۳ است با ۴۳۴ هم این اتفاق می‌افتد پس to be در جاهای خودشان پیدا می‌شوند در موقعیت ۴۲۹ ولی باید مطمئن باشیم or not هم این وسط هست در نتیجه سراغ لیست or not هم می‌رویم اگر اونجا or not هم رخ داده بود که ما حاصل مون رو پیدا کردیم.

این توی پوزیشنها پریدن و نگاه کردن مثل همان الگوریتم سرچ قبلیمون است یعنی یک الگوریتم را دو بار صدا میزنیم فقط توی حالت اول آیدی ها را می‌گوییم باید مساوی باشید اگر ایدی سند بالا با آیدی سند پایین مساوی است که این سند جواب من است.

در پوزیشن ها ما باید +۱ بکنیم باید بدونید کدام را به علاوه یک بکنیم اونی که زودتر اومده (عددش کمتر است) تا برسید به اونی که دیرتر آمده است به این صورت ما می‌توانیم به query هایی positional هم پاسخ بدهیم.

Proximity queries

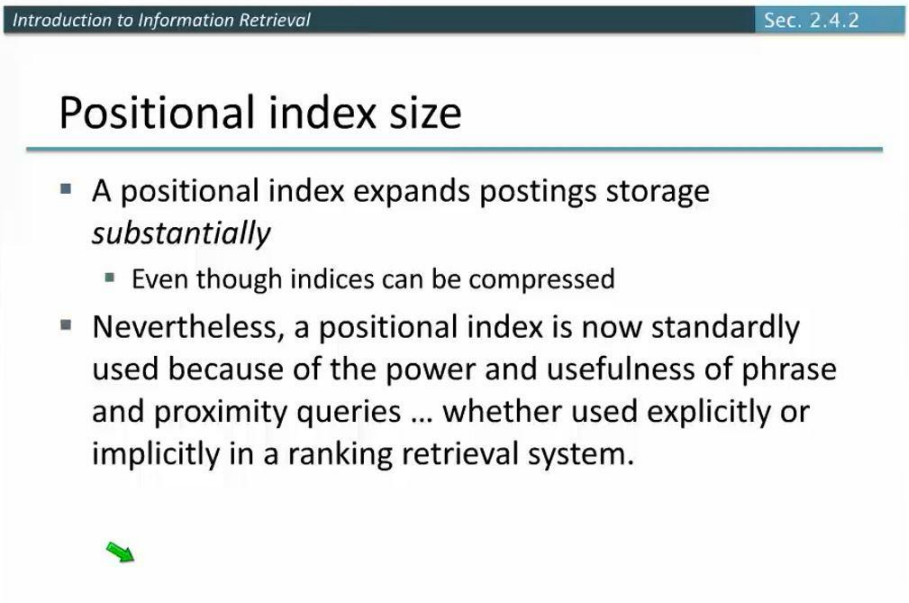
- LIMIT! /3 STATUTE /3 FEDERAL /2 TORT
  - Again, here, /*k* means "within *k* words of".
- Clearly, positional indexes can be used for such queries; biword indexes cannot.
- Exercise: Adapt the linear merge of postings to handle proximity queries. Can you make it work for any value of *k*?
  - This is a little tricky to do correctly and efficiently
  - See Figure 2.12 of *IIR*

حالا نکته ای که هست این است که وقتی من positional invert index داشته باشم (همان invert index است فقط position کلمات را توی index ذخیره کردیم)وقتی این رو داشته باشم علاوه بر phrase query می توانم proximity query را هم پاسخ بدهم .

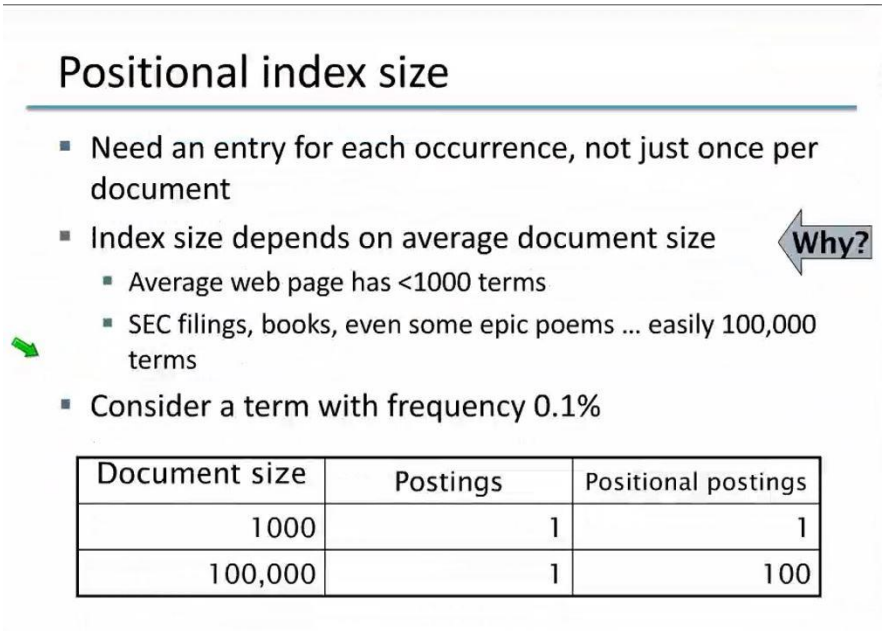
proximity query هم query های خیلی مهمی اند خیلی کاربردی اند شما یک وقت می گویید information retrieval اگر در کنار هم بودند با فاصله ۳ تا کلمه هم قابل قبول است یعنی ۳ تا کلمه بین شون باشه اشکال ندارد بصورت زیر نشون می دهد :

LIMIT! /3 STATUTE /3 FEDERAL /2 TORT

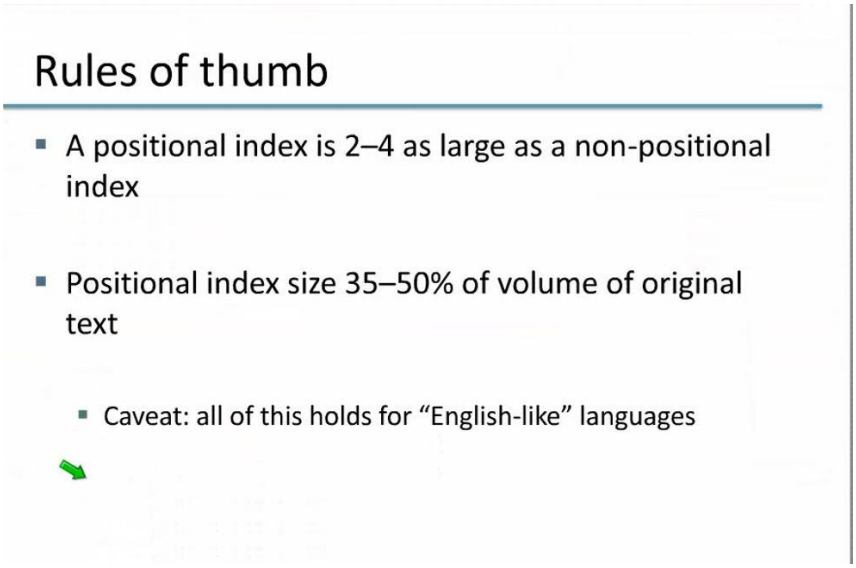
یعنی اگر ۳ تا کلمه هم بین شون باشه اشکال نداره و ۲ تا کلمه هم بین FEDERAL و TORT باشد اشکال ندارد اینطوری اون ۲و ۳ یی که می گیرید شما الگوریتم تون مشخص می شود مثلا اگر POSITION یکی شون x است باید x+3 <y باشد.جواب است البته باید یک مقایسه دیگر هم برداریم که بفهمیم که از چه مقداری مثلا x-3<y<x+3 شرط هاشو درست بکنید تا مطمئن شوید که این دوتا مثلا x+1=y یا x+2=y یا همین طور مقدار عددی که نیاز دارید بین اینها باشد را می توانید شما شرط ها تون را درست بکنید تا جستجو را انجام بدهید این proximity ها خیلی می توانند کمک بکنند به کیفیت جستجوتون .



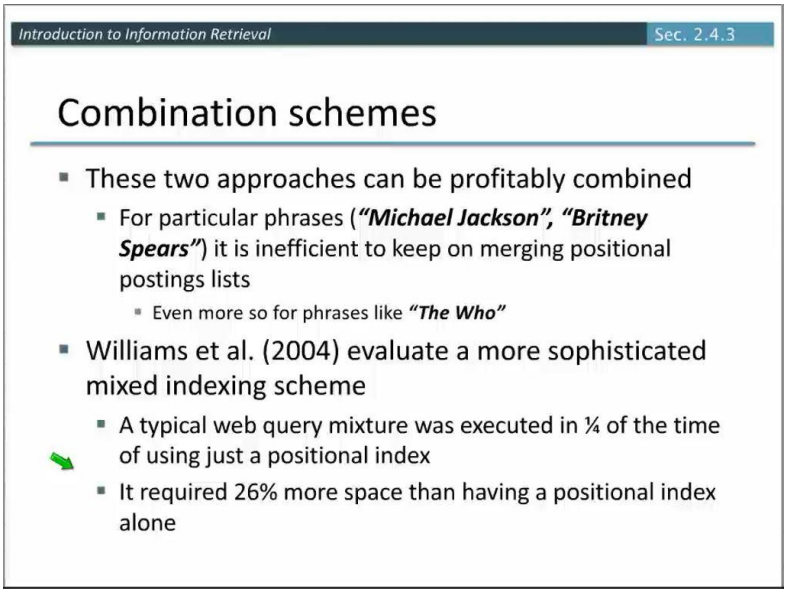
نکته ای که هست این positional index ها عموما حجم زیادی را میگیرند و در فصل های بعد یاد میگیریم invert index شان را فشرده بکنیم که حجم کمتری روی سیستم تون بگیرد.



یک برآورد کوچکی که بخواد داشته باشه می گوید که در این صورت اون میانگین طول اسناد ما تاثیر دارن روی این که positional index ما چقدر بشود.



حالا یک تخمینی تقریبا می گوید وقتی positional invert index بخواهید حدودا ۴ برابر کنید یک non-positional index ذخیره می شود و ضال میگیرد و عموما positional index ها نسبت به خود متن تون بین ۳۵٪-۵۰٪ اون اندازه متن جا می شوند خود متن ها کاراکتر دارند،کلمات ۸ یا ۹ کاراکتری دارد شما اون کلمات رو حذف می کنید فقط position شون را نگه می دارید . positionl شون هم که دیگه اعداد integer است و اعداد int مشخص است تعداد حجم مشخصی را نگهداری می کند.



از روش هایی که گفتیم برای phrase query ها بخواهید پاسخ دهید می توانید یک روش ترکیبی هم استفاده بکنید روش ترکیبی یعنی اینکه من همون phrase هایی که خیلی تکراری هستند مثل information retrieval و search engine ، Stanford university و .. اینها رو مستقیم در index تون بیارید که اگر پیدا شدند خیلی سریع سراغ position نرید اگر پیدا نکردید برید سراغ position ها .(آخری مطلبی که توی این قسمت بیان کرده)

پایان فصل اول -مدل بازیابی بولی به همراه phrase query ها

چند نکته!!

بولین که دارید بازیابی میکنید یک مدل بازیابی است که مبحثش bag of set هم می گویند شما وقتی که کل سند تون رو بگیرید و این سند را تبدیل به یکسری term (کلمه) بکنید به صورتی که ترکیب کلمات بهم بخوره بصورتی که تعداد تکرارشون دیگه وجود نداشته باشد فقط وجود داشتن و وجود نداشتن شون رو توی اسناد نگاه بکنید انگار که این سند تون را با دید یک مجموعه نگاه می کنید یک set دارید نگاه میکنید .

می خواهیم بگوییم که مدل بولی وقتی کار میکند که شما دارید سندهاتون را بصورت مجموعه می بینید بصورت set می بیند بهش bag of word می گوین سبدی از کلمات ،کلمات ر ریختی توی یک سبد و روی آن دارید پردازش انجام می دهید.

این مدل بولی چالش های خاص خودش را دارد وقتی سندها رو با دید مجموعه میبینید چالش های خاص خودش را دارد اولین چالش این است که تعداد رخ داد و تعداد تکرار کلمات توش دیده نمیشود

اینکه فقط می‌گویند یک کلمه در یک سند هست یا خیر. وقتی تعداد تکرار نباشد خود به خود شما اهمیت سند ها را رو هم نمیتونید محاسبه کنید کدوم سند خیلی بهتر است کدام سند ضعیف تر است . وزن به سندها آن نمی‌توان داد در نتیجه سند ها را نمی توانید مرتب کنید بر اساس اهمیت شون و خوب بودنشان برای کوئری های شما.

Query های بولین فقط yes و no جواب می‌دهند این سند query را پوشش می‌دهد این سند query را پوشش نمی‌دهد منطقی است. این سند justify می‌کند و اون query ما را ارضا می کند یا این query را اصلا پوشش نمی دهد.خب تو این فضا اگر نتوانید مرتب سازی بکنید یکدفعه اگر ۱۰۰ هزار تا سند برای شما برگرداند هیچ ترتیبی بین این سندها نمی توان قائل شد.

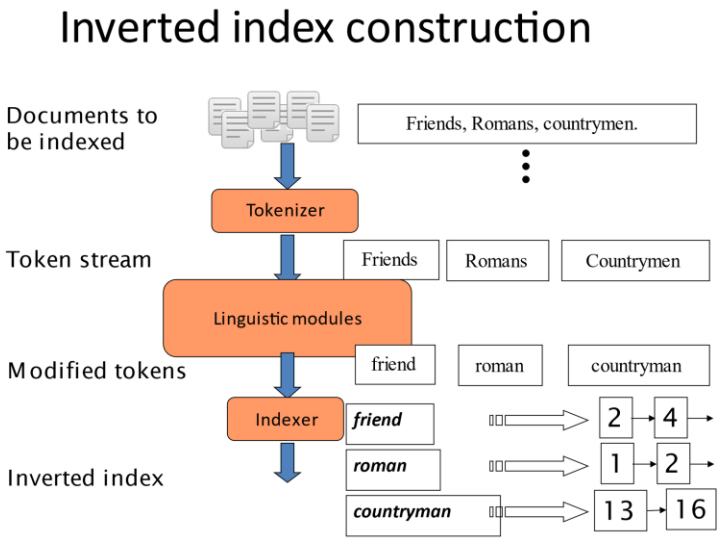
همه سندها دو تا کلمه را دارند اما کدام سند بهتر است توی وب دارید جستجو می کنید تعداد زیادی صفحه می آید همه هم اون دو تا کلمه را دارد که هیچ عاملی (پارامتری) وجود ندارد که این مدل بازیابی بفهمه کدومشون بهتر است هیچ حدسی نمی‌تواند بزند (مدل بولین) این نقض اساسی این مدل بازیابی اطلاعات است.

ما باید پارامترهای دیگری (بچه ها پرسیدند) به عنوان TF، ID و غیره باید پارامترهای بیشتری را در نظر بگیریم اطلاعات بیشتری استفاده بکنیم تا بتوانیم مدل های بازیابی اطلاعات قوی‌تری داشته باشیم. توی بحث بعدی که با هم خواهیم داشت آنجا سند ها را ما دیگه به شکل مجموعه نمی‌بینیم . به شکل بردار می‌بینیم هر سندی که به ما می‌دهند تبدیل میکنیم به یک بردار.

توی دنیای بردارها وارد خواهیم شد و محاسبات برداری به ما کمک می‌کند تا بتوانیم این سندهای مرتب را پیدا کنیم اینجا دیگه query تبدیل خواهد شد به بردار، فرق بردار با مجموعه این است که ترتیب بین کلمات مهم است تعداد تکرار مهم است که هر کلمه چند بار رخ داده و به یک مدل های بازیابی با کیفیت بهتری خواهیم رسید که آنها دیگه به ما rank می‌دهند به ما وزن می دهند و می توانیم سند ها را بر حسب این که کدامشان مرتبط تر و مناسب تر هست مرتب و دسته بندی بکنیم و به کاربر نشان بدهیم.

تمرین

برنامه ی بنویسید که مجموعه ای از اساد متنی را بگیرد و positional invert index را تولید کند به شکل اسلاید زیر



Term : تعداد سندهایی که term داخلشون هست

Normalize کردن : حروف بزرگ به کوچک تبدیل کردن

; , و .. حذف شوند

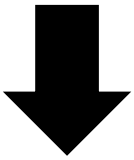
Stopword ها حذف می شوند

نحوه ساخت invert index توضیح داده شد

هر کلمه را در یک آرایه اضافه کنید با شماره ID ،بعد مرتب کنید بعد در نهایت invert index بسازید.

جلسه ۴

به لطف ساناز نیکخواه



فصل دوم

ترم وکبلری و پستینگز لیستز (The term vocabulary and postings lists)

به طور کلی در هر فصلی از کتاب و اسلایدها یک بخش ریکپ Recap در ابتدای فصل هست که به مرور و بررسی خلاصه و توضیحات فصول گذشته می‌پردازد که ما در این کلاس از آنها گذر می‌کنیم. این فصل هم همین کار را می‌کنیم.

ترم وکبلری The Term Vocabulary

ما تا به حال فقط در مورد بازیابی بولی به صورت ساده صحبت کرده‌ایم. یعنی تا به حال دو فرض اساسی داشته‌ایم:

- دقیقا سند را می‌شناسیم و یک سند مشخصی در اختیار ما است.
- دقیقا می‌دانیم ترم چیست و مشخص است و مثلا گفته‌ایم آنهایی که با فاصله از هم جدا شده‌اند ترم یا توکن هستند.

اما در واقعیت هر دوی این موارد می‌توانند بسیار پیچیده باشند و به این سادگی نیستند. مقدار خیلی کمی‌در مورد مشکلات اسناد صحبت می‌کنیم و تاکید اصلی‌مان را روی مسائل مربوط به ترم‌ها می‌گذاریم.

مشکلات اسناد

قبل از این که به مشکلات ترم‌ها بپردازیم باید دو مبحث اساسی فرمت و زبان را پوشش دهیم. پاسخ اینکه سند ما چه فرمتی است و آیا PDF یا Doc یا XML و ... است و تمام فرمت‌های مختلف مربوط به نرم‌افزارهای مختلفی که داریم را باید پیدا کنیم. همینطور باید پاسخ اینکه زبان محتوای ما چیست و آیا چینی یا ژاپنی یا کره‌ای و... است را پیدا کنیم که البته برای ما این سه زبانی که گفتیم خیلی شبیه هستند و البته کره‌ای یک مقدار کمی‌متفاوت است. همچنین ما در یک زبان ممکن است مجموعه کاراکتر یا کاراکتر ست‌های مختلفی داشته باشیم.

(خاطره از استاد: یک دانشجوی ارشد یک پروژه ای داشت و برای موتو جستجو کار می‌کرد و یک کاری به وی داده بودند که به این شکل بود که یک لینک را قبل از دانلود، بتواند حدس بزند محتویاتش از چه زبانی است. این کار را از کانتکست دور و بر لینک و اینکه این لینک در چه HTML ای هست و خود URL لینک و دامنه که از دات ir است یا چه کشور دیگری و ... با کنار هم قرار دادن این موارد می‌توانست حدس بزند که این زبان زبان فارسی است و می‌صرفد که دانلودش کنیم و یا زبان دیگری است و دانلودش نمی‌کنیم. پس این بحث می‌تواند حتی در حد پروژه کارشناسی ارشد پیچیده باشد.)

این دست مسائل را و به خصوص تشخیص زبان را با تکنیک‌هایی مثل کلسیفیکیشن حل می‌کنیم. مثلا چند سند فارسی را به ما می‌دهند و می‌گویند این‌ها نمونه فارسی است و حالا سندهای دیگری که می‌آیند را کلسیفای می‌کنیم و به زبان‌های مختلف دسته بندی می‌کنیم. روش کلسیفیکیشن به همین دلیل مساله مهمی‌است و یکی از کاربردهای مهمش همین رفع مساله تشخیص زبان است. که البته در فصول بعد بیشتر به آن می‌پردازیم.

باز هم ممکن است پیچیدگی‌های بیشتری داشته باشیم. مثلا یک سند ممکن است شامل زبان‌های مختلفی و فرمت‌های مختلفی باشد. مثلا:

- یک ایمیل به زبان فرانسه با یک اتچمنت PDF به زبان اسپانیولی.

یکی دیگر از مشکلات ما این است که واحد سند چیست و سر و ته سند کجاست؟ مثلاً:

- یک فایل؟
- یک ایمیل؟
- یک ایمیل با ۵ اتچمنت؟
- یک گروه از فایل‌ها (مثل ppt پاورپوینت یا latex یا HTML)؟

اگر لتک latex کد زده باشیم یا محتوی‌مان را تایپ کرده باشیم می‌دانیم که یک سند را با چندین فایل کنار هم می‌سازد.

تمام این نکات، فضای واقعی و مشکلات واقعی در مورد مشکلات اسناد را بهتر نشان می‌دهند. البته ما در این درس از این مشکلات صرف نظر می‌کنیم و فضا را خیلی راحت می‌بینیم و می‌گوییم فرض کنید هر فرمتی هم که دارید، خیلی راحت تبدیل شده به تکست و متن ساده و تصاویر و دیاگرام‌ها و فونت و فرمت و لینک و ایمیل و ... هرچیز دیگری حذف شده و متن خالی در اختیار دارید. از این جهت این کار را می‌کنیم که کارمان ساده‌تر شود. البته نکته‌ای که هست این است که اگر مثلا روی PDF بخواهیم کار کنیم، لایبرری‌هایی هستند در اینترنت که خیلی راحت PDF ما را می‌گیرند و متن ساده‌اش را پس می‌دهند. و در هر زبانی هم که کد بزنیم (مثل جاوا یا پایتون یا سی و ...) و با هر زبانی که برنامه نویسی می‌کنیم این لایبرری‌ها موجود هستند و فرمت‌های مختلف را برای ما به همدیگر تبدیل می‌کنند.

مشکلات ترم‌ها (مشکلات رایج در همه زبان‌ها به طور عمومی‌و همینطور در زبان‌های غیر انگلیسی)

اسلاید زیر دارد سعی می‌کند چهار تعریف مختلف را توضیح دهد که ما نیازی به همه آنها حس نمی‌کنیم.

Definitions

- Word – A delimited string of characters as it appears in the text.
- Term – A "normalized" word (case, morphology, spelling etc); an equivalence class of words.
- Token – An instance of a word or term occurring in a document.
- Type – The same as a term in most cases: an equivalence class of tokens.

دانشجو فقط تفاوت ترم و توکن را بداند از نظر استاد کافی است. دو مورد دیگر دارند کمی‌پیچیدگی را زیاد می‌کنند. ترم و توکن را مرور می‌کنیم.

توکن هر تکه‌ای است که در سند موجود است.

ترم آن است که ما در توکن‌ها بیاییم تکراری ها را حذف کنیم و به باقیمانده آنها ترم بگوییم. البته ترم باید نرمالایز (حروف بزرگ به کوچک تبدیل شده باشند، شکلهای مختلف مواردی مثل s جمع را حذف کنیم و مفردش کنیم و اگر غلط املایی دارد اصلاح کنیم و ...) هم شده است.

توکنایز کردن و این که یک سند چند توکن یا ترم دارد را با مثال زیر مرور می‌کنیم.

In June, the dog likes to chase the cat in the barn. – How many word tokens? How many word types?

در عبارت فوق کلمات in و the هر کدام دو و سه بار توکنشان تکرار شده. ولی هرکدام یک ترم محسوب می‌شوند. یعنی اینجا در مجموع ۱۲ توکن داریم و ۹ ترم.

جایی گه می‌خواهیم اینورتد ایندکس بسازیم، برای ساختن ترم از توکن یکی از کارهایی که قبلا هم گفتیم که باید انجام دهیم، نرمالایز کردن است. نرمالایز کردن یعنی همه حروف را تبدیل کنیم به حروف کوچک و s جمع را برداریم و از ریشه کلمه استفاده کنیم و ... . این نرمالایز کردن یک پری‌پراسسی است که باید روی متن انجام دهیم تا بتوانیم توکن‌هایی که کاندید هستند برای اضافه شدن به دیکشنری را تشخیص دهیم. نمونه این کار را در اسلاید زیر می‌بینیم.

Tokenization: Recall construction of inverted index

- Input:
  Friends, Romans, countrymen. | So let it be with Caesar | ...
- Output:
  friend | roman | countryman | so | ...
- Each token is a candidate for a postings entry.
- What are valid tokens to emit?

پس از تشخیص توکن‌ها، نرمالایزیشن در هر زبانی هم که باشیم باید انجام شود که کار سختی هم نیست و گفتیم باید لورکیس Lower case کنیم و حروف بزرگ را به حروف کوچک تبدیل کنیم. نکته دیگر این است که اگر کلمات با فرمت‌های مختلفی نوشته می‌شوند به یک فرمت تبدیلشان کنیم. مثلا USA و U.S.A هر دو یک مفهوم هستن و به ایالات متحده امریکا اشاره می‌کنند و ما باید این را با یک فرمت قراردادی دلخواه بنویسیم. حتی اگر دلمان بخواهد آن را به xxx تبدیل می‌کنیم ولی مهم این است که همه فرمت‌ها به یک فرمت قراردادی نوشته شوند. که البته معمولا در اینجا سعی می‌کنیم فرمتی را که کاربر هنگام سرچ به احتمال زیاد در کوئری استفاده می‌کند ما هم به عنوان فرمت اصلی در نظر بگیریم. به این فرمت‌های متفاوت و هم‌ارز عنوان ایکوئیولنس کلس equivalence classes یا کلاس هم‌ارزی می‌دهیم. می‌گوییم این‌ها با هم هم‌ارز هستند و هرجا در سند یکی از این نمونه‌ها را دیدیم، به جایش هم‌ارز و نماینده‌اش را قرار می‌دهیم. این کار را به

این دلیل انجام می‌دهیم که اگر کاربر یکی از نمونه‌های کلاس هم‌ارزی را سرچ کرد ما بتوانیم نمونه‌های دیگر را هم در جواب به او برگردانیم.

بعضی جاها این کلاس‌های هم‌ارزی مشکل‌تر از مثال فوق هم می‌توانند باشند. مثلا Windows و windows دو کلاس متفاوتند و اگر سیستم IR ما بتواند این را تشخیص دهد یعنی خیلی قدرتمند است ولی در عین حال کارایی کمتری دارد. چون تشخیص این موارد نیاز به محاسبات بیشتری دارد و سیستم را کند می‌کند.

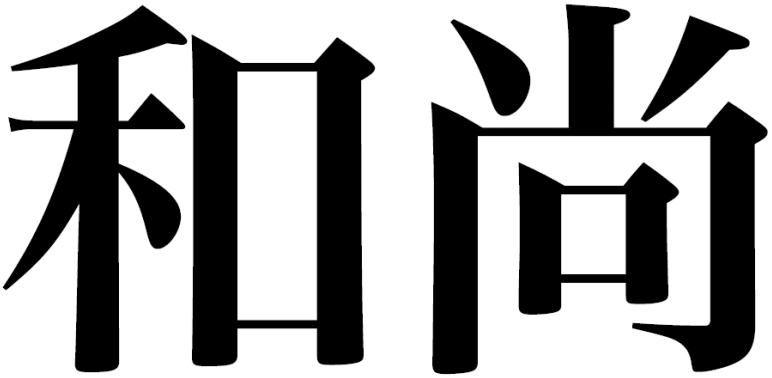
جلسات پیش راجع به سختی‌های این کار صحبت کرده‌ایم. باز هم مرور می‌کنیم. به عنوان مثال موقع توکنایزیشن شاید آپستروف را جزو جدا کننده‌های دو کلمه در نظر بگیریم. اما برای بعضی کلمات مانند O'Neill یا O'Reilly نباید این کار را بکنیم چون اینها کل‌شان یک کلمه است و سرهم هستند.

یا مثلا در مورد کامپاندها و ترکیبات و کلمات دو تکه‌ای مثل Hewlett-Packard یا چند تکه‌ای مثل State-of-the-art که چهار کلمه‌ای است یا co-education یا data base (که البته در اینجا کتاب شورش را درآورده و database سرهم نوشته می‌شود) یا San Francisco یا Los Angele-based company و ... که کلماتی هستند که خود از چند کلمه ساخته شده‌اند و اینکه ما در اینجا اسپیس و فضای خالی و خط تیره را توکنایز کننده و جدا کننده توکن‌ها در نظر بگیریم یا نه خود قابل بررسی است و جای بحث دارد.

در مورد تاریخ‌ها و اعداد و اینکه اصلا این‌ها را توکنایز بکنیم و در ایندکسمان قرار دهیم یا خیر هم باید بررسی و بحث کنیم. یا حتی ترکیب حرف و عدد مثل B-52 که اسم خاص است و نام یک بمب افکن است. یا مثلا عدد IP که خود از ۴ عدد تشکیل شده. یا مثلا انواع فرمت شماره تلفن‌ها. گوگل همه اینها را ایندکس می‌کند. سیستم‌های بازیابی قدیمی‌معمولا اعداد را ایندکس نمی‌کردند. در حالیکه در اکثر مواقع اینها فیچرهای مفیدی هستند.

چالش دیگری که در بعضی زبان‌ها مثل چینی داریم این است که اصلا فضای خالی بین کلمات نیست و همه پشت سر هم هستند و حتی اوضاع وقتی پیچیده‌تر می‌شود که دو شکل در زبان چینی جدا در نظر گرفته شوند هرکدام یک معنی و با هم در نظر گرفته شوند یک معنی دیگر می‌دهند. به مثال زیر دقت کنید:

Ambiguous segmentation in Chinese



The two characters can be treated as one word meaning ‘monk’ or as a sequence of two words meaning ‘and’ and ‘still’.

در مثال فوق دو کاراکتر را می‌توان یک کلمه دید به معنی راهب و یا به صورت یک توالی دیدشان به معنی «و هنوز». چنین ابهاماتی را در زبان فارسی هم تا حدودی مشابه‌اش را داریم. مثلا مادر یک کلمه است و «ما در» دو کلمه که البته در اینجا با فاصله جدا می‌شوند. این نبودن فاصله در زبان‌های دیگری هم مانند آلمانی وجود دارد. ترکیبات در زبان آلمانی به صورت یک کلمه سر هم بزرگ نوشته می‌شوند. و ما بالاخره در توکنایزیشن و در سیستم پردازشی‌مان باید بتوانیم با این موارد کنار بیاییم. در اسلاید زیر مثالهای دیگری از این مورد می‌بینیم.

Other cases of "no whitespace"

- Compounds in Dutch, German, Swedish
- Computerlinguistik → Computer + Linguistik
- Lebensversicherungsgesellschaftsangestellter
- → leben + versicherung + gesellschaft + angestellter
- Inuit: tusaatsiarunnanngittualuujunga (I can't hear very well.)
- Many other languages with segmentation difficulties: Finnish, Urdu, ...

از دیگر مواردی که عنوان کردیم یکی هم این است که ممکن است در یک سند بیش از یک زبان یا بیش از یک الفبا داشته باشیم. در مثال زیر ۴ الفبا وجود دارد.

Japanese

ノーベル平和賞を受賞したワンガリ・マータイさんが名誉会長を務めるＭＯＴＴＡＩＮＡＩキャンペーンの一環として、毎日新聞社とマガジンハウスは「私の、もったいない」を募集します。皆様が日ごろ「もったいない」と感じて実践していることや、それにまつわるエピソードを８００字以内の文章にまとめ、簡単な写真、イラスト、図などを添えて１０月２０日までにお送りください。大賞受賞者には、５０万円相当の旅行券とエコ製品２点の副賞が贈られます。

4 different "alphabets": Chinese characters, hiragana syllabary for inflectional endings and function words, katakana syllabary for transcription of foreign words and other uses, and latin. No spaces (as in Chinese).

موضوع دیگری که باید مورد توجه قرار بگیرد از راست به چپ نوشتن است. فارسی و عربی از راست به چپ نوشته می‌شوند. در حالیکه اعداد در این دو زبان از چپ به راست نوشته می‌شوند و در یک جمله شامل عدد و کلمات، دو جهت حرکت خواهیم داشت. یا اگر مخلوطی از فارسی وانگلیسی داشته باشیم هم مشکل ساز می‌شود.

از مشکلات دیگر در زبان‌های غیرانگلیسی اکسان‌ها و علایم روی حروف و حرکت‌ها است که برای نرمالایزیشن معمولا این‌ها به سادگی حذف می‌شوند.

از دیگر مشکلات، ادغام‌ها و اوملات‌ها هستند که برای نرمالایزیشن حروف خاصی را می‌توان با ترکیبی از حروف دیگر جایگزین کرد.

- Accents: résumé vs. resume (simple omission of accent)
- Umlauts: Universität vs. Universitaet (substitution with special letter sequence "ae")

در نهایت باید سعی کنیم مهمترین حالت یک کلمه را در نظر بگیریم. یعنی آن حالتی را که به احتمال زیاد کاربر در کوئری خود به کار می‌برد.

مثلا در زبان‌هایی که در حالت استاندارد از علایم و اکسنت‌ها و حرکت‌ها استفاده می‌شود، کاربران معمولا آنها را تایپ نمی‌کنند.

از دیگر مواردی که باید به آن توجه کرد برهم‌کنش تشخیص زبان و نرمالایزیشن است. به مثال زیر دقت کنید.

- Normalization and language detection interact.
- *PETER WILL NICHT MIT.* → MIT = mit
- *He got his PhD from MIT.* → MIT ≠ mit

در زبان آلمانی می‌توان MIT را معادل mit و هم‌ارز دانست اما در انگلیسی نمی‌شود و MIT نام خاص یک دانشگاه است.

مشکلات ترم‌ها (مشکلات زبان انگلیسی)

از همه این مواردی که مربوط به زبان‌های دیگر است عبور کنیم و برگردیم به زبان انگلیسی.

اولین کاری که انتظار داریم به طور اتوماتیک روی توکن انجام دهیم برای نرمالایز کردن، کیس فولدینگ Case folding است. یعنی تمام حروف کلمه را اگر بزرگ هستند برگردانیم و به صورت کوچک بنویسیم.

دومین کاری که بعد از تشخیص توکن باید انجام دهیم این است که باید تشخیص دهیم که آیا این توکن استاپ ورد Stop word هست یا نه. ما کلماتی را که به درد نمی‌خورند را در ایندکس نباید اضافه کنیم. مثلا to, be, of, in, a, which, would و حروف اضافه و امثال این‌ها که کلماتی هستند که هیچ مفهوم مستقلی را به ما نمی‌رسانند و اگر در یک سند باشند یا نباشند در هر حال در باره معنی سند حرفی نمی‌زنند. این‌ها را در ایندکس نباید اضافه کرد و برای همین نام Stop word به این مجموعه داده شده که یعنی اضافه‌شان نکنید و فیلترشان کنید. نحوه این فیلتر کردن به این شکل است که بعد از تشخیص توکن در متن، می‌آییم آن را در یک لیست از قبل آماده شده استاپ وردز (که در اینترنت نمونه‌های مختلفی از آن موجود است. اگر جستجو کنید برای عبارت stop words list نتایج مختلفی را می‌بینید که افراد مختلفی تهیه کرده‌اند و معمولا شامل حدود ۲۰۰ الی ۳۰۰ کلمه می‌شود و به صورت تکست و متن است و لیستی از کلمات است.) بررسی می‌کنیم و اگر توکن جزو لیست بود به دیکشنری اضافه نمی‌شود و اگر جزو لیست نبود به دیکشنری اضافه می‌شود.

البته این حذف کردن استاپ وردها در سیستم‌های بازیابی اطلاعات قدیمی‌روش استاندارد بود و به عنوان مثال الان گوگل آنها را هم ایندکس می‌کند. چون می‌خواهد دقت بالایی ارائه دهد.

مخصوصا وقتی می‌خواهیم فریز کوئری‌هایی را مثل King of Denmark پاسخ دهیم، با حذف حرف اضافه دیگر نمی‌توانیم به این کوئری پاسخ دهیم. از این جهت است که دیگر زیاد جالب نیست که استاپ وردها را حذف کنیم، به خصوص وقتی با فریز کوئری سر و کار داریم.

در حال حاضر بیشتر موتورهای جستجوی تحت وب این کلمات استاپ وردز را هم ایندکس می‌کنند.

در انتها نگاهی می‌اندازیم به لیست ذیل که مواردی است که تا به حال یاد گرفتیم باید انجام دهیم برای عملیات تهیه دیکشنری و تهیه ترم از داخل اسناد که لیست کاملی است. ما از این لیست همه موارد به جز استیمینگ Stemming را یاد گرفته‌ایم.

- Stop words
- Normalization
- Tokenization
- Lowercasing
- Stemming
- Non-latin alphabets
- Umlauts
- Compounds
- Numbers

استیمینگ Stemming یا ریشه‌یابی

شاید مهم‌ترین بحثی که در این فصل داریم همین ریشه‌یابی باشد. پیشنهاد این است که برای تهیه دیکشنری، به جای هر فرمی‌از کلمه، ریشه کلمه را به دیکشنری در اینورتد ایندکس اضافه کنیم. مثلا به جای books کلمه book یا به جای taking کلمه take یا به جای کلمه went کلمه go را اضافه کنیم. این که برویم و ریشه کلمه را در بیاوریم بحث مهمی‌است.

- چطور به ازای هر کلمه ریشه پیدا کنیم؟
- آیا باید یک دیکشنری آماده داشته باشیم؟
- راهی نیست که یک الگوریتمی‌داشته باشیم که به طور اتوماتیک کلمه را بگیرد و ریشه‌اش را پس دهد؟
- اگر چنین الگوریتمی‌داشته باشیم مزیت‌ها و معایب آن چیست؟
- اگر یک دیکشنری آماده داشته باشیم مزیت‌ها و معایب‌اش چیست؟
- به طور کلی در IR، استیمینگ چه خاصیتی خواهد داشت؟

اینها همه سوالاتی است که برای ما مطرح است و ما در جلسه بعد به آن خواهیم پرداخت.

[مرور و مقدمه ابتدای جلسه]

لیستی که در صفحه قبل می‌بینیم برای تکه تکه کردن و توکنایز یک سند متنی به اجزائش است که ما عموماً آن اجزاء را ترم می‌نامیم. این لیست فهرستی است برای اینکه چطور ترم‌ها را داخل سند تشخیص دهیم و چطور یک سند را به یک مجموعه از ترم‌ها و اجزا تبدیل کنیم جهت اینکه بتوانیم بر مبنای آن محاسبات متنی‌مان را انجام دهیم. علاقمندیم که وقتی یک سند را به اجزاء تجزیه می‌کنیم، این اجزا معنی داشته باشند. مثلا سند را با «فاصله» تجزیه می‌کنیم و بعد اسم و کلمه و فعل و امثال آن را جدا می‌کنیم به صورتی که هر توکن را می‌توان تحلیل کرد و می‌توانیم متوجه شویم که چیست.

البته جای سوال است که آیا ما همیشه سند را به اجزایی تجزیه می‌کنیم که این اجزا با مسما و با معنی باشند یا خیر؟ دقت کنیم که حتی حروف اضافه هم معنی دارند. البته اگر استاپ وردها را حذف کنیم به احتمال زیاد حروف اضافه هم حذف می‌شوند. دانشجویان پیشنهاد می‌دهند که مثلا اگر اجزاء از نوع POSIX باشند در زمینه علوم کامپیوتر معنی دارند اما در زبان عادی بی‌معنی هستند. اما باید دقت داشت که منظور ما این نیست. بالاخره هر کلمه یا عبارتی از مثلا یک متن کامپیوتری را با فاصله توکنیاز کرده باشیم با توجه به اینکه ما علم کامپیوتر داریم، آن عبارات هم با معنی حساب می‌شوند و توسط ما درک می‌شوند و منظور سوال ما این نیست.

منظور ما این است که با یک روش خاص به نحوی توکنایز کنیم که ماحصل کار -ببخشید- چرندیاتی بیش نیستند. آیا چنین چیزی به درد ما می‌خورد؟

فرض کنیم سیستم توکنایزر ما به سندی شامل تنها یک عبارت POSIX بر بخورد و خروجی‌اش چنین توکن‌هایی باشد:

--P, -PO, POS, OSI, SIX, IX-, X--

این هفت توکن که خروجی توکنیازر ما از سند هستند و توکن‌هایی کاملا بی‌معنی هستند، به درد ما می‌خورند؟ در آینده خواهیم دید که این تکنیک خیلی کاربردها دارد و خیلی دردها را دوا می‌کند و مثل نُقل مشکل گشا است. به خصوص در زبان فارسی که ما به سادگی نمی‌توانیم توکنایز کنیم و با حروف چسبیده و جدا و انواع فرم‌های به هم ریخته و ... سروکار داریم که کار را سخت می‌کنند، این تکنیک کارگشا و مشکل‌گشاست.

این تکنیک که n-gram نام دارد، خیلی کابرد دارد و خیلی جاها استفاده می‌شود و کارهای خیلی سنگینی را با این تکنیک به سادگی می‌توان انجام داد. از دانشجویان می‌خواهیم که خودشان به عنوان یک تمرین بروند تحقیق کنند که n-gram چیست و چه کاربردهایی دارد و در محاسبات ما به چه شکل استفاده می‌شود. n-gram فقط روی متن یا بازیابی اطلاعات نیست که کاربرد دارد و خیلی جاها استفاده می‌شود. اما عمده بحث آن روی تکست و متن است و پیدا کردن شباهت اسناد و کلاستر و دسته بندی کردن اسناد و لنگوئج مادلینگ و ... و به طور کلی خیلی بحث ساده‌ای هم هست.

n-gram خیلی راحت تکست را می‌گیرد و تبدیل می‌کند به یک «مجموعه» و این هنر آن است. به محض اینکه متن و سند ما تبدیل به مجموعه می‌شود، از این لحظه است که همه بحث‌های ریاضی وارد می‌شوند و خیلی راحت می‌توانیم عملیات ریاضی مثل اشتراک و اجتماع و اندازه گیری تعداد عناصر مجموعه و ... که محاسبات ریاضی قابل فهمی‌هستند را داشته باشیم و به اهدافمان برسیم.

اما در این درس فعلا منظور ما از توکنایزیشن، جدا کردن کلمات با معنی است. معمولا برای توکنایز کردن با معنی، «فاصله» را به عنوان توکنایزر در نظر می‌گیریم. یعنی با وقوع یک فاصله می‌گوییم توکن تمام شده است و توکن بعدی شروع خواهد شد. البته یادمان هست که کلمات و توکن‌هایی دو یا چند بخشی داشتیم که در واقع باید یک کلمه یا توکن سرهم در نظر

گرفته می‌شدند. مثلا Los Angeles یا Iran University of Science and Technology که سر هم هستند و کنار هم ارزش خیلی بیشتری برای ما دارند و توکنایزیشن با فاصله در این موارد کار ما را سخت می‌کند. اما در ادامه خواهیم دید که با n-gram به چه شکل و به چه راحتی و به صورت جالبی توکن‌های دو قسمتی یا آنها که با فاصله یا با خط فاصله از هم جدا شده‌اند را می‌توان مشکلشان را حل کرد.

[پایان مرور و مقدمه ابتدای جلسه]

خلاصه این که در آن لیست نرمالایزیشن را داشتیم که یک سری پاکسازی است روی توکن. مثلا حروف اول را اگر بزرگ هستند کوچک بکنیم. البته اشاره کردیم که نرمالایزیشن شاید به نوعی بحث کلی تری روی همه موارد لیست است. مثلا در لیست لورکیسینگ را به طور جداگانه آورده است. یا استیمینگ که پیدا کردن ریشه کلمه است. مثلا با حذف s آخر یا es آخر یا ed یا ing ریشه‌یابی کنیم. البته می‌دانیم که با حذف هر ing هم نمی‌توانیم ریشه‌یابی کنیم. مثلاً در کلمه sing اگر ing را حذف بکنیم از این بیچاره چیزی باقی نمی‌ماند. پس برای انجام استیمینگ به یک سری تکنیک‌ها نیاز داریم.

اگر بخواهیم یک دیکشنری داشته باشیم که هر کلمه‌ای که به آن می‌دهیم، ریشه آن را به ما بدهد، مثلا sing را به آن بدهیم و به ما همان sing را به عنوان ریشه بدهد، چنین دیکشنری‌ای خیلی بزرگ می‌شود. برای این کار باید هر مشتقی از هر ریشه اضافه شود مثلا sing را یک ed به آخرش اضافه کنیم و در دیکشنری اضافه کنیم. دوباره ing اضافه کنیم و به دیکشنری بیافزاییم و ... و هر مشتقی از sing باید در این دیکشنری اضافه شود. پیدا کردن این مشتق در دیکشنری سخت می‌شود و دائما هم باید به روز شود و دائما کلمات جدید می‌آید و ... و خلاصه این روش مطلوب نیست.

شخصی به نام پورتر Porter که یک زبان دان انگلیسی است الگوریتمی‌را ایجاد کرد متشکل از چهل پنجاه قاعده و رول برای پیدا کردن ریشه کلمه ورودی و استیمینگ. مثلا می‌گوید اگر ing انتها را حذف کردیم و کمتر از دو کاراکتر باقی ماند، این کار را نباید انجام دهیم و ing جزو ریشه است. این الگوریتم به هر زبان برنامه‌نویسی روی اینترنت موجود است و شاید در حدود ۸۰ الی ۱۰۰ خط کد باشد. اینجا یک نمونه آنلاین را مشاهده می‌کنید.

به جای آن دیکشنری بزرگ و ناکارامد که پیشتر ذکر شد می‌توانیم از این قطعه کد کوچک پورتر استفاده کنیم و ریشه کلمات را پیدا کنیم و در دیکشنری قرار دهیم. پورتر نیازی به کاستمایزیشن و در نظر گرفتن کلمات جدید ندارد (بر خلاف رهیافت دیکشنری کامل). اما باید در نظر داشت که الگوریتم پورتر هم همیشه نمی‌تواند جوابگو باشد و طبیعتا ایراداتی هم دارد و نمی‌توان از آن انتظار عملکرد مطابق یک دیکشنری بزرگ را داشت. یعنی ممکن است ناگهان در موارد خاصی سر و ته کلمه را بزند و موجود عجیبی را تحویل دهد. مثلاً کلمه Automation را به autom تبدیل می‌کند.

نکته: لوسین Lucene یک موتور جستجوی اوپن سورس تحت مالکیت آپاچی است که به طور کامل در زبان جاوا نوشته شده است. مفاهیم این درس از جمله استیمینگ و لاورکیسینگ و ... در لوسین به خوبی استفاده شده است. اگر دانشجویان بخواهند برای جایی یک موتور جستجو بنویسند بهتر است از این موتور جستجوی اوپن سورس استفاده کنند که خیلی خوب کار را راه می‌اندازد.

نکته مهم: از نظر استاد به ترتیب با انجام توکنایزیشن، لاور کیسینگ، حذف استاپ وردز و در نهایت استیمینگ مهمترین کارها انجام شده و بقیه زیاد مهم نیستند. توکنایزیشن را با فاصله و کاما و ویرگول و نقطه و ... انجام می‌دهیم و بعد لاور کیس می‌کنیم، چون برای پیدا کردن استاپ وردها باید به فرم لاورکیس جستجو را انجام دهیم و در نهایت با الگوریتم پورتر استیمینگ می‌کنیم و هرچه باقی ماند و از این تونل رد شد می‌توانیم در دیکشنری اضافه کنیم.

نکته: استیمینگ، لنگوئج دیپندنت Language dependent و وابسته به زبان است. الگوریتم پورتر برای زبان انگلیسی است و برای زبان فارسی ما باید یک الگوریتم دیگر بسازیم که البته کارهایی در این زمینه انجام شده و احتمالا در گیت‌هاب موجود است. ایجاد الگوریتم استیمینگ برای زبان فارسی سخت‌تر است چون زبان فارسی بی‌نظم‌تر است و قاعده‌مندی آن کمتر است. هنر ما این است که قاعده‌های بیشتر و قواعد ترکیبی استفاده کنیم برای ساخت چنین الگوریتمیو در نهایت اگر در یک دسته از کلمات قواعد جوابگو نبود از یک دیکشنری استفاده کنیم. در هر صورت این الگوریتم در زبان فارسی یک نیاز جدی است. مثلا در موتور جستجوی ملی یا خیلی جاهای دیگر به یک استیمر خیلی خوب فارسی نیازمندیم.

البته «خیلی خوب» که یک حالت ایده‌آل است. مثلا همین الگوریتم پورتر یا الگوریتم‌های مشابه هم خودشان خیلی اشکال و گیر دارند. به شکل زیر دقت کنید:

Three stemmers: A comparison

Sample text: Such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation

Porter stemmer: such an analysi can reveal featur that ar not easili visibl from the variat in the individu gene and can lead to a pictur of express that is more biolog transpar and access to interpret

Lovins stemmer: such an analys can reve featur that ar not eas vis from th vari in th individu gen and can lead to a pictur of expres that is mor biolog transpar and acces to interpres

Paice stemmer: such an analys can rev feat that are not easy vis from the vary in the individ gen and can lead to a pict of express that is mor biolog transp and access to interpret

در این شکل یک متن نمونه و خروجی الگوریتم پورتر و دو الگوریتم استیمر دیگر آورده شده. همانطور که مشاهده می‌شود خیلی از کلمات پس از استیمینگ باز هم دچار مشکل هستند و در عمل این کلمات مشکل‌دار دارند در دیکشنری اضافه می‌شوند. مثلا Picture که خودش ریشه است و مشکلی ندارد به عبارت بی معنی Pictur تبدیل شده.

سوال مهم این است که اصلا چرا باید استیمینگ و ریشه‌یابی کنیم و مزیت این کار چیست؟ بعضی از دانشجویان پیشنهاد می‌کنند که مزیت این کار کاهش حجم دیکشنری و همینطور افزایش سرعت است. اما این، مساله اساسی ما نیست. به خصوص این روزها با داشتن ابزاری مانند هاردهای SSD دیگر مشکل حجم و سرعت تا حدود زیادی به صورت سخت‌افزاری مرتفع شده است. شاید بهتر باشد سوال را با در نظر گرفتن این پیش‌فرض که CPU و هارد بسیار پرقدرت و پرسرعت در اختیار داریم بررسی کنیم تا به نکته اصلی برسیم. پاسخ این است که به کمک این کار می‌توانیم اسنادی را با کوئری مچ بکنیم که ریشه کلمات کوئری را هم داشته باشند و جوابهای بیشتری از اسناد مشابه به کاربر ارائه دهیم.

فرض کنید books در سندی وجود دارد. اگر کاربر کوئری بزند که بخواهد اسناد شامل book را دریافت کند، آیا سند فرضی را باید به وی برگردانیم یا خیر؟ معلوم و بدیهی است که دوست داریم این کار را بکنیم و این سند را هم به کاربر برگردانیم. در

حالی که می‌دانیم books عینا مطابق کوئری کاربر نیست. اما با انجام عمل استیمینگ، این مشکل برطرف می‌شود و این سند هم به کاربر برگردانده می‌شود.

با انجام عمل استیمینگ، تعداد اسنادی که در جواب کوئری کاربر برگردانده می‌شوند بیشتر خواهند شد و اصطلاحاً ریکال Recall یا صدا زدن بالا می‌رود اما پرسیژن Precision یا دقت کم می‌شود. یعنی ممکن است اسنادی بی‌ربط به نیاز اطلاعاتی کاربر هم به عنوان جواب برگردانده شود. در واقع مزیت، افزایش ریکال است که دیگر مطمئن می‌شویم که هر حالت ممکن از کلمه را بر می‌گردانیم. اما عیب این روش، کاهش پریسیژن است.

سوال: آیا استیمینگ، افکتیونس effectiveness یا کارایی را بهبود می‌دهد؟ پاسخ این است که استیمینگ برای بعضی کوئری‌ها خوب است و برای بعضی دیگر خوب نیست. مثال خوب: کوئری books که به book استیم می‌شود و پاسخ‌های مناسبی را می‌دهد. مثال بد: کوئری Automate که به Autom استیم می‌شود و نتایج بی‌ربط را برمی‌گرداند. یا مثلا مثالی که در کتاب وجود دارد Oper است که بر اساس الگوریتم پورتر ریشه کلمات بسیار متنوع با معانی ناهمگون مانند operate, operating, operates, operation, operative, operatives, operational و ... است. با چنین مجموعه‌ای از کلمات که ریشه یکسان دارند چنین کوئری‌هایی از استیمینگ سود نخواهند برد و بلکه نتایج بی‌ربطی را هم ارائه خواهند داد:

“operational AND research”, “operating AND system”, “operative AND dentistry”

مشاهده می‌شود که مفاهیمی‌بسیار متنوع شامل تحقیق در عملیات که یک درس بسیار مهم در صنایع است، سیستم عامل و جراحی دندانپزشکی ممکن است در این کوئری‌ها با استیمینگ، اسناد بی‌ربطی را به عنوان پاسخ دریافت کنند و کیفیت کار پایین می‌آید.

اگر بخواهیم جمع‌بندی داشته باشیم بهتر است بگوییم که در کل استفاده از استیمینگ پیشنهاد می‌شود.

نکته: قبلا دیدیم که منظور از استاپ وردز کلماتی هستند که معنی خاصی نمی‌دهند و کاربر در کوئری به دنبال این کلمات نیست. مانند: at, a, an, of, is, not و این کلمات کمک خاصی به جستجوی سند نمی‌کنند. یعنی «ما» اینطور در نظر می‌گیریم که کاربر کوئری‌ای نخواهد داد که در آن بگوید دنبال اسنادی هستم که مثلا of در آنها باشد و کاربر همیشه دنبال کلمات با معنی است. البته این مورد خود جای بحث زیاد دارد و در درس فصل ۶ به آن بر می‌خوریم.در آنجا بررسی می‌کنیم که یک کلمه چقدر بار معنی یک سند را به دوش می‌کشد. مثلا اگر به ما بگویند در یک سند کلمه a وجود دارد، ما متوجه نخواهیم شد که آن سند در مورد چیست. اما اگر بگوییم در یک سند کلمه Naderi وجود دارد، بلافاصله فضای جستجوی ذهنی ما به مفاهیم خاصی محدود می‌شود و مثلا می‌گوییم احتمال دارد این سند در مورد دکتر نادری از دانشگاه علم و صنعت باشد. کلمه‌ای خاص‌تر مانند Ali Daei را درنظر بگیریم، در این حالت می‌گوییم این سند به احتمال زیاد در مورد فوتبال است. مثلا اگر بگوییم در سند کلمه «توپ» وجود دارد، ذهنمان به دو زمینه مفهومی‌محدود می‌شود و می‌گوییم این سند یا در مورد توپ جنگی است یا در مورد توپ ورزشی. پس در اینجا ما با دو گروه کلمات سر و کار داریم. کلماتی که بخشی از بار معنی سند را بر دوش دارند و کلماتی که بخشی از بار معنی سند را بر دوش ندارند که این دسته دوم را استاپ وردز در نظر می‌گیریم.

نکته: مورد دیگری که خیلی در دنیای موتورهای جستجو در زبان انگلیسی به زیبایی روی آن کار شده است و خیلی استفاده می‌شود و چندین هزار مقاله حول و حوش آن نوشته شده است مفهومی‌است به نام Wordnet که در واقع گرافی است از کلمات و ارتباطات بین آنها. این مفهوم خیلی به درد جستجو و بازیابی و تکست پراسسینگ می‌خورد. البته به زبان انگلیسی

است. در زبان فارسی هم تلاش‌هایی صورت گرفته که این گراف عینا ترجمه شود ولی طبیعتا با توجه به تفاوت‌های ساختاری زبان‌ها، این گرافِ ترجمه شده به کیفی وردنت انگلیسی عمل نمی‌کند و نقص‌هایی دارد.

اگر در این زمینه‌ها بخواهیم کارهای علمی‌انجام دهیم همین زمینه وردنت فارسی خیلی کمک می‌کند و زمینه جالبی است و به خصوص اگر با یک زبان دان در ارتباط باشیم استیمر اتوماتیک فارسی مطالب و وردنت فارسی مباحث بسیار جذابی هستند.

نکته: الگوریتم‌های استیمر، الگوریتم‌هایی چهل پنجاه خطی هستند که بر اساس یک سری رول و قانون کار می‌کنند و نمی‌توانیم از آنها انتظار داشته باشیم که همه کلمات را درست استیم کنند. بعضی جاها بالاخره مشکلاتی دارند. مثلا ممکن است دو کلمه که معنی یکسانی ندارند را یک ریشه یکسان از آنها به ما بدهد و این جزو خطاهای این الگوریتم‌ها است.

نکته: الگوریتم‌های استیمر مختلفی داریم، اما معروف‌ترین آنها همین الگوریتم پورتر است که بر اساس یک سری قرار داد کار می‌کند. ۵ فاز کاهش یا ریداکشن Reduction دارد که به صورت ترتیبی و یکی یکی روی کلمه اعمال می‌شوند و انگار یک کلمه دارد از یک پایپ لاین PipeLine رد می‌شود و این پردازش‌ها مطابق قواعد روی آن انجام می‌شود تا به ریشه‌اش تبدیل شود که دیگر وارد بحث‌های جزئی آن نمی‌شویم. الگوریتم‌های دیگری تحت عنوان لوینز Lovins یا پِیس Paice و خیلی الگوریتم‌های دیگر هم وجود دارد که استیمینگ انجام می‌دهند. در شکل زیر نکات مهم الگوریتم پورتر مرور شده است.

Porter algorithm

- Most common algorithm for stemming English
- Results suggest that it is at least as good as other stemming options
- Conventions + 5 phases of reductions
- Phases are applied sequentially
- Each phase consists of a set of commands.
  - Sample command: Delete final *ement* if what remains is longer than 1 character
  - replacement → replac
  - cement → cement
- Sample convention: Of the rules in a compound command, select the one that applies to the longest suffix.

نکته: گوگل استیمینگ انجام نمی‌دهد و خود کلمات را کامل در نظر می‌گیرد. گوگل حتی استاپ وردز را هم حذف نمی‌کند. مثلا اگر a را در گوگل جستجو کنیم با تقریب خوبی می‌توانیم تعداد کل اسناد لاتین در گوگل را بفهمیم.

نکته: در مورد این که چه تکنیک‌ها و الگوریتم‌ها و روش‌هایی در موتورهای جستجو استفاده شود، توافقی میان آنها (موتورهای جستجو) وجود ندارد. موتورهای جستجو در مسابقه هستند و بیشترین زحمت را می‌کشند و تا جایی که بتوانند کار می‌کنند تا بتوانند بهترین پاسخ را به کوئری‌ها بدهند. البته به طور دقیق معلوم نیست داخل الگوریتم‌هایشان چه اتفاقی

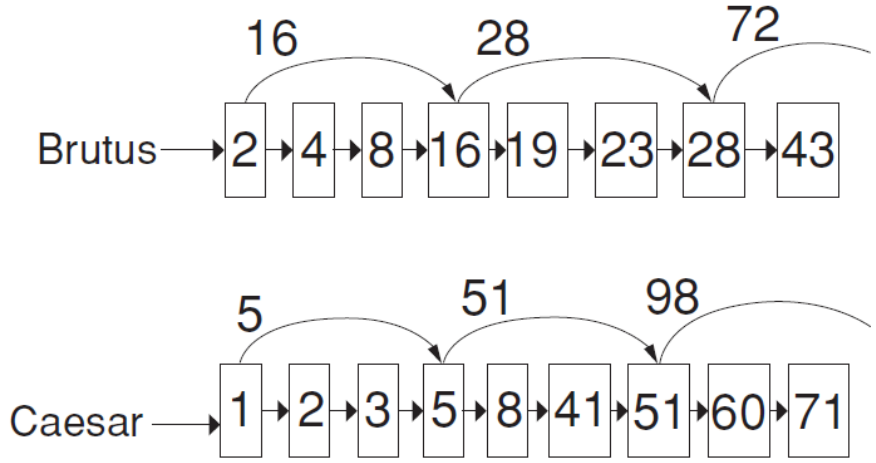
می‌افتد و مثلا ما خبر نداریم که گوگل چطور استیم می‌کند. به نظر می‌رسد با توجه به منابع پردازشی خیلی زیادی که دارد به راحتی در یک دیکشنری خیلی بزرگ سرچ می‌کند و کلمه را پیدا می‌کند و این احتمال هست که اصلا از الگوریتم پورتر هیچگاه استفاده نمی‌کند. الگوریتم پورتر بیشتر در موتورهای جستجوی کوچکی که ما می‌خواهیم پیاده سازی کنیم به درد می‌خورد.

اسکیپ پوینترز Skip Pointers

این بخش یعنی اسکیپ پوینترز زیاد مهم نیست اما آن را یک بررسی اجمالی می‌کنیم. به مثال اصلی این درس برگردیم. فرض کنید دو کلمه Brutus و Calpurnia را به ما داده‌اند و گفته‌اند تمام اسنادی را که شامل این دو کلمه هستند را بیابید. برای این کار گفتیم که باید بیاییم اشتراک پستینگز لیست دو کلمه را پیدا کنیم و این اشتراک پیدا کردن دارای پیچیدگی از مرتبه O(m+n) است. این روش به طور تئوریک با مرتبه پیچیدگی کمتر قابل اجرا نیست.

اما یک تکنیک ابداع شده به نام اسکیپ پوینترز به این صورت که می‌گوییم طول پستینگز لیست‌ها عموما خیلی زیاد است. مثلا در نظر بگیریم پستینگز لیست Brutus را داریم به طول 10^6 عضو که این خیلی زیاد است و پیمایش یکی یکی این اعضا زمان زیادی می‌برد. طول پستینگز لیست Calpurnia را هم 10^2 در نظر می‌گیریم.

در این تکنیک می‌گوییم حداکثر تعداد اعضا حاصل از اشتراک این دو پستینگز لیست برابر طول پستینگز لیست کوچکتر و معادل ۱۰۰ است. یعنی ما باید در پستینگز بزرگتر، حدود یک میلیون عضو را پیمایش کنیم برای یافتن حداکثر ۱۰۰ عضو. ایده ما این است که به جای پیمایش یکی یکی اعضای پستینگز لیست‌ها، چندتا چندتا برویم و جلو مقایسه را انجام دهیم. عموما عدد پیشنهادی برای این «چند» برابر است با جذر طول پستینگز لیست که در اینجا برای پستینگز بزرگتر ۱۰۰۰ و پستینگز کوچکتر ۱۰ می‌شود. یعنی ما در پستینگز لیست بزرگتر بهتر است هزارتا هزارتا اشاره‌گرمان را ببریم جلو و در پستینگز کوچکتر ده تا ده تا. دقت کنید که اگر در یک پرش، زیاد جلو پریدیم، باید به عقب برگردیم. در هر پرش به عقب، اندازه پرش نصف گام قبلی در نظر گرفته می‌شود. مثلا اگر گام پرش ۱۰۰ عددی رو به جلو داشتیم و در مقایسه متوجه شدیم که ممکن است یک عضو اشتراک را در این پرش رد کرده باشیم، گام پرش به عقب ما ۵۰ عددی خواهد بود و اگر هنوز هم در مقایسه متوجه بشویم که ممکن است در ۵۰ تای قبلی عضو اشتراکی داریم این بار گام رو به عقب ۲۵ عددی خواهد بود و ... . به اسلاید زیر به عنوان مثال نگاه کنید:



نکته بسیار مهم این که پیچیدگی هیچگاه بهتر نمی‌شود اما به نظر می‌رسد تعداد مقایسه‌ها در این ایده کمتر می‌شود. البته این تکنیک برای مواقعی که طول یکی از پستینگز لیست‌ها و در نتیجه مجموعه اشتراک کم است جوابگو است. اگر تعداد اشتراکات دو لیست زیاد باشد حتی این تکنیک ممکن است عملیات ما را پیچیده‌تر کند و این روش کار ما را بدتر می‌کند و اینطور نیست که بگوییم این روش همیشه خوب جواب می‌دهد.

در واقع ما «احتمال» می‌دهیم که مجموعه اشتراک دو لیست تعداد اعضای زیادی ندارد و به همین دلیل چندتا چندتا اشاره‌گرها را جلو می‌بریم. این احتمال در عالم واقع احتمال خوبی هم هست. مثلا خیلی کم اتفاق می‌افتد که هر سندی «علی» داشت «دایی» هم داشته باشد. تعداد اسناد دارای «علی» قطعا زیادند و به همین دلیل و با این تکنیک می‌صرفد که پستینگز لیست «علی» را تند تند و چندتا چندتا برویم جلو.

نکته: در ایام قدیم طبق تجربه این روش بسیار به اجرای عملیات بازیابی کمک می‌کرد و سرعت را افزایش می‌داد. اما در دوران کنونی و با توجه به CPUهای سریع امروزی این روش آنقدرها تاثیری ندارد.

فرِیز کوئری Phrase Queries

اگر یک کوئری داشته باشیم که بخواهیم در آن دو یا چند کلمه کنار هم قرار گرفته باشند، مانند Stanford University را یک فریز کوئری می‌نامیم. یعنی یک عبارت یا فریز داریم مانند Stanford University و می‌خواهیم سندهایی را پیدا کنیم که در آنها این دو کلمه حتماً به همین ترتیب کنار هم باشند. به عبارتی دنبال سندهایی هستیم که این فریز در آنها باشد. حدود ۱۰ درصد از کوئری های وب، فریز کوئری هستند.

سه چهار روش برای پاسخگویی به این نوع کوئری‌ها وجود دارد. دو تا از راه حل‌ها و روش‌ها را با هم بررسی می‌کنیم.

بایورد ایندکس Biword Indexes

در این روش ایندکسمان را به نحوی توسعه می‌دهیم که کلمات پشت‌سرهم هم داخل ایندکسمان باشند. یعنی علاوه بر اینکه کلمات جدا را در ایندکس قرار می‌دهیم، عبارات دو کلمه‌ای را هم در ایندکس قرار می‌دهیم. یعنی با عبارات دو کلمه‌ای هم مانند یک توکن یا ترم برخورد می‌کنیم. مثلا از 2-gram استفاده می‌کنیم و سند را توکنیاز می‌کنیم. به مثال زیر دقت کنید. فرض کنیم سند ما شامل یک جمله به شکل زیر است.

Index every consecutive pair of terms in the text as a phrase.

در هنگام توکنیاز، اول تک تک کلمات را توکنایز می‌کنیم و به دیکشنری اضافه می‌کنیم و بعد از 2-gram استفاده می‌کنیم و عباراتی مانند این عبارات که در واقع همان Biword های ما هستند را استخراج می‌کنیم:

Index every, every consecutive, consecutive pair, pair of, of terms,…

و این‌ها را هم مانند توکن‌های عادی بررسی و نرمالایز و ... می‌کنیم و به دیکشنری اضافه می‌کنیم.

نکته: اگر مثلا از 3-gram استفاده می‌کردیم عبارات استخراجی به این صورت بودند:

Index every consecutive, every consecutive pair, consecutive pair of, pair of terms,…

این روش به همین سادگی است. البته این روش باعث افزایش حجم دیکشنری از مرتبه حداکثر توان ۲ می‌شود. یعنی اگر کل ترم‌ها T عدد باشند، حجم دیکشنری جدید حداکثر T*T یا T^2 خواهد بود. البته در واقعیت همیشه کمتر است و تنها زمانی دقیقا برابر T^2 خواهد شد که در اسناد، هر کلمه‌ای در کنار هر کلمه دیگری بیاید که چنین چیزی تقریبا غیر ممکن است. با این روش، Biword ها را می‌توانیم پاسخگو باشیم. حالا اگر Triword Phrase یا عبارات سه کلمه‌ای داشته باشیم چه کنیم؟ در این حالت از 3-gram هم استفاده می‌کنیم و حجم دیکشنری هم از مرتبه T^3 خواهد بود که در این حجم دیگر یک مقدار کارمان سنگین است و زیاد صرفه ندارد. اگر Long Phrase داشته باشیم چه؟

در اینجا می‌توان از یک تکنیک جدید استفاده کرد. مثلا مثال stanford university palo alto را در نظر بگیریم که یک فریز ۴ کلمه‌ای است. در اینجا اگر 2-gram را به دیکشنری اضافه کرده باشیم می‌توانیم به جای ذخیره کردن 4-gram عبارت فوق، از کوئری بولی زیر استفاده کنیم:

"stanford university" AND "university palo" AND "palo alto"

در واقع ما اینجا چون پیچیدگی 2-gram یا bi-gram یا biword را تحمل کرده‌ایم و هزینه آن را پرداخته‌ایم، می‌توانیم از آن استفاده بهینه کنیم.

تکنیک به صورت خلاصه اینطور معرفی می‌شود که خود کوئری لانگ فریز را هم به وسیله 2-gram توکنایز می‌کنیم و بعد بین این توکن‌ها AND می‌گذاریم و کوئری بولی را اجرا می‌کنیم. در این صورت یک فیلترینگ خیلی خوب صورت گرفته و شاید از بین چند میلیون سند، یکی دو سند برگردانده شوند که حائز این شرایط باشند. ما این اسناد را بر می‌گردانیم به شرط آنکه یک پست پراسس هم روی آنها انجام دهیم.

این نکته مهمی‌است که در واقع ما به صورت منطقی نمی‌توانیم ادعا کنیم که کوئری بولی جدید، به طور دقیق کوئری لانگ فریز ما را ستیسفای و برآورده می‌کند. ممکن است سندی وجود داشته باشد که تک تک توکن‌های bi-gram را داشته باشد اما این توکن‌ها پشت سر هم نباشند و در جاهای مختلف سند قرار گرفته باشند. در این صورت کوئری بولی، این سند را یک جواب می‌داند. در حالیکه این سند جواب کوئری لانگ فریز ما نیست و به همین علت باید یک پست پراسس هم انجام دهیم. در این حالت سندهای جدا شده را پیمایش می‌کنیم و لانگ فریز را در آنها جستجو می‌کنیم که این دیگر برای سرورها زیاد کار سختی نیست. چون ما قبلا به کمک کوئری بولی یک فیلترینگ بزرگی انجام داده‌ایم و تعداد زیادی از اسناد را حذف کرده‌ایم و فقط تعداد سندهای کمی‌باقی مانده‌اند و در این حالت می‌گوییم که اشکال ندارد و می‌صرفد که برویم از ابتدا تا انتهای سند، کوئری لانگ فریز را بررسی کنیم. کتاب درسی در این قسمت به جای اصطلاح پست پراسس از Post-Filtering استفاده می‌کند.

در انتهای این بخش باید گفت که در مجموع این روشِ بایورد ایندکس به ندرت استفاده می‌شود و زیاد روش خوبی نیست. ایرادات آن را باید مرور کنیم. دانشجویان پیشنهاد می‌کنند که یکی از ایرادات آن حجم دیکشنری است. اما ایراد این روش این مورد نیست، چون در حال حاضر با استفاده از هاردهای پرقدرت و پرظرفیت به این مشکل بر نمی‌خوریم. اما در اسلاید درسی هم به این مساله اشاره شده به عنوان یکی از ایرادات مساله که ما امروزه زیاد آن را جدی نمی‌گیریم.

از ایرادات بایورد ایندکس می‌توان به مساله فالس پازتیو False Positive اشاره کرد. برای توضیح آن بر می‌گردیم به مثال آخر و فریز ۴ کلمه‌ای. در تکنیکی که معرفی کردیم، اسنادی به عنوان جواب برگردانده می‌شدند که باید پست فیلتر یا پست پراسس می‌شدند. اگر پست فیلترینگ نداشته باشیم، همه سندهایی که دارای لانگ فریز مد نظر هستند را مطمئنیم در میان جواب‌ها داریم. اما علاوه بر آن متاسفانه اسناد دیگری را هم در میان جواب‌ها داریم که در حقیقت جواب فریز کوئری ما

نیستند که به اینها جواب‌های فالس پازتیو می‌گوییم و باید دقت کنیم که به این اسناد عنوان «جواب نادرست» نسبت ندهیم. دلیل این که نباید از عنوان «نادرست» استفاده کنیم این است که همانطور که در زمینه‌های دیگر علم کامپیوتر مانند داده کاوی هم دیده‌ایم می‌توانیم ۴ حالت برای جواب‌ها در نظر بگیریم شامل ترو پازتیو True Positive و فالس پازتیو False Positive و ترو نگتیو True Negative و فالس نگتیو False Negative که یک جدول دو در دو را تشکیل می‌دهند.

می‌گوییم که مشکل فالس پازتیو معمولا با یک فیلتر اضافه یا پست فیلتر قابل حل است. اما اگر در یک عملیات به شرایط فالس نگتیو بر بخوریم خیلی بد است و رفع این مشکل کار سختی است. در اینجا مثلا فالس نگتیو را می‌توان این طور مثال زد که در انتهای عملیات، سندی که باید به عنوان جواب بر می‌گرداندیم را بر نگرداندیم و پیدا کردن این سند کار مشکلی است و انگار اصلا عملیاتی انجام نداده‌ایم و پیدا کردنش سخت یا غیر ممکن است.

اما در حالت فالس پازتیو، یک فیلترینگ خوبی صورت گرفته، اما چند سند اضافه‌تر از مجموعه جواب (علاوه بر مجموعه جواب) برگردانده شده است. در اینجا دیگر پست فیلتر کردن کار ساده‌ای است چون مجموعه اسناد بدست آمده، خیلی کمتر از کل اسناد هستند و به سادگی با یک پست فیلتر می‌توانیم مشکل را حل کنیم. در مثال ما، تا قبل از پست فیلتر مطمئن هستیم که سندهای دارای فریزِ ۴ کلمه‌ای حتما در مجموعه جواب هستند. اما چند مورد سند غیر جواب هم ممکن است لابلای اسناد ما باشند. اگر به نوعی روش ما فالس نگتیو بود به این مفهوم بود که قبل از پست فیلتر، هنوز همه اسنادی که جواب ما هستند، از دل مجموعه کل اسناد بیرون کشیده نشده‌اند.

در همین زمینه روش دیگری به نام پوزیشنال ایندکس را در جلسه آینده بررسی می‌کنیم.

جلسه ۴

دوستان عزیز لطفا یه ربع اخر فیلم رو ببینید ب دلیل خرابیlmsانتونستم فرمول رو بنویسم ومثالش موفق باشید

جلسه و جلسات پیش در مورد بازیابی اطلاعات بولین صحبت کردیم. جایی که کوئری های ما از not - or - and ساخته می شوند به صورت منطقی و ما از شاخص معکوس استفاده می کنیم که این شاخص معکوس ارتباطات کلمات را با اسنادی که داخلشون هست برقرار می کند و اینکه یک کلمه در چه اسنادی است .

حالا داستان از این قرار به ما یه مجموعه از اسناد مرتبط با کوئری به ما بر می گردونه ولی هیچ نظری در مورد اینکه کدام یک از این اسناد بهتر است به ما نمیده یک منطقی بوده رفته دیده اون منطق در سند برقرار است و True دارد و اگر برقرار نباشد False میده و کاری اینجا به درصد و کمتر و بهتر و ... نداره ما میخواهیم سیستمی داشته باشیم که علاوه بر اینکه سندی مرتبط رو به کاربر نشون میده بلکه کدومشون هم مرتبط تر هست یه وزنی به این اسناد بده براساس وزنی که به اسناد میده ما سندارو مرتبط می کنیم اون سندی که از همه وزنش بیشتر rank بالاتری تلبیده و زودتر به کاربر نشون داهد بشه مدلهای متفاوتی هستند که این کار برو برامون انجام میدن. اسناد بر مبنای رابطشون و مقدار ارتباطی که به کوئری وزن می دهند میگن سند چقدر به کوئری کاربر نزدیکه چقدر به اون موضوع مرتبط و بعد از این وزن ما استفاده می کنیم و rank و ترتیب سندها رو توی لیستی که به کاربر می خواهیم نشون بدیم مشخص می کنیم یکی از مهمترین مدلهای بازیابی اطلاعات که این کار را انجام میده و به سندهای ما وزن میده مدل برداری است، مدل فضای برداری است (Vector Space Model) مدل های زیاد دیگه ای مطرح شدن مثل مدل احتمالی ، مدل زبانی Language و روش های مختلف دیگه مثل شبکه عصبی و ....اما یکی از مهمترین روش بازیابی اطلاعات مدل برداری هست. چیکار میکنه مدل برداری؟ ما توی جلسه

پیش گفتیم سندهامون رو تبدیل می کنیم به مجموعه کلمات اینجا میگیم سندمون رو تبدیل می کنیم به یک بردار به یک vector فرض ما براین است که شما بردار رو توی ریاضیات خوندید ولی در حد مرور تکرار میشه (موضوع فصل ۶) برای اینکه بتونیم از مدل vector space استفاده کنیم تا به اسنادمون وزن بدم برای وزن دادن به اسناد باید قبلش یه کاری انجام بدم و اون اینکه قبل از وزن دادن به اسناد باید به کلماتم و Term ها وزن بدم.

سوال؟

به نظر شما چند فرمول وزن دهی به کلمات وجود دارد؟ به چند روش می تونم وزن بدم؟ بیشتر از یک، روش های مختلفی برای وزن دادن به کلمات هست . بعضی از روش های موقعیت و مکان کلمات رو در نظر میگیرن بعضی از روش ها IDF رو در نظر میگیرن بعضی از روش ها اینها رو به گونه های مختلف ترکیب میکنند. خب ما باید بریم سراغ اون روش ها به کلماتم وزن میدم تا بتونم سندها رو تبدیل به بردار بکنم.اول یه مروری داشته باشیم به فصل قبل و بعد داستان های بولین که میخواییم برسیم به vector space. بولین که گفتیم چندین ایراد داره یکی از ایراداتش که گفتیم setمیده و برای شما مرتب سازی نمیکنه. یه ایراد دیگه که ممکن داشته باشه هرکسی نمیتونه کوئری بولین بزنه .ما عموما وقتی وزن میدیم به اسناد به یک زوج سند کوئری وزن می دیم، یعنی این سند خاص ما چقدر مرتب با این کوئری است. ما منظورمون وزن تنها به سند نیست بلکه به زوج کوئری وداکیومنت وزن میدهیم که نشان دهنده مقدار ارتباط و مناسب بودن یک سند کوئری است عموما فرمول هایی که ما بدست می آوریم و استفاده می کنیم دوست داریم این عددها بین ۰ و ۱ باشن, نرمالایز شده باشند بین صفر و یک خوبیش اینه که میدونی max تون چیه و minتون چیه وسط چند و ... برای همین بیشتر فرمول های عدد رو بین صفر و یک میدن و نشون میدن که یک کوئری چقدر میتونه مناسب باشد برای یک سند براساس این وزن یک و صفر هست sort می کنیم داکیومنت ها رو تا بتونیم نشون بدیم حالا داستان اینه که وزن رو چطوری حساب بکنیم .اگر هیچ کلمه کوئری در سند نباشد باید وزن صفر باشد هرچی تعداد رخداد یک کلمه در یک سند بیشتر باشد اون سند مرتبط تر با اون کوئری

البته گفتیم توی مدل برداری سندو کوئری رو بتدیل به بردار می کنیم اما تو همون دنیای مجموعه ها که ؟؟؟ می کردیم می تونستیم به یک عدد برسیم مثلا اگر من کوئری رو A بگیرم و سندم رو Bبگیرم و هر دو مثل مجموعه در نظر بگیرم من میتونم یه فرمول اینطوری در نظر بگیرم که یک فرمول معروف همهست به اسم فرمول jacccard

|A /\ B| |A \/ B|

می تونم اشتراک تعداد کلمات که توی کوئری هست با سند حساب کنم و تقسیم بر اجتماع کنم که سایز شونم در نظر گرفته باشم این به من یه عدد میرسونه و من براساس این عدد می توانم مرتب سازی کنم سندهامو ولی نکته ای که هست چقدر این قرمول برای من دقیق؟

این جاکارد پس تو همون دنیای مجموعه ها کار میکنه ولی کیفتیش اون چیزی که مامیخوایم نیست. وقتی یک سند و کوئری کاملا مساوی باشن یک میشه وقتی اشتراکی هم نداشته باشین صفر میشه اون مزینی که ما گفتیم که باید بین صفر و یک باشن وجود داره پش این اشترک شدن تقسیم بر تعداد اجتماعشون میشه. خب ایرادی که داره فرمول جاکار ..... در نظرنمیگیره به گفته کامل تر وزن term ها رو در نظر نمگیره انگار نه انگار اون term که تواشتراک نیست یک term خیلی مهم انگار نه انگار اون term که تو اشتراک یک term بدردنخوره (وزن term ها رو در نظر نمیگیره) خب به همین خاطر جاکارد زیاد اینجا خوب جواب نمیده و میریم سراغ مدل برداری اولین کلماتی که به ذهن بچه ها رسیده بود و نام برده بودن برای اینکه من بتونم وزن بدم به term هام

Term frequencey -> تعداد رخداد کلمات توی اسناد هرچی یک term توی سند بیشتر رخداده باش اون term بیشتر با اون سند در ارتباط و اون سند بیشتر مرتبط با اون term این همون ماترسی که جلسه قبل ساختیم ماتریس word documentکه وقتی من میخوام term frequencyدرنظربگیریم یچیزی به این شکل میش یعنی دیگه اونجا شما ۰و۱ندارید بلکه ازدنیای اعدادطبیعی میتونید مقادیرتون روانتخاب بکنید اینجا یعنی مثلا

anthony,

157 بار درسند

anthony and clepatra

اومده .

خب دراینجاازفضای بولی خارج میشیم ومیریم توی فضای اعداد طبیعی

اینجا یه گریزی بزنم به اون فصل بردار من به یک سند نگاه بکنم عددهای ستونش وقتی بررسی بکنی میبینی این یک بردار هست ۱۸۵۲۲_۰

یک بردار که هملت رونشون میده

وشما جای این اعددهارو نباید عوض کنید جاشون روعوض کنی یعنی ۰کهanthonyبوده با۲که brutusبوده عوض میشه منظور از بردار همینه که جای اعداد روعوض نکنید تا بتونید باهم مقایسه کنید

مثلا۷۳رو با۰مقایسه کنیم یابقیه رو باهم مقایسه کنید این اولین قدم برای برداری کردن سندهاست پس خیلی ساده چیشد :

۱_برای من کافی تمامtermهای کل اسناد مو لیست کنم

وقتی تمام کلمات رو دارم نگا میکنم یکسری کلمات هستند که داخل بعضی ازسندها نیستند یعنی شما ممکن ۰داشته باشیم وقتی تمام کلمات رولیست کردی که پیشنهاد میشه به صورت حروف الفبابب مرتب بشه (یعنی ازaشروع بشه الی اخر)شماتعداد رخداد کلمان رودراسناد مینویسید

۰۲۲۰۰۸۱

پس این مهم چه نرتیبب ازکلمات باشند که برای همه یکدست ویک طور باش وقابل مقایسه ومحاسبه باشند

اینطوری اولین نسخه بردارابرامون درمیاد ولی اینطوری برامون بردار کیفیت بالایی نیست مابایدبردارهامون کیفیتش بالاترباش وبازیابی قوی تری داشته باشیم

خب من اینجا بازم چون کلماتم رو مرتب کردم به صورت الفبایی این موقعیت و ترتیب کلمات رودر سند لحاظ نمیکنم واینکه مثلا رضاازعلی سریعتر بااینکه ۱۸۰درجه باهم فرق دارند وباهم دعواشون میشه که شما کدوم جلمه رو بگید مدل بازیابی اونارو درنظرنمیگیره .

اولین پارامتر term frequency روباTF(t,d)نشون میدیم تعداد رخداد term__>

t

درسندdازاین پارامترها میتوانیم استفاده کنیم برای وزن دهی خیلی ساده میتوانیم اونایی کهTFبالاتری دارند وزن ببشتری بدیم حالا یه نکته ایی هست میگن اگر یک term__>

۱۰بار توی یک سند رخ بده ولی یه termدیگر یکبار رخ بدهددرسته اون termکه ۱۰بار رخ داده اهمیت بیشنری داره وزن بیشتری میگیره اما نه لزوما به اندازه ۱۰برار بنابراین باید راهی داشته باشیم که این درصدها(۱۰برابر.۱برابرو...)نباش

یکی از روشهای خیلی خوبی که میان این ضریب رو درست میکنن لگاریتم گرفتن اگر شما ازاون termکه رخداد درسندهاتونه لگاریتم بگیرید اگر۰باش که لگاریتم صفر تعریف نشدس +۱هم میکنیم که لگاریتم ۱صفر نشه یک کلمه ی که یکبار رخ داده وزنش صفر نشه

واگر لگاریتم رو بسط بدیم لگاریتم nرو میرسیم به یکسری عدد به عنوان۱+۱/۲+۱/۳.....۱/nکه احتمال زیاددیدید که لگاریتم رومیشه با یه سری نوشت یه دلیلی هست که ماز لگاریتم استفاده میکنیم. برای محاسبه وزن کلمه میگیم وقتی یک کلمه TF1داد به محض اینکه دریک ساعتی ظاهرمیشه خب یک وزنی میگیره .اینکه این سند درمورد این کلمه داره حرف میزنه ، مفاهیمشون به هم نزدیک درکل دفعه اول ۱ولی دفعه دوم لزوما ۱نیست یه وزن کمتری میگیره دومین رخداد کلمه درسند اینقدر اهمیت ندارد که اولیش اهمیت دارد مثلا دفعه اول ۱واحد اضافه میکنم دفعه دوم ۱/۲اضافه میکنم دفعه سوم۱/۳اضافه میکنم و...

این اعدادی که گفتیم همون لگاریتمnحالا که تونستیم به کلمات وزن لگاریتمی دهیم میتونیم به کل سند وزن دهیم نمره ی سندم میشه مجموع وزن هایی که کلمات میگیرن به شرط که

دراشتراک باشند پس کلماتی که در سندهستند و درکوئری مشترک هستند پیدامیکنم وزنشونو که لگاریتم حساب میکنم باهم یک سیگما ساده میگیرم جمع میکنم ومیشه نمره ی که به من سندتوی قدم بعدی یه پارامتردیگه به نامidfدرنظر میگیریم که کیفیت وزن دهی ماروبهترمیکنه بحث تعداد رخداد دریک ,collectionبه دو صورت میتونم رخداد کلمات در یک collectionرو حساب کنم:

۱-شمارش که چندبار رخ داده

(که زیاد بدرد نمیخوره)

۲_من کارنداشته باشم چندبار دریک سند رخ داده بیام ببینم درچندسند وجوددارد ===»سندها را بشمارم که در ۱۰تاسند یا۱۰۰تا سند یا....

عموما کلماتی که عاما عمومی هستند مثل فعل های اضافه یا از،ان،بود و..... قیدها،..... عموما درهمه اسناد هستند عموما سندهای که اینارو دارن زیادن وقتی زیادن یعنی معنا ی زیادی نمیدن ولی برعکس اون کلمه زیزی گولو که خیلی کم رخ میده بنابراین وزن بیشتری باید بگیره درکوئری تا کلمه ی که وزن عام تری دارد .پس اینکه توی جاکارد گفتیم که این تواشتراکات نمیبینه کدوم کلمه حذف شده واهمیت کلمات رونگاه نمیکنه اینه پس باید اهمیت کلمات روببینه پس اهمیت کلمات یامعنای کلمات میبینهکلمات کمیاب اطلاعات ببشتری رو به ما میرسونن تا اونایی که پرتکرارن که منظورشون تکرار در اینجا درمجموعه اسناد

خب من میخوام وزن بیشتری به اونایی که رخدادکمتری دارند وبرعکس

ما موضوع document frequencuمطرح کردیم وباdf نشون میدن که فاکتور جدید که برابرباتعداد اسنادی که در collection که شامل termیا حاویterm

dfمیشه داکیومنت فری کونسی

idf :inferce document frequency

idfهرچه بیشتر باش بهتر

tfهرچه بیشتر باش بهتر

dfهرچه کمتر باش بهتر

دوستان عزیز لطفا یه ربع اخر فیلم رو ببینید ب دلیل خرابیlmsانتونستم فرمول رو بنویسم ومثالش موفق باشید

جلسه پنجم تاریخ ۰۰/۰۲/۱ درس بازیابی اطلاعات

جلسه پیش و جلسات قبل ما درمورد بازیابی پیشرفته اطلاعات بولین صحبت کردیم و جایی که Query ما از And وOrوNat ساخته میشوند به صورت منطقی وما از یک شاخص معکوس استفاده میکنی که این شاخص معکوس ارتباط کلمات را با اسنادی که داخل آنها است را برقرار میکند .میگوید یک کلمه در چه اسنادی است باید یکسری از Query مانند AndوOr وNat و ترکیب آنها را پاسخ بدهیم اما داستان از این قرار است که مدل بازیابی اطلاعات به ما یک مجموعه را بر میگرداند یک مجموعه از اسناد مرتبط به Query ولی هیچ نظر درمورد اینکه کدام یک از این اسناد بهتر است را به ما نمی دهد و هیچ حرفی برای گفتن در آنجا ندارد یک منطقی بود وقتی میبیند آن منطق در ان سند برقرار است جواب درست داده است و اگر ان منطق برقرار نباشد جواب نادرست داده است ولی در این وسط هیچ کاری با درصد و یک مقدار یا یک تیکه ای بهتر یا بدتر این حرفا ندارد ما میخواهیم یک سیستمی داشته باشیم که علاوه بر اینکه سندهای مرتبط را به کاربر نشان می دهد بگویید کدا م مرتبط تر است یک وزنی به این سندها بدهد براساس وزنی که به سندها میدهد ما سندها را مرتب کنیم ان سندی که ازهمه وزن بیشتری دارد جایگاه بالاتری را بگیرد و سریع تر و زود تر به کاربر نشان داده شود در اینجا مدل های مختلفی هستند که این کار رابرای ما انجام میدهند اسناد بر مبنای رابطه شان ومقدار ارتباطی که به Query دارند وزن میدهند میگویند چقدر یک سند به Query کاربر نزدیک است یا چقدر به ان موضوع مرتبط است و از این وزن ما استفاده میکنی وجایگاه و ترتیب سندهای را داخل لیستی که به کاربر میخواهیم نمایش بدهیم انجا مشخص می کند یکی از مهمترین مدل های بازیابی اطلاعات که این کاررامیخواهد انجام بدهد و به سندهای ما وزن بدهد مدل برداری است یا مدل فضای برداری است ومدل های زیادی مطرح شده اند مثل مدل های احتمال و.... اما یکی از مهمترین روش های بازیابی اطلاعات مدل برداری نیز میباشد کار این مدل چه

میتواند باشد یا چه کار انجام میدهد ما جلسه پیش گفتیم سندهای خود را تبدیل میکنیم به مجموعه ای ازکلمات اینجا میگوییم سند خود را تبدیل میکنم به یک بردار فرض ما براین است که بردارها را داخل ریاضیاتی خوانده ایم اگر فراموش کردید در اینجا در حد اینکه برای شما تکرار بشود بیان خواهد شد که تا چه حدی با بردارها سروکار داریم واز انها استفاده میکنیم پس VECTOR SPACE موضوع فصل ششم است برای اینکه ما بتوانیم ازvector space استفاده بکنیم تا به اسناد خود وزن بدهیم

سوال : به نظر شا چند تا فرمول وزن دهی به کلمات وجود دارد .

جواب : روش های مختلفی وزن دهی به کلمات داخل استاد وجود دارد.

خوب ما میرویم سراغ مدل های که به کلمات ما وزن میدهند و کلمات را تبدیل به بردار بکنیم پس ازچپ به راست به term هاخواهیم پرداخت که چگونه وزن میدهیم و بعد چگونه انها را میبریم روی مدل برداری که مدل برداری به ما کمک میکند که به سندها وزن بدهیم.

ماعموما وقتی وزن میدهیم به اسناد که یک زوج سند یاquery وزن میدهیم یعنی اینکه این سند خالی ما چقدرمرتبط است با این query..query یک داستانی است که به ان نمیپردازیم فقط میخواهیم بدانیم چگونه میخواهیم به اسناد خود وزن بدهیم وزن دادن مخصوص اسناد نیست بلکه به یک زوج سند وزن میدهیم که نشان دهنده یک مقدارارتباط ومناسب بودن یک سند با یکquery است عموما فرمول های که ما میخواهیم استفاده کنیم وبدست می اوریم دوست داریم این اعداد بین عدد صفر و عدد یک باشد خوبی بین عدد صفر و عدد یک این است که بزرگترین و کوچکترین عدد چند است .یک query چه مزیتی دارد برای یک سند براساس این وزن که به صفر یا یک است همان طور که گفتیم داخل srot میکنیم داخل یک سند تا انها را بتوانیم نشان بدهیم به یک کاربر

داستان از این قرار است که این وزن را چطور حساب کنیم اگر هیچ کلمه ای از سند داخل هیچquery نباشد باید صفر باشد هر چی رخداد یک کلمه در یک سند بیشتر باشد ان سند مرتبط تر است با ان query .البته گفتیم اگر در این مدل برداری سند و query را تبدیل به بردار کنیم در داخل همان دنیایی مجموعه ها زندگی میکردیم میتوانیم به عدد یک برسیم مثلا اگر من query را مقدار A بگیرم و سند را B بگیرم هر دوتا را مثل مجموعه در نظر بگیرم من میتوانم از یک فرمول به شکل این استفاده کنم $A \cup B / A \cup$ در اینجا میتوانیم اشتراک تعدادی که داخل query است را با سند حساب کنیم تقسیم بر اجتماع کنیم که size انها را در نظر گرفته باشیم این به من یک عدد میرساند که من براساس این عدد میتوانم سندهای خود رامرتب سازی بکنم نکته ای که اینجا وجود دارد این است که چقدر این فرمول برای ن دقیق است پس این فرول jacard در همان دنیایی مجموعه ها کار میکند اما خوب کیفیت ان طوری نیست که من میخواهم وقتی سند با query را با هم مساوی باشند عدد یک میشود واگر اشتراکی نداشته باشند عدد صفر میشود و ان مزیتی که ما گفتیم که بین عدد صفر و عدد یک باشد وجود داد پس این اشتراک انها تقسیم بر تعداد اجتماع انها است اما ایرادی که این فرمول دارد این است که وزن termها را در نظر نمیگیرد انگار نه انگار ان term که داخل اشتراک نیست یا یک term بدرد بخور است که وزن term ها را در نظر نمیگیرد به همین خاطر این فرمول اینجا جواب نمیدهد .هرچه یک term داخل یک سند بیشتر رخ داده باشد ان سند بیشتر با ان term در ارتباط است و ان سند بیشتر مرتبط است با ان term .

اولین پارامتر Term frequency را با TF نشان میدهی تعداد رخداد term(t) در سندD از این پارامتر میتوانیم استفاده کنیک برای وزن دهی چطور خیلی ساده اونایی که tf بالاتری دارند را وزن بالاتری بدهیم حالا اینجا یک نکته ای است که میگویم اگر یک term ی ۱۰ باردر یک سند رخ بدهد و یک termی ۱ بار رخ بدهد درسته ان termی که ۱۰ بار رخ داده است اهمیت بیشتری

دارد و وزن بیشتری میگیرد اما لزوما نه به اندازه ۱۰ برابر یک وقت در یک سیستم کلمه ای نامبرده شده است اما با ضمیر به ان مراجعه میشود یک وقت میبینیم همان قدر که کلمه اول است سند در مورد ان است و فقط از کلمات نزدیک با ان مترادف یا همسایه ها یا موضوعات مختلف به ان استفاده میشود من باید یک راهی داشته باشم که بتوانم این درصد ها را جوری درست کنم که ۱۰ برابر نباشد یکی از این روش های خیلی خوبی که من می ایم از این ضریب را درست میکند لگاریتم گرفتن است انگار ما از ان term ی که رخ داده است در داخل سندها لگاریتم بگیریم و اگر بزرگتر از عدد صفر باشد جواب عدد صفر است و اگر بزرگتر از صفر نباشد جواب تعریف نشده است و بعلاوه عدد یک میکنیم که لگاریتم عدد صفر نباشد اگر ما لگاریتم را بسط بدهیم لگاریتم (N) را میرسیم به یک سری عددی $1 + \frac{1}{2} + \frac{1}{3} + \cdots + 1/n$ –احتمالا زیاد دیده ای که لگاریتم را میتوانیم با یک سری زمانی نوشت یک دلیل است که ما از یک لگاریتم استفاده میکنیم برای محاسبه وزن کلمات .چراما در اینجا لگاریتم میگیریم میگویم وقتی که یک کلمه ای TF(1) دارد به محض اینکه در یک سند ظاهر شود خوب ان کلمه یک وزنی میگیرد که این سند در مورد این کلمه میخواهد حرف بزند که مفاهیم انها به هم نزدیک است که وزن یک میگیرند اما اگر دفعه اول امد وزن ان برابر با عدد یک است ولی لزوما دفعه دوم وزن ان برابر با عدد یک نیست بلکه کمتر از عدد یک است اما دومین رخداد کلمه در سند اینقدر اهمیت ندارد که اولین رخداد اهمیت دارد یعنی هر بار که ما رخداد کلمه را میبینیم یک کم از اهمیت ان کمتر است حالا که ما میتوانیم به کلمات ود وزن بدهیم میتوانیم به کل سند ود هم وزن بدهیم که نمره سند ما میشود مجموع وزن هایی که کلمات میگیرند به شرط اینکه در اشتراک باشند پس کلمات که در سند هستند و در query مشترک هستند پیدا میکنیم و وزن انهایی که لگاریتم میگیرند را حساب میکنیم و با ه. یک سیگما$\sum$ ساده میگیریم و انهایی را جمع میکنیم و این میشود این نمره ای که میخواهیم به سند بدهی با این روش من پاراتر term frequency را در نظر گرفته ام.

اما ن به دو صورت رخداد کلمات در کالکشن را حساب کنم ما کار نداریم که در یک سند چند بار یک کلمه رخ داده است بیاییم ببینیم کلمه در چندتا سند وجود دارد شند ها میشماریم در دوتا سند یا در n سند ان کلمه رخ داده است اینجا یک مفهومی به من میرساند عموما کلماتی که عام هستند و عمومی هستند مثل حرف های اضافی (است . بود. ان . انها . اگر) عموما تعداد سندهای که این ها را دارند زیاد هستند وقتی زیاد هستند یعنی معنای زیادی به ما نمیدهند مثلا اگر کلمه پیروزی دریک سند باشد احتمال اینکه ان سند در مورد فوتبال باشد هست اا وقتی واقعاخود کلممه فوتبال در یک سند باشد احتمالش خیلی بیشتر میشود کلمیاب و کم تکرار اهمیت بیشتری را به ما میرسانند تا اونایی که پر تکرار هستند و منظور شان از تکرار اینجا در یک سند نیست بلکه در مجموعه ای از اسناد است

جلسه پیش در مورد بازیابی برداری صحبت شد این جلسه در مورد
نحوه ارزیابی سیستم های بازیابی اطلاعات صحبت میکنیم
برای بهتر شدن سیستم باید آن را ارزیابی کنیم در هر مرحله .

به دو حالت میتوان سیستم را ارزیابی کرد
ارزیابی کارایی Efficiency : به عنوان مثال یک سیستم چه مصرف منابعی دارد مانند cpu ram storage و سرعت آن چگونه است

ارزیابی کارآمدیEffectiveness: شاید یک سیستم سریع باشد ولی سند های بدردنخوری را برای ما بیاورد پس کارآمدی مهم تر از کارایی میباشد(خوب بودن نتایج و مرتبط بودن نتایج)

نحوه ی مقایسه ی سیستم های IR از دید کارآمدی:
حالت اول: به عنوان مثال دو سیستم IR2, IR1 را میخواهیم با هم مقایسه کنیم
یک راه این است که query های مختلف را روی هر دو اجرا کرده و نتایج را با هم مقایسه کنیم و یا تعداد user های مختلف را روی هر دو امتحان کرده و نتیجه را ببینیم که خیلی راه خوبی و دقیقی نیست

حالت دوم: این راه که راه خیلی از کتاب های بازیابی اطلاعات هم هست ابتدا یک معیار مانند x را در نظر گرفته و هر دو سیستم را با این معیار خود میسنجد و پس از آن دو مقدار x1,x2 بدست می آید که میتوان آن ها را با هم مقایسه کرد
خوبی این حالت این است که ارزیابی را تا سیستم Xn هم میتوان ادامه داد و محدود نیستیم

حال که فهمیدیم راه دوم (استفاده از معیار) بهتر است انواع معیار را تحلیل خواهیم کرد

انواع معیار:

۱)معیار های رتبه بندی نشده unranked : این معیارها ساده تر هستند و کاری به رتبه نتایج ندارند و فقط نتیجه برایشان مهم است
۲)معیار های رتبه بندی شده ranked: در این معیار علاوه بر نتیجه رتبه بندی آن هم مهم است به عنوان مثال اگر نتیجه ی ما یک آرایه باشد این برایمان اهمیت دارد که عنصر ۳ رتبه بالاتری دارد یا ۱

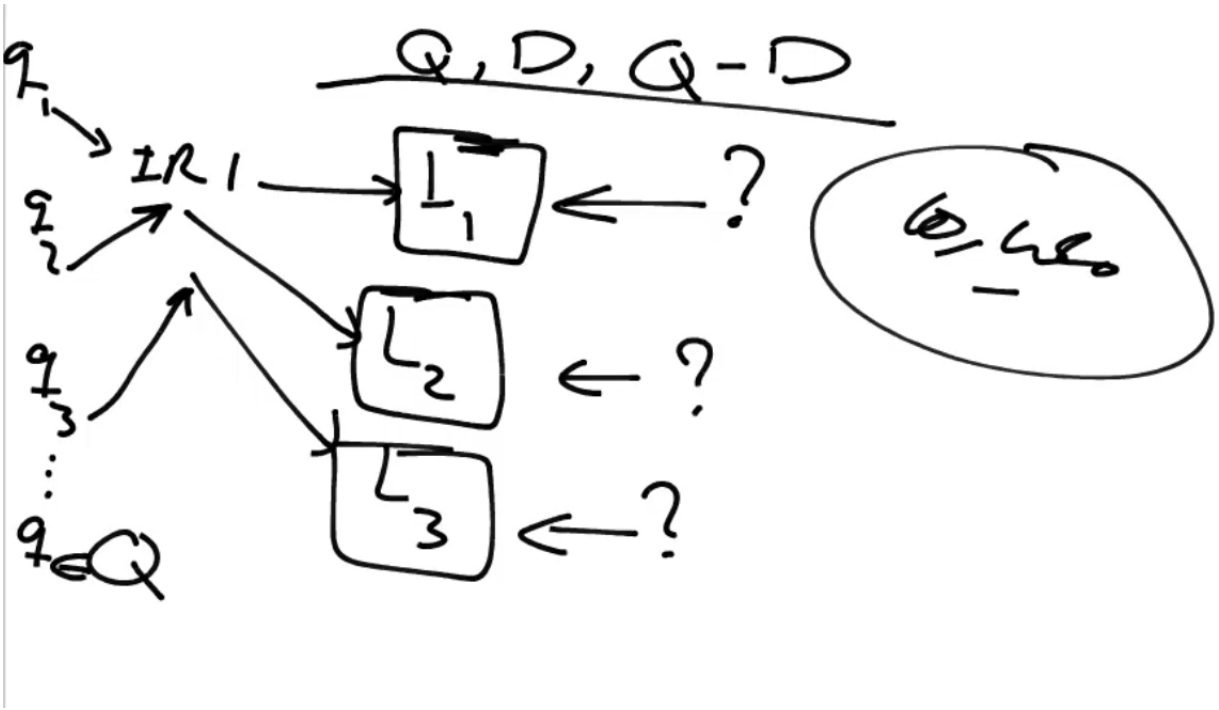
نکته : معیار های unranked بیشتر مورد استفاده قرار میگیرند

نحوه ارزیابی سیستم های IR رتبه بندی نشده :
ما برای ارزیابی و مقایسه نیاز به یک **benchmark** داریم

Benchmark از سه قسمت تشکیل شده است:
1: **corpus** : به این معنیست که من از قبل باید یک مجموعه ای اسناد یا document رو تعریف کنم (D)

2: **queries**: یک تعداد کوئری جهت ارزیابی اسناد باید تولید کنیم (Q)

3: **Q-D**: حال ما به عنوان یک انسان (داور) باید مشخص کنیم که هر کوئری مربوط به کدام اسناد میباشد و با آن در ارتباط است
به عنوان مثال ممکن ما به ازای هر 50 کوئری معادل میلیون ها سند داشته باشیم



به عنوان مثال در شکل در سیستم IR1 به ازای کوئری اول (q1) لیست یک (L1) برمیگردد حال میتوان این کوئری ها و لیست ها را در سیستم های دیگر بررسی و مقایسه کنیم
حال فرض کنید یک داور کوئری ها و لیست ها را بررسی کرده و به ازای همه ی کوئری ها (q) یک لیست جهت سنجش بدست آورده که به آن Lq میگوییم

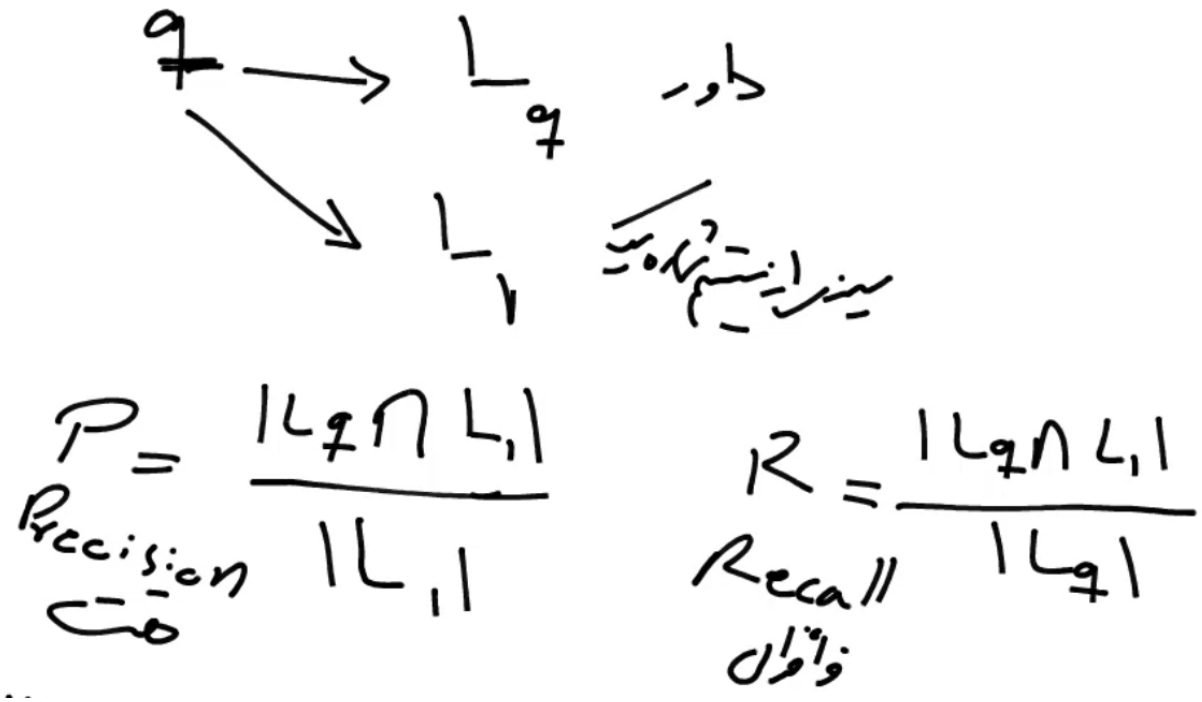
از نتایج بدست آمده از ارزیابی دو معیار برای ما تعریف میشود:

1)P(Precision) : دقت
2) R(Recall): فراخوانی

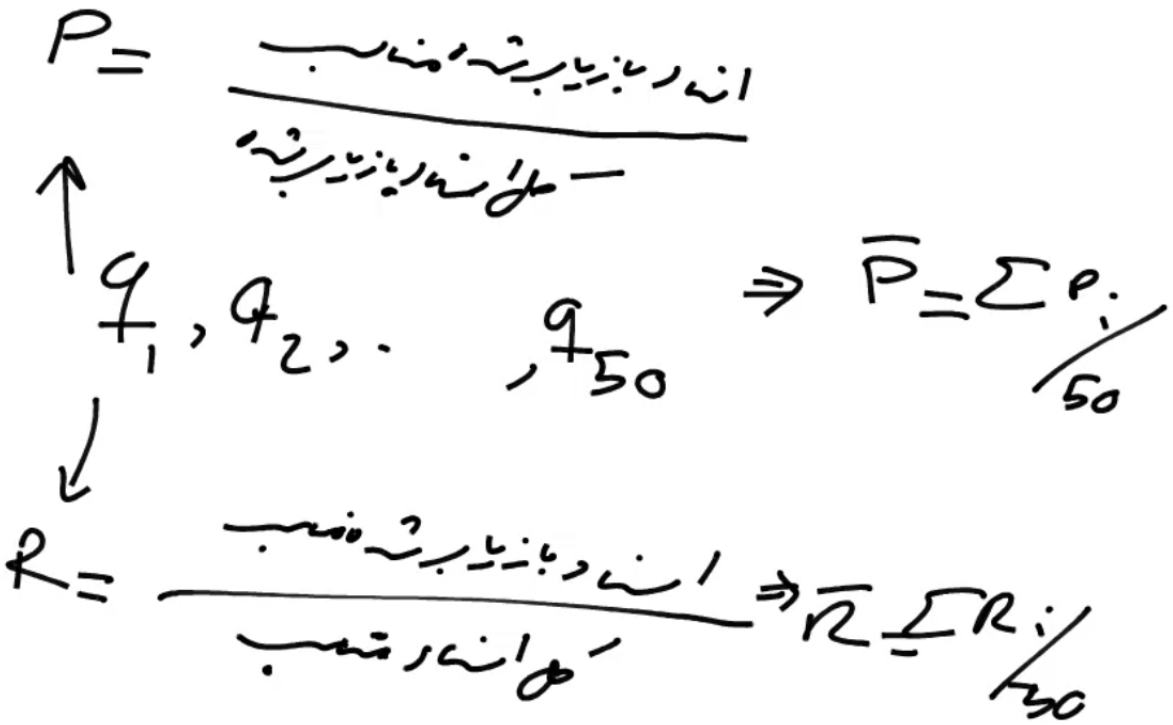
با توجه به شکل زیر P به معنی درصد اسناد مناسب بازیابی شده به کل اسناد بازیابی شده میباشد. به عنوان مثال ما در L1 (سند های بدست آمده از سیستم) 10 سند داریم

که 3 تا از اون ها با اسنادی که داور تایید کرده مشترک هستند پس بنابراین دقت ما میشود 30 درصد.

و در ادامه با توجه به شکل زیر R به معنی درصد اسناد مناسب بازیابی شده به کل اسناد مناسب میباشد به عنوان مثال داور ها 2 سند را مناسب تشخیص داده اند (Lq) و ما 100 سند از سیستم شماره ی یک به دست آورده ایم (L1) و پس از بررسی متوجه شده ایم فقط در یک مورد وجه اشتراک دارند . پس فراخوانی سند ما برابر میشود با 50 درصد.

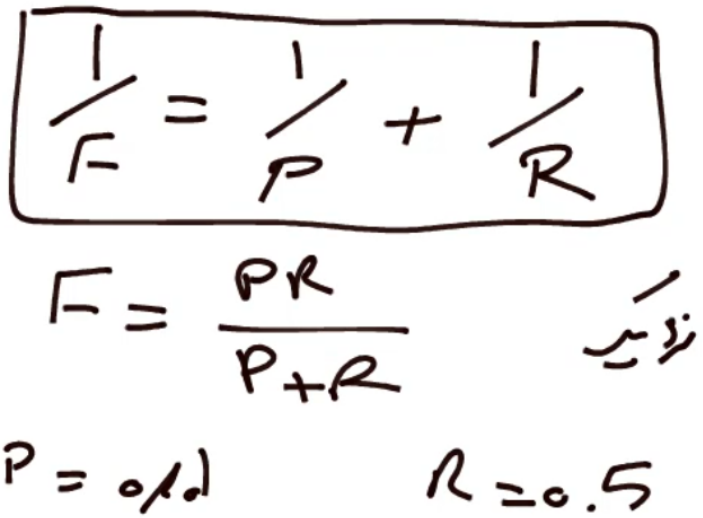


حال فرض کنید این روند را برای 50 تا سیستم محاسبه کنیم و از Pها و Qها میانگین بگیریم تا دو عدد برای ارزیابی کل سیستم بدست بیاوریم(توضیح در شکل زیر)



حال اتفاقی که میافتد این است که ممکنه در دو سیستم R یکی بزرگتر از سیستم دیگری و P آن کوچکتر باشد و بالعکس . برای همین میتوانیم این دو معیار را با هم ترکیب کرده و به معیار F برسیم

و بهترین راه جهت بدست آوردن F-measure مطابق شکل زیر است



نکته: ممکن است لیست ما یک میلیون نتیجه داشته باشد و ما فقط بخواهیم 10 تا از اونهارو بررسی کنیم برای این کار مینویسیم

P@10
R@10

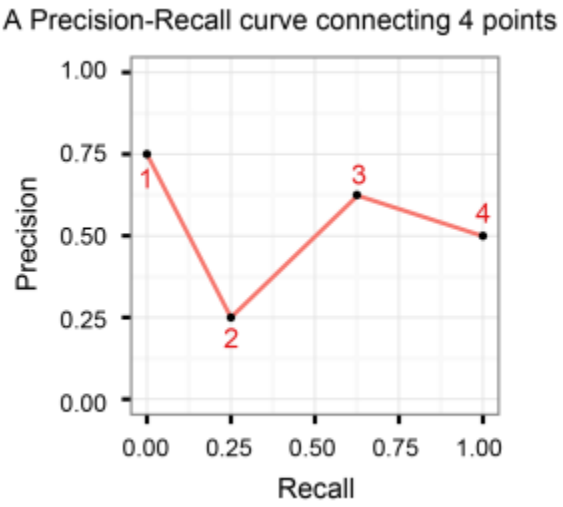
این چند روش برای بدست آوردن معیار در سیستم های unranked بود

یکی از راه های بررسی معیار در سیستم های رتبه بندی شده ranked این است که P@i ها و R@i ها رو با هم مقایسه کنیم
مثلا اگر P@1 یک سیستم همیشه نسبت به p@10 آن بالاتر باشد به این معنی است که همیشه نتیجه ی اول دقت بالایی نسبت به 10 تای بعدی دارد پس رتبه بندی به خوبی انجام شده است

در جلسه ی گذشته در مورد معیار های رتبه بندی شده ranked کمی صحبت شد

یکی دیگر از راه های بدست آوردن **معیار های رتبه بندی شده** استفاده از نمودار برحسب P و R میباشد

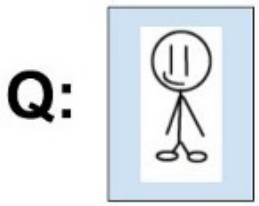
Threshold	Recall	Precision
1	0.0	0.75
2	0.25	0.25
3	0.625	0.625
4	1.0	0.5

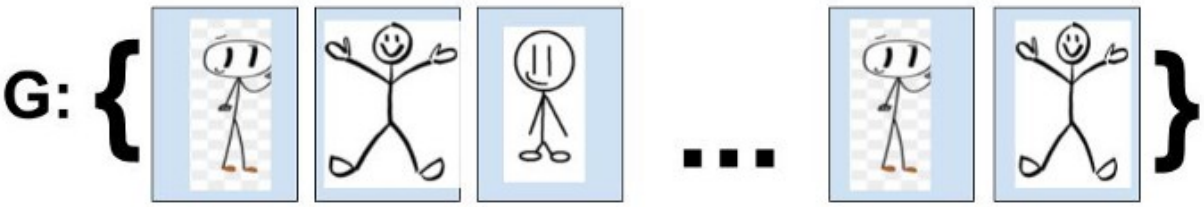


نکته: هرچقدر سطح زیر نمودار بیشتر باشه سیستم بهتری داریم. (در مورد R و P جلسه ی گذشته بحث شد)

یک روش کارآمد دیگر در بدست آوردن معیار های رتبه بندی شده استفاده از MAP میباشد
mean average precision

برای بدست آوردن mAP ابتدا باید AP (average precision) تمام کوئری ها را حساب کرده و در نهایت میانگین همه ی آن ها را بگیریم تا mAP بدست بیاید





به عنوان مثال ما در عکس بالا Q را به عنوان سند مناسب و G به عنوان اسناد بازیابی شده در نظر میگیریم و پس از بررسی به نتیجه ی زیر میرسیم
همانطور که مشاهده میشود در اینجا فقط سه نتیجه ی قابل قبول داشتیم و از یه جایی به بعد همشون رد میشن

حال بایستی P@i را با اعداد مختلف بدست بیاوریم

$$P@1 = 1/1 = 1$$

$$P@2 = 1/2 = 0.5$$

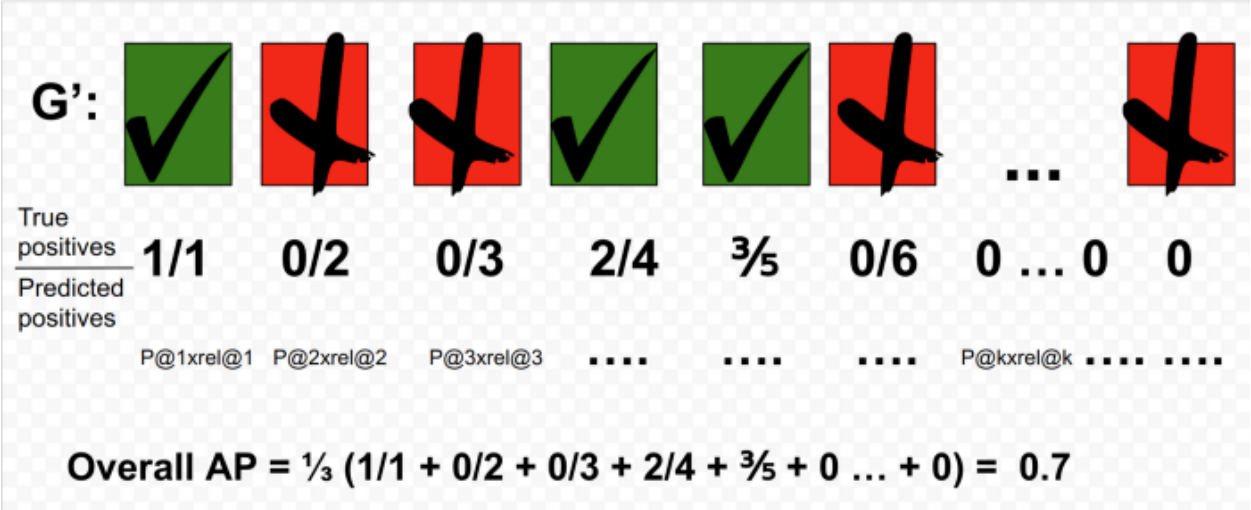
$$P@3 = 1/3 = 0.33$$

$$P@4 = 2/4 = 0.5$$

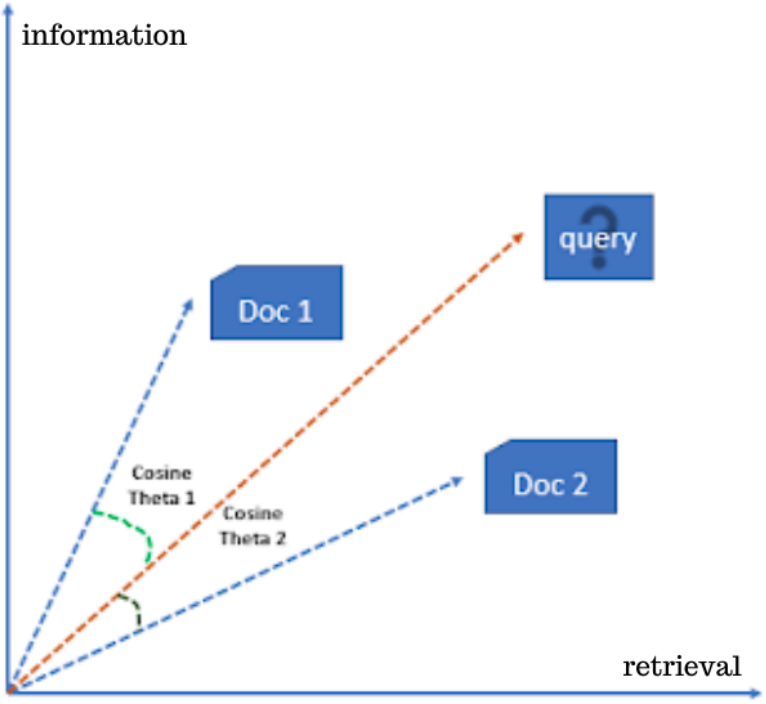
$$P@5 = 3/5 = 0.6$$

$$P@n = 3/n$$

حال میتوانیم AP را با اعداد در دسترس محاسبه کنیم



ادامه ی بازیابی اطلاعات برداری:
برای درک بهتر موضوع ابتدا فرض کنید ما تعدادی term داریم مثلا کلمات information و retrieval ترم های ما هستند برای جستجو حال بر اساس نیازمون میتوانیم یک query به دست بیاوریم مثلا در اینجا کلمه ی retrieval دو برابر کلمه ی information برای ما اهمیت دارد پس هر داکیومنتی که دریافت کردیم اگر کلمه ی retrieval داخلش بود به query ما نزدیک تر است



حال مطابق شکل فرض کنید داکیومنت های مختلفی داریم و میخواهیم نتایج را بسنجیم

این بدیهیست که هرچه زاویه نمودار کمتر باشد داکیومنت ما به query نزدیک تر است

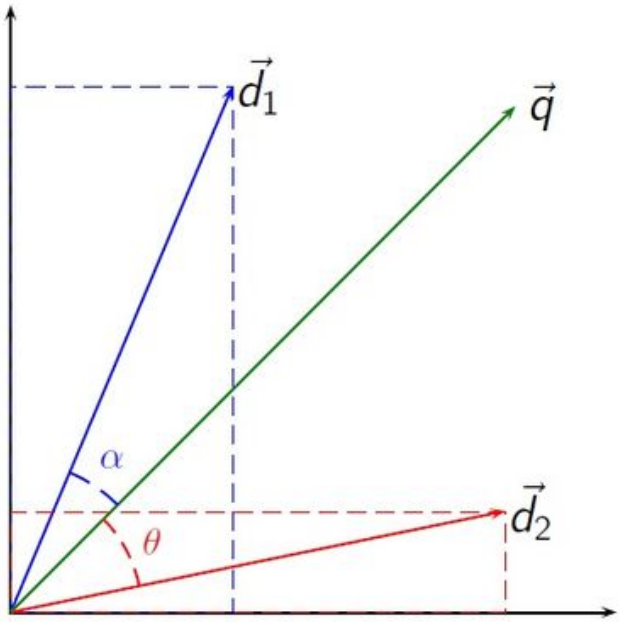
برای بدست آوردن اختلاف آن ها میتوانیم به آن ها وزن دهیم

به عنوان مثال : وزن داکیومنت های مختلف

$$d_j = (w_{1,j}, w_{2,j}, \ldots, w_{t,j})$$

وزن query:

$$q = (w_{1,q}, w_{2,q}, \ldots, w_{n,q})$$



به عنوان مثال میخواهیم زاویه ی بین q و d2 را بدست بیاوریم. با استفاده از ضرب بردار ها به روی اندازه ی آن ها کسینوس زاویه به دست می آید

$$\cos\theta = \frac{\mathbf{d_2} \cdot \mathbf{q}}{\|\mathbf{d_2}\| \, \|\mathbf{q}\|}$$

حال فرض کنید از شما خواسته شده تا similarity یک داکیومنت با کوئری رو محاسبه کنید به عنوان اولین قدم باید وزن های هر کدوم محاسبه بشه و سپس این عملیات را برای محاسبه کسینوس انجام بدیم

$$\cos(d_j, q) = \frac{\mathbf{d_j} \cdot \mathbf{q}}{\|\mathbf{d_j}\| \, \|\mathbf{q}\|} = \frac{\sum_{i=1}^{N} w_{i,j} w_{i,q}}{\sqrt{\sum_{i=1}^{N} w_{i,j}^2} \sqrt{\sum_{i=1}^{N} w_{i,q}^2}}$$

مثال: ما دو تا داکیومنت داریم

Document 1: (cat, runs, behind, rat)
Document 2: (Dog, runs, behind, cat)

و یک کوئری

Query: (rat)

با توجه به اینکه پنج کلمه ی متفاوت داریم پس 5 ترم در نظر میگیریم و وزن هارو محاسبه میکنیم

Term document matrix			
words\documents	Document1	document2	query term
cat	1	1	0
runs	1	1	0
behind	1	1	0
rat	1	0	1
dog	0	1	0

حال که وزن ها محاسبه شد به راحتی با جایگذاری similarity هر داکیومنت به دست میاید

جلسه ۹

تا به حال در مورد بازیابی اطلاعات کلاسیک صحبت کردیم در این نوع بازیابی یک کوئری داده می شود و یک لیست از اسناد مرتبط به کاربر نشان داده می شود.

حال به سراغ سیستم بازیابی اطلاعات تعاملی interactive می رویم، یعنی یکسری تعامل بین کاربر و سیستم جود دارد. کاربر کوئری می دهد ، سیستم سند می دهد. کاربر فیدبک میدهد و سیستم بهبود میدهد.

Relevance feedback & Query expansion

Relevance : مرتبط بودن سندها با کوئری

Best known relevance feedback method : Rocchio feedback

Rocchio باعث بهبود لیست اسناد بازگشتی می شود و عموما علت این موضوع اضافه کردن کلمات مرتبط است

Rocchio در فضای برداری کار می کند.

مثلا : برای دانش آموز : کلمات مرتبط = نیمکت – مدرسه – میز و ...

۲ راه برای افزایش کیفیت بازیابی اطلاعات وجود دارد:

۱- Relevance feedback
۲- Query expansion

نکته : این روش ها recall را افزایش می دهند در حالیکه گاهی اوقات باعث کاهش precision می شود

نکته: Query expansion در فضای برداری معنی دارد پس در فضای برداری and معنی ندارد کلمات در کنار jaguar قرار می گیرد و هر چه کلمات بیشتر شود اسناد بیشتر خواهد شد.

برای استفاده از این مفاهیم دو دید متفاوت می توانیم داشته باشیم:

۱- Local : این دید در حد کوئری کاربر است : Relevance feedback

۲- Global: در این دید کوئری کنار گذاشته می شود و به طور کلی چه دیکشنری ، مفاهیم و ... مطرح می شوند.

اصول اولیه Relevance feedback

- کوئری ها کوتاه هستند.
- موتور جستجو لیستی از اسناد را باز می گرداند.
- کاربران یکسری از اسناد را به عنوان مرتبط و سری دیگر را به عنوان اسناد نامرتبط مارک می کنند. فیدبک : ۱-صریح ۲- ضمنی (روش بهتر)
- در ارائه موتور جستجو ارائه جدیدتری نمایش می دهد
- موتور جستجو کوئری جدید اجرا می کند و نتایج جدید باز می گرداند.

محاسبات :

چون در فضای برداری هستیم برای محاسبه مرکز مجموعه از نقاط centroid ، مختصات نقاط را با هم جمع می کنیم و تقسیم بر تعداد نقاط می کنیم.

$$\vec{\mu}(D) = \frac{1}{|D|} \sum_{d \in D} \vec{v}(d)|$$

مرکز

مجموعه اسناد d

Rocchio قصد دارد به دنبال کوئری بهینه ساز شده برود .

$$\vec{q}_{opt} = \arg\max_{\vec{q}}[\text{sim}(\vec{q}, \mu(D_r)) - \text{sim}(\vec{q}, \mu(D_{nr}))]$$

D_r : مجموعه اسناد مرتبط

D_{nr} : مجموعه اسناد نامرتبط

Sim = similarity

به عبارت دیگر:

$$\vec{q}_{opt} = \mu(D_r) + [\mu(D_r) - \mu(D_{nr})]$$

نکته: سیستمی به نام smart وجود دارد که Rocchio را به صورت علمی استفاده می کند.

جلسه ۱۰ تاریخ ۱۴۰۰/۳/۱۹

گفتیم که query expansion حالت گلوبال دارد برعکس relevance feedback که حالت لوکال دارد

- کوئری اکسپنشن مانند relevance feedback یک روشی برای بالا بردن Recall هست
- منظور از global query expansion این است که این کوئری تغییر پیدا میکند و modify میشود بر مبنای یک منبع گلوبال مثلا دیکشنری
- یک منبع گلوبال آن داده هایی هستند که وابسته به کوئری نیستند
- Relevance feedback وابسته به کوئری هستند (query-dependent) و global query expansion وابسته به کوئری نیستند (query-independent)
- مبنای اصلی query expansion این هست که از کلمات هم معنی استفاده بکنیم مثلا یک کاربر یک کلمه مثل database سرچ میکند و یک سند خیلی خوب هم برایش وجود دارد اما کلمه database توش نیست و درمورد دستور select وquery وsql و این چیز هاست. دراینجا سند شامل آن مفهوم میباشد و کاربر چاره ای ندارد جز سرچ این مفهوم، هرچند در خود سند این کلمه وجود ندارد.

- دیتابیسی که این کلمات هم معنی را جمع آوری میکند که هم دستی هم به صورت مشتق شده اسناد تحت وب ساخته میشود، thesaurus نام دارد.
- مثل آمدن کلمه بیمارستان در کنار کلمه پزشکی
- که عموما recall را بالا میبرد
- ممکن است برای کلماتی که ابهام دارند به صورت قابل توجهی precision را پایین بیاورد

در اسلاید جدید مطلبی که باهاش سرکار داریم در این جلسه در رابطه با مفاهیم و محتواست

یک سری اسلاید ها رو استاد رد کرد تا قسمت latent semantic indexing

ایده اصلی این قسمت اینست که بجای اینکه ما اسناد را به کلمات map بکنیم، اسناد را به مفاهیم map بکنیم.

همینطور کلمات رو هم map میکنیم به مفاهیم

پس مفاهیم بین document و term قرارمیگیرند

مثلا مفهومی به اسم dbms مرتبط است با کلماتی مثل Data، system، retrieval (هرکدوم یک وزنی میگیرند مثلا Data وزن بیشتری میگیرد چون با مفهوم dbms خیلی مرتبط است)

LSI شبیه یک Thesaurus اتوماتیک کار میکند. ممکن است اسنادی را بازیابی کنیم که آن کلمه مورد نظر را نداشته باشند اما با ارتباطاتی که بین کلمات هست سند هایی را به ما نشان میدهد

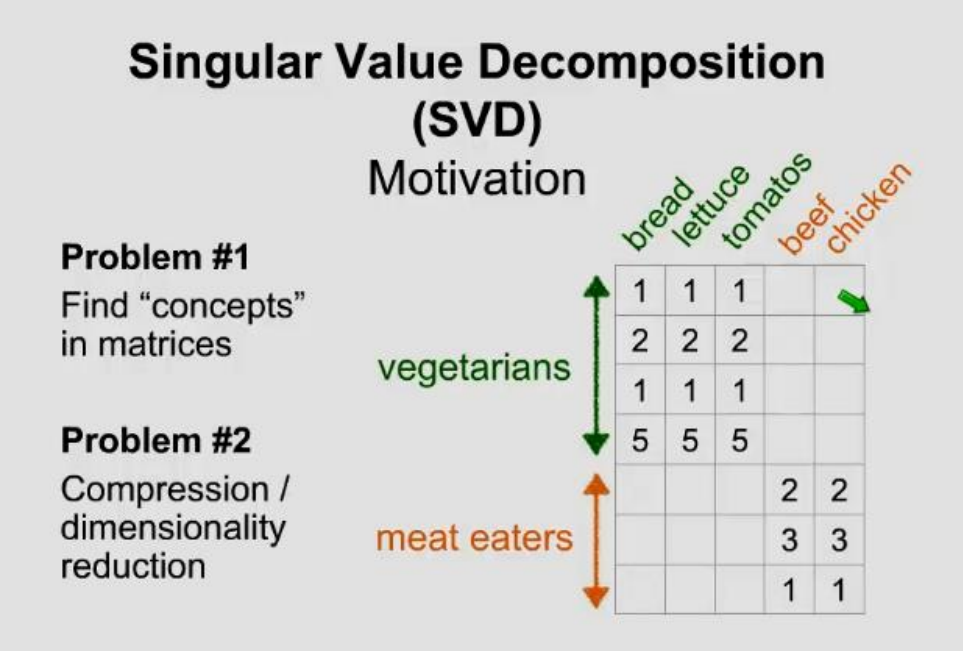
مثلا کاربر دنبال یک سندی میگردد که کلمه system درونش باشد LSI سند هایی را به او نشان میدهد که درون آن data و retrieval دارد. پس LSI به مفهوم کلمه "system" نگاه میکند و چک میکند که مفهوم system با مفهوم data و retrieval مرتبط است و دنبال آن اسناد مرتبط میگردد و به کاربر نمایش میدهد.

- concept هارا از اسناد درمیاوریم
- یک thesaurus اتوماتیک میسازیم

- بجای اینکه درمورد کلمات صحبت بکنیم درمورد concept ها صحبت میکنیم(اینکار باعث کاهش ابعاد ماتریس ها میشود چون عموما تعداد کانسپت ها از تعداد ترم ها خیلی کمتر است).

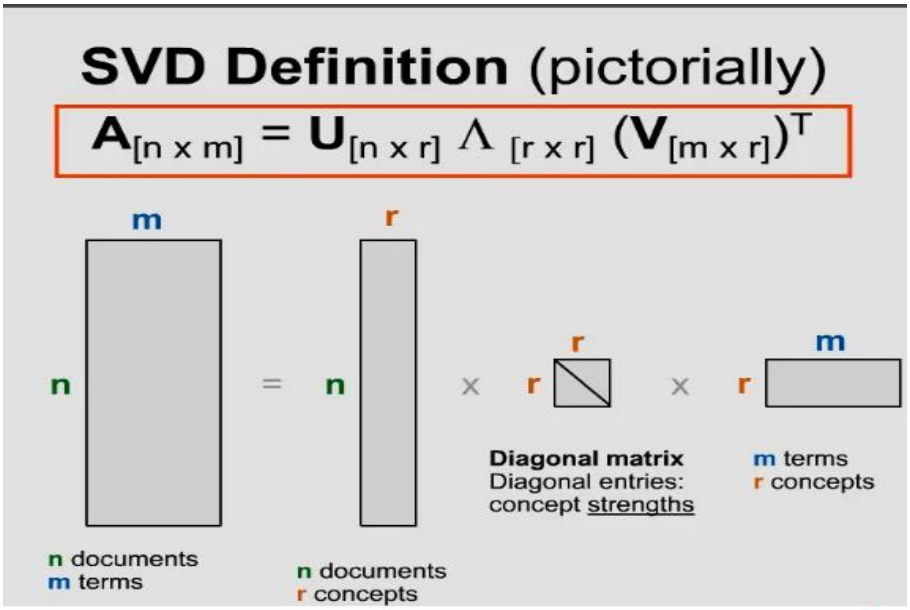
Singular value decomposition(svd)

در این شکل دو مدل سند داریم با دو دسته کلمات



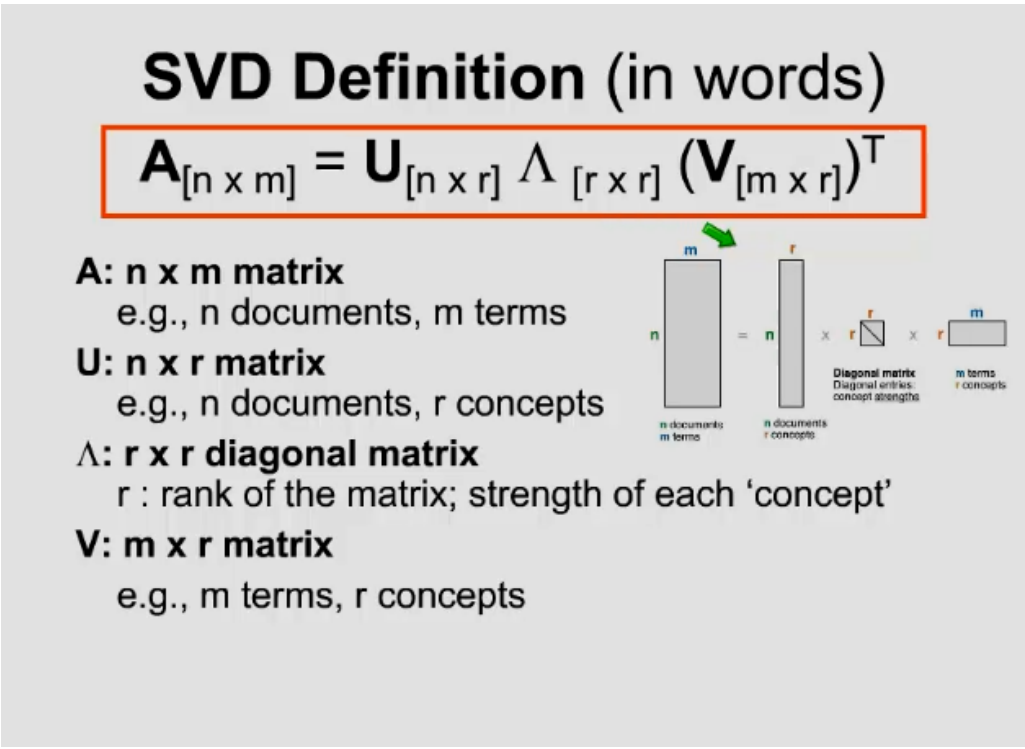
حالا امکان داره بهمون بگن ابعاد این ماتریس رو کاهش بدید و فشرده سازی انجام بدید

SVD یک ماتریس را به سه ماتریس تجزیه میکند



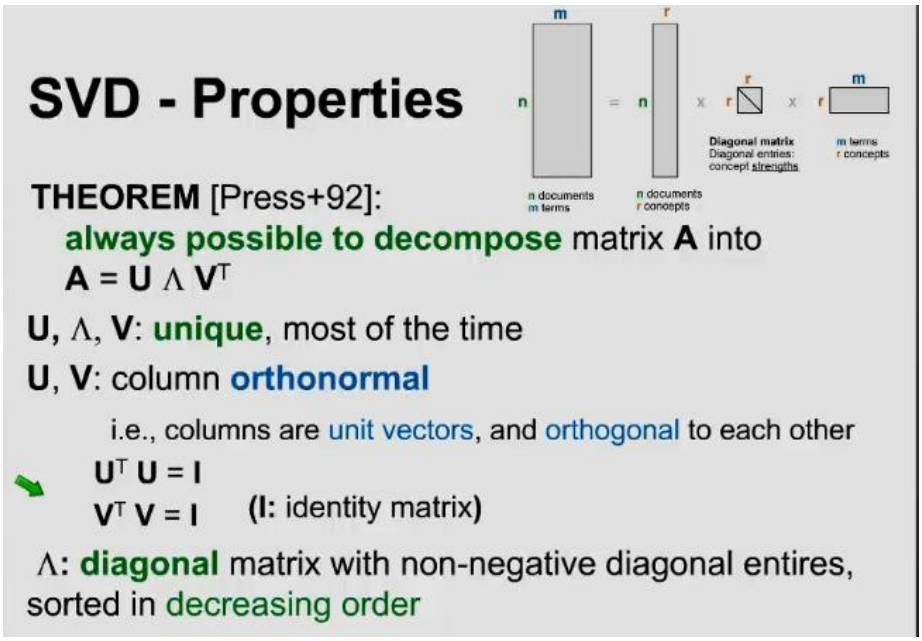
m تا داکیومنت داریم و n تا ترم و r تا کانسپت (سطر ها n و ستون ها m)

قطر این ماتریس وزن مفهوم هارا به ما نشان میدهند مثلا درایه ۲,۲ یعنی وزن مفهوم شماره ۲

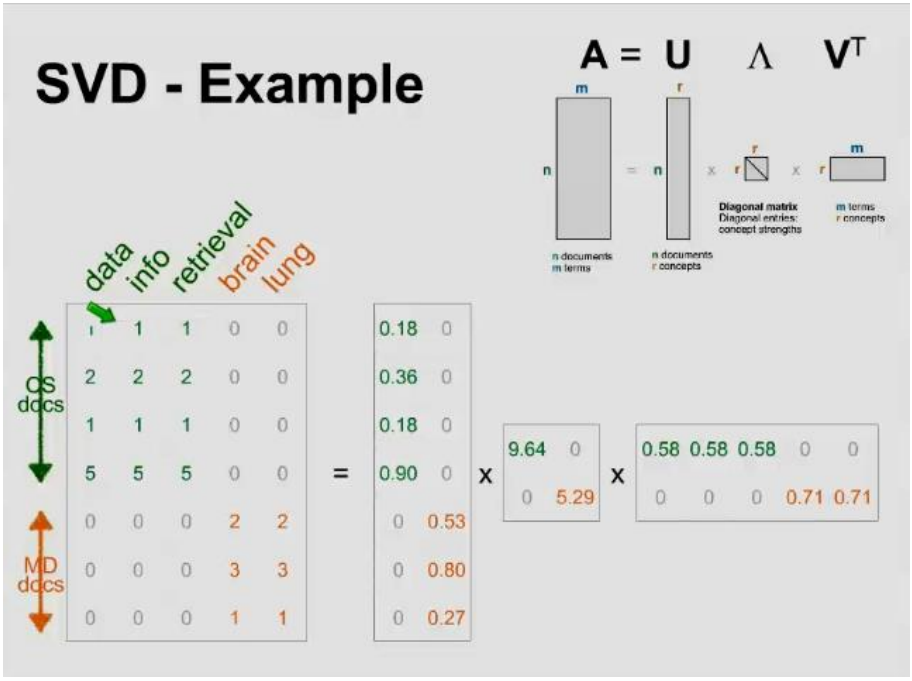


یک تئوری داریم در جبرخطی که میگوید میتوانیم همیشه ماتریس A را به دو ماتریس U و Vترانهاده تجزیه کنیم

و ماتریس U و V ستون هایشان عمود برهم هستند

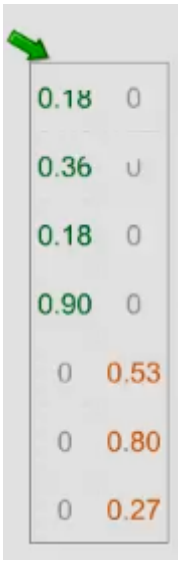


برمیگردیم به مثالمون



وقتی ماتریس سمت چپ را به برنامه خود(پایتون یا متلب) میدهیم سمت راست را به ما میدهد

و یعنی داکیومنت اولمون (رنگ سبز) در رابطه با مفهوم اول وزنش مقدار دارد و در رابطه با مفهوم دوم عدد صفر گرفته اند و همچنین داکیونت دوم در رابطه با مفهوم دوم مقدار های عددی خود را دارند(نارنجی) و در رابطه با مفهوم اول مقدار صفر گرفته اند.



ماتریس دوم (از سمت راست) قدرت هر مفهوم را نشان میدهد.

یک نکته در مورد ماتریس قطری این هست که باید از بالا به پایین مرتب باشد(بزرگ به کوچک)

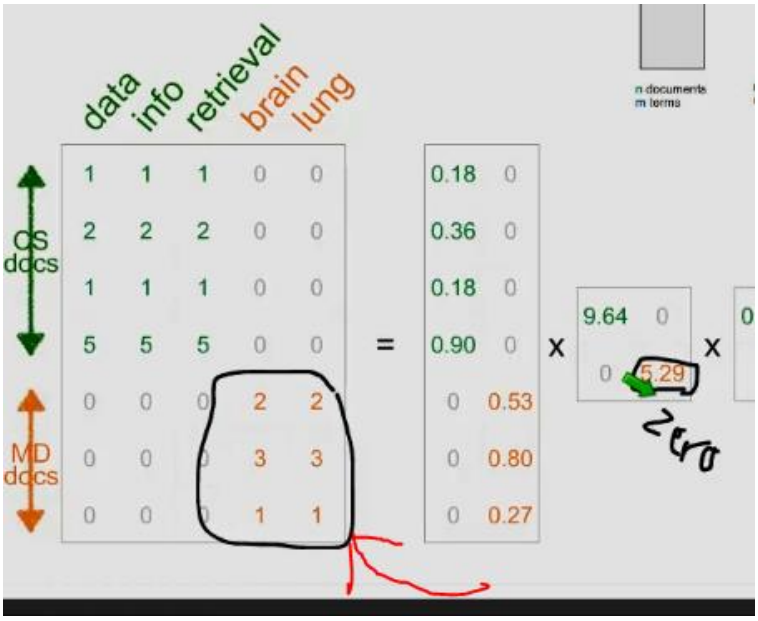
سوال: اگر عدد نارنجی در این ماتریس بجای ۵,۲۹ عدد بزرگی مثل ۲۰ بود باید چه کاری انجام داد؟

پاسخ: باید ردیف های نارنجی را با جای ردیف های سبز در ماتریس سمت چپ جابجا کنیم به طوری که نارنجی ها کلا بالا باشند. با اینکار اطلاعات سند بهم نمیریزد. حالا ستون هارا نیز جابجا میکنیم یعنی همه نارنجی ها را میفرستیم سمت چپ و همه سبز هارا میفرستیم سمت راست.

پس ماتریس به اینصورت میشود که بجای اینکه ابتدا data را بدهیم brain را میدهیم.(اینکار هارا وقتی میکنیم که آن ماتریس قطری نزولی مرتب شده است)

سوال: اگر در همان ماتریس دوم از سمت راست بجای عدد ۵,۲۹ عدد صفر قرار دهیم چه اتفاقی رخ میدهد؟

پاسخ: اینکار یعنی یک مفهوم را از سند حذف کرده ایم یعنی خود به خود این مقادیر(با فلش قرمز مشخص شده) نیز صفر میشوند.



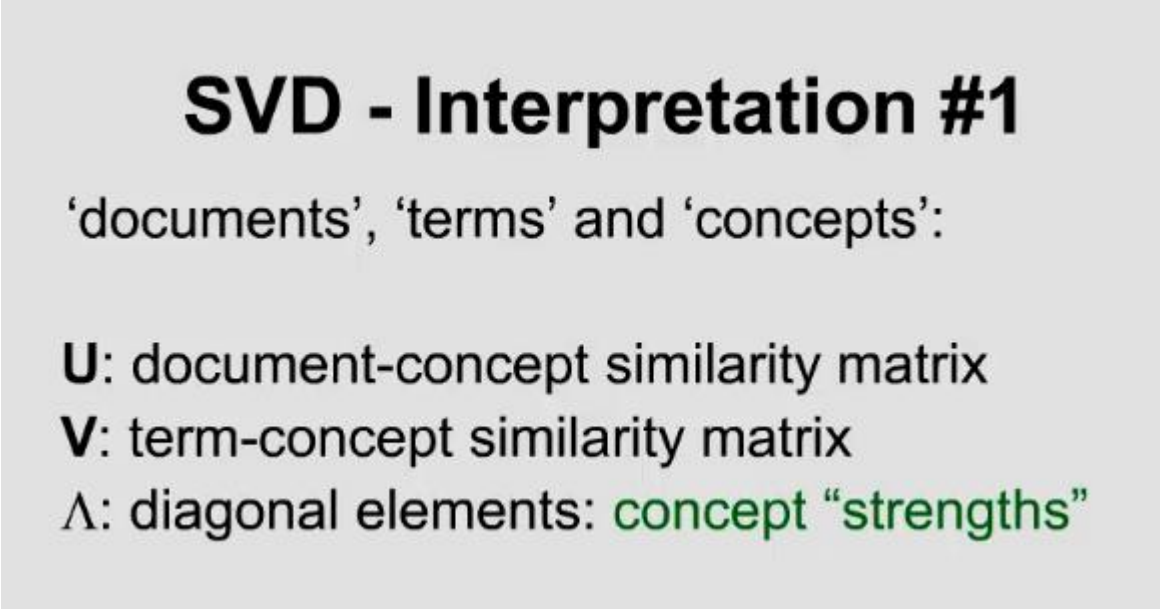
خب حالا یک بحثی پیش میاد فرض کنید عدد ۵,۲۹ عددی کوچک باشد مثلا ۰,۰۱ این به چه معناست؟

به این معناست که این مفاهیم خیلی ضعیف هستند و بهشون شک داریم و نویز حساب میشوند در نتیجه برای کاهش ابعاد ما انتهای ماتری قطری (اعداد کوچک در پایین و انتهای ماتریس) را صفر درنظر میگیریم و در اخر ضرب را انجام میدهیم تا ماتریس A ساخته شود.

به این ترتیب ما فقط مفاهیم اصلی را نگه میداریم و نویز هارا حذف میکنیم درنتیجه ابعاد به صورت چشمگیر کاهش پیدا میکند.

دلیل مفهوم similarity برای ماتریس document-concept این هست که چون concept و document بردار هستند پس وزنی که در این ماتریس می آید یک similarity می باشد.

تفاسیر مختلف از svd



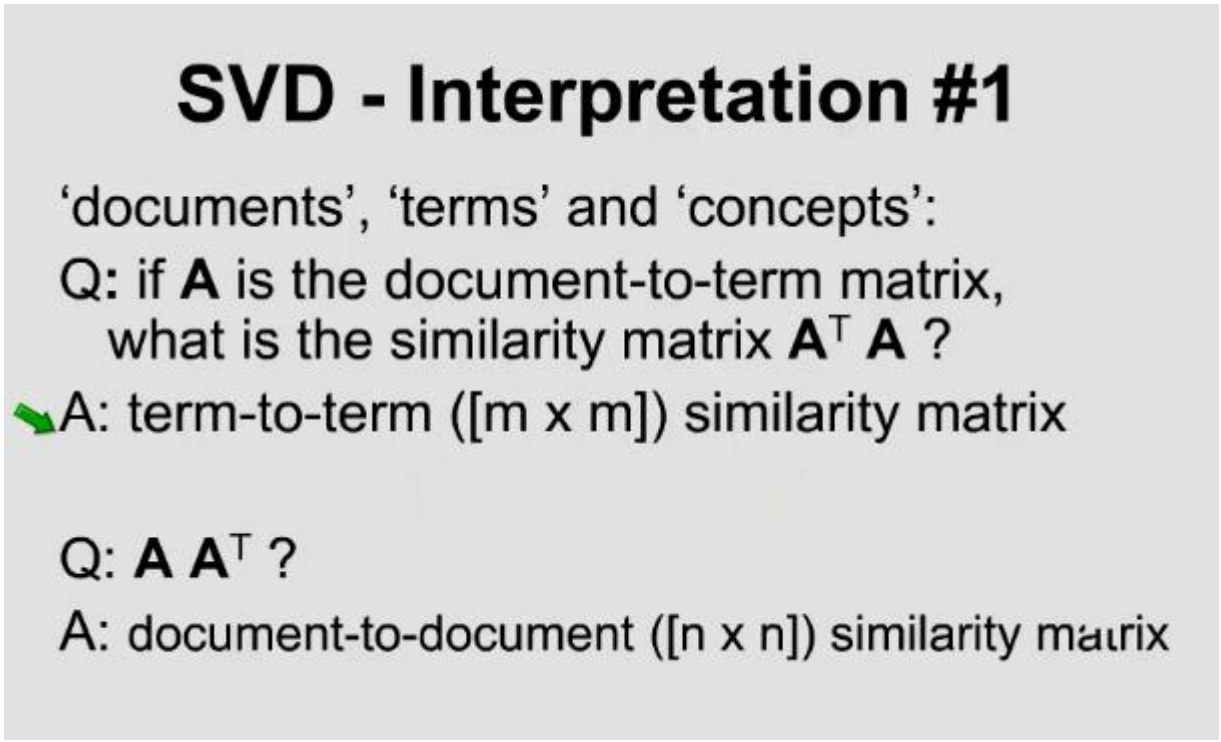
سوال پایین

اگر ماتریس A داکیومنت-ترم باشد، AA^T چه خواهد بود؟

پاسخ: میشود شباهت داکیومنت ها با داکیومنت ها

سوال اگر ماتریس A داکیومنت-ترم باشد، A^TA چه خواهد بود؟

پاسخ: میشود شباهت ترم ها با ترم ها



حالا اگر A ما داکیومنت ترم نبود و مثلا ترم داکیومنت بود آنموقع برعکس میشد یعنی AA^T میشد ترم در ترم و A^TA میشد شباهت داکیومنت با داکیومنت

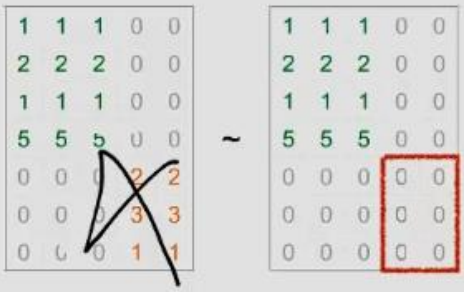
استاد یک سری اسلاید هارو رد شد

SVD - Interpretation #2

More details

Q: how exactly is dim. reduction done?

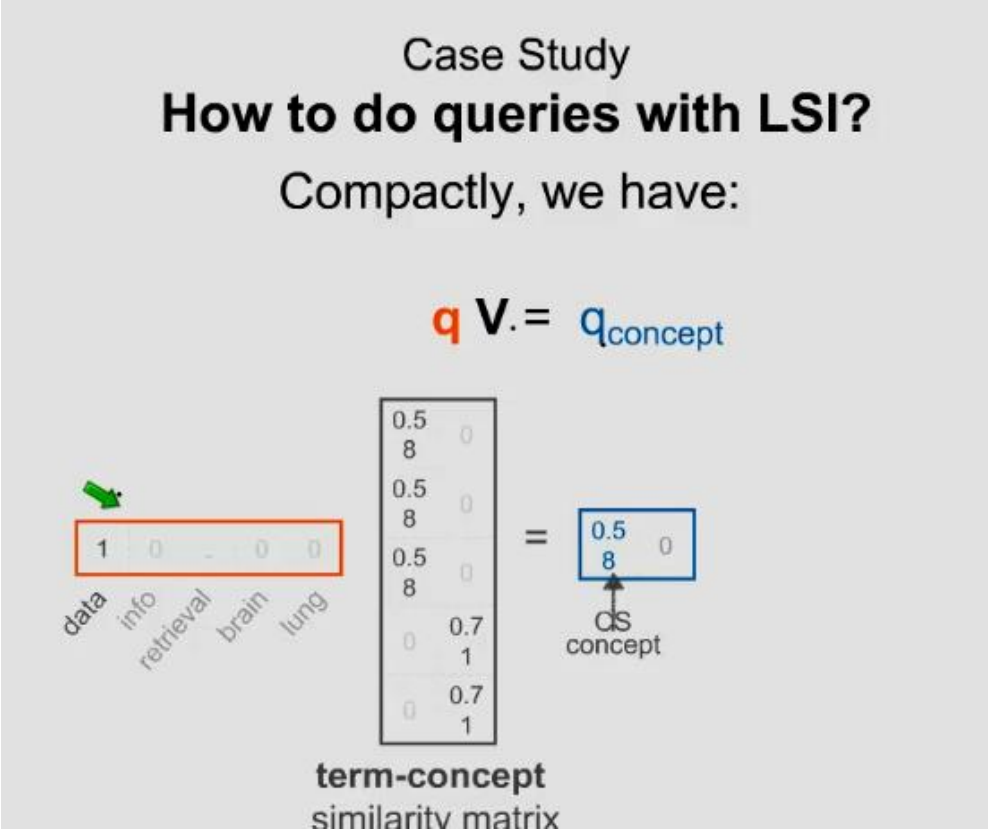
A: set the smallest singular values to zero:



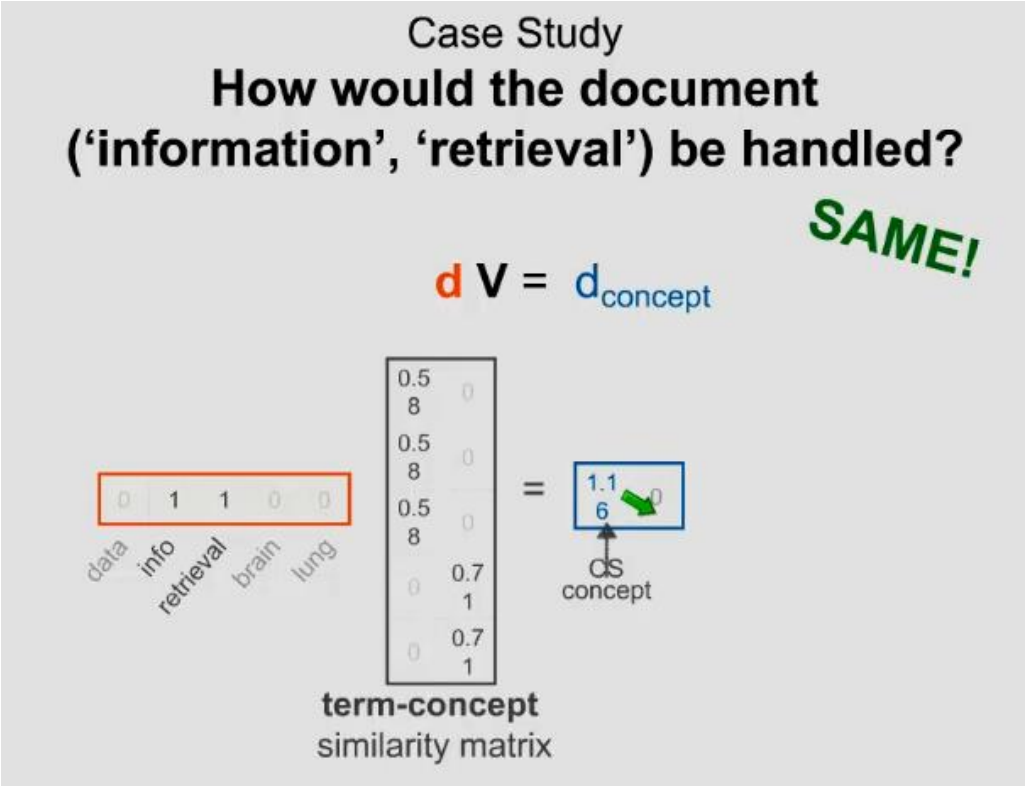
SVD - Interpretation #3

- finds non-zero 'blobs' in a data matrix =
- 'communities' (bi-partite cores, here)

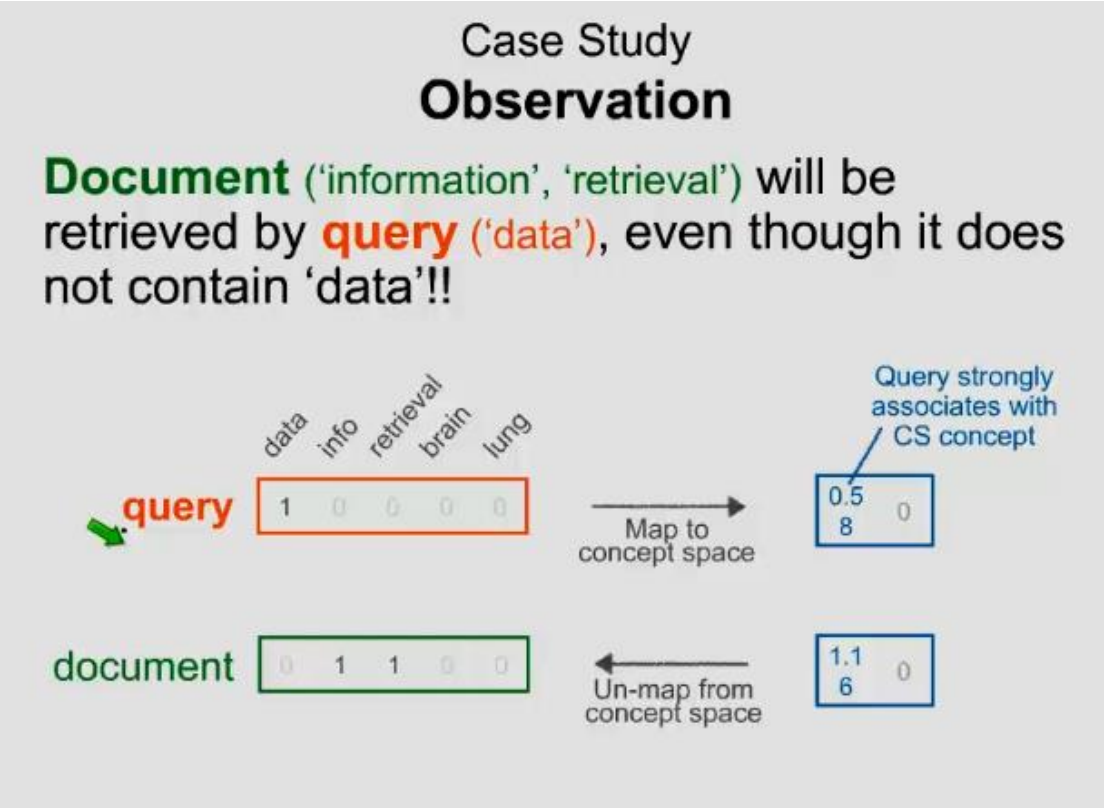
پس ما ترم را درV ضرب میکنیم و مفاهیم را به ما میدهد یعنی عدد ۱ در سمت چپ ارتباطش با هرکدام از ترم ها ضرب میشود و حاصل ها باهم جمع میشوند و و نتیجه (۰,۵۸) خواهد شد یعنی ارتباطی که این ترم ما با این مفهوم q دارد ۰,۵۸ است.



داکیومنت ها هم به همین صورت d را در v ضرب میکنیم تا کانسپت دربیاد و مقدار ارتباط داکیومنت ما با این مفهوم در شکل زیر ۱,۱۶ خواهد بود



در این شکل کلمه data در سند نیامده اما چون با مفهوم ما رابطه دارد(۰,۵۸) و در سند هم با مفهوم ارتباط داریم (۱,۱۶) درنتیجه میتوانیم این سند را به عنوان نتیجه کوئری کاربر نمایش دهیم.



تا الان درمورد بازیابی اطلاعات کلاسیک صحبت کردیم در جلسه ی آخر قرار است در مورد بازیابی در دنیای وب صحبت شود.
در بازیابی وب یکی از مهم ترین موارد ارتباط بین اسناد هست که به آن گراف وب میگوییم که بسیار پیچیده است.

Web Search

Advances & Link Analysis

Meta-Search Engines

موتور جستجویی که خودش اسناد رو index نمیکنه و از چند موتور جستجوی دیگه استفاده میکنه برای این کار

- Search engine that passes query to several other search engines and integrate results.
  - Submit queries to host sites.
  - Parse resulting HTML pages to extract search results.
  - Integrate multiple rankings into a “consensus” ranking.
  - Present integrated results to user.
- Examples:
  - Metacrawler
  - SavvySearch SavvySearch
  - Dogpile

روال کارش به این صورته که query خودش رو داخل سایت های مختلف میزنه سپس اونهارو پارس میکنه و بعد رتبه بندیشون میکنه

HTML Structure & Feature Weighting

- Weight tokens under particular HTML tags more heavily:
  - <TITLE> tokens (Google seems to like title matches)
  - <H1>,<H2>… tokens
  - <META> keyword tokens
- Parse page into conceptual sections (e.g. navigation links vs. page content) and weight tokens differently based on section.

Bibliometrics: Citation Analysis

بحث ما در این اسلاید ها بیشتر مربوط به انالیز لینک است و کاری به محتوای آن ندارد به عنوان مثال مقاله ی ما ممکن است به چند مقاله ی دیگر لینک شود و تشکیل یک گراف دهد و ما به آنالیز آن بپردازیم

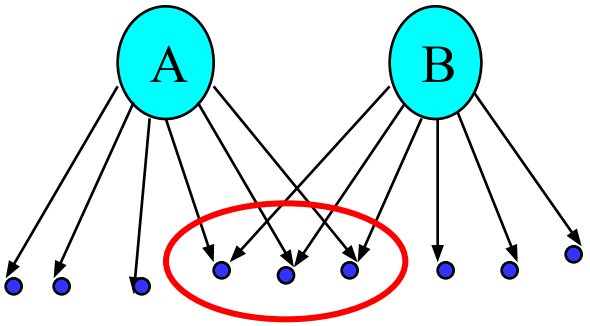
- Many standard documents include *bibliographies* (or *references*), explicit *citations* to other previously published documents.
- Using citations as links, standard corpora can be viewed as a graph.
- The structure of this graph, independent of content, can provide interesting information about the similarity of documents and the structure of information.
- CF corpus includes citation information.

Impact Factor

- Developed by Garfield in 1972 to measure the importance (quality, influence) of scientific journals.
- سال 1972 یک معیار و میزانی ایجاد شد تا مشخص کند یک مجله یک علمی چقدر کیفیت و طرفدار دارد
-
- Measure of how often papers in the journal are cited by other scientists.
- معیارشون هم به این شکل بود که این مقالات چقدر توسط سایر افراد استناد داده میشن
-
- Computed and published annually by the Institute for Scientific Information (ISI).
- همه ی اینها سالانه توسط موسسه ISI محاسبه و منتشر میشد
- The *impact factor* of a journal *J* in year *Y* is the average number of citations (from indexed documents published in year *Y*) to a paper published in *J* in year *Y*−1 or *Y*−2.
- Does not account for the quality of the citing article.
- در اینجا تعداد استناد ها برایمان مهم است نه کیفیت محتوای مقاله

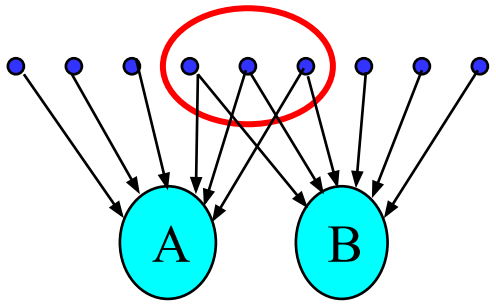
Bibliographic Coupling

- Measure of similarity of documents introduced by Kessler in 1963.
- معیاری برای محاسبه ی شباهت اسناد میباشد
- The bibliographic coupling of two documents *A* and *B* is the number of documents cited by *both A* and *B*.
- Size of the intersection of their bibliographies.
- Maybe want to normalize by size of bibliographies?
- در اینجا A,B هر دو به سه تا سند ارجاع دارند پس میتونیم بگیم شبیه هم هستند و هرچه اشتراک بیشتر باشد شباهت هم بیشتر است و ربطی به محتوا ندارد. با تقسیم اشتراک به اجتماع هم میتوانیم آن ها را نرمال کنیم



Co-Citation

- An alternate citation-based measure of similarity introduced by Small in 1973.
- Number of documents that cite both *A* and *B*.
- Maybe want to normalize by total number of documents citing either *A* or *B* ?
-
- Co-Citation یک معیار دیگر سنجش شباهت است با این تفاوت که در اینجا باید همه ی مقالات بررسی شوند که آیا به A,B استناد داده شده اند یا خیر . پس از لحاظ محاسبه سنگین تر از قبلی است



Citations(استناد) vs. Links(لینک)

- Web links are a bit different than citations:
  - لینک ها و استناد ها علی رغم شباهت هایشان متفاوتند
  - Many links are navigational.
    - لینک ها قابلیت این رو دارند که ما از هر صفحه ای به صفحه ی دیگر برویم
    -
  - Many pages with high in-degree are portals not content providers.
    - لینک ها شبیه پورتال هستند و هدف محتوایی ندارند
  - Not all links are endorsements.
    - همه ی پیوند ها مورد تایید نیستند
  - Company websites don't point to their competitors.
    - وب سایت های کمپانی ها به رقبای خود اشاره نمیکنند بر خلاف Citations
  - Citations to relevant literature is enforced by peer-review.

Authorities محتوای خوب

- *Authorities* are pages that are recognized as providing significant, trustworthy, and useful information on a topic.
  - صفحاتی مشخص هستند که یک منبع اطلاعاتی خوب و مفید برای یک موضوع خاص میباشند
- *In-degree* (number of pointers to a page) is one simple measure of authority.
  - همین باعث میشود که تعداد مراجعه به آن صفحه بیشتر شود و هرچه *In-degree* بالاتر باشد authority نیز بالاتر است
- However in-degree treats all links as equal.
- Should links from pages that are themselves authoritative count more?

Hubs

- *Hubs* are index pages that provide lots of useful links to relevant content pages (topic authorities).
- Hub pages for IR are included in the course home page:
  - http://www.cs.utexas.edu/users/mooney/ir-course

هاب ها برخلاف اسلاید قبلی به صفحاتی میگویند که لینک های مفید درآن جمع آوری شده باشد مانند لینک بالایی

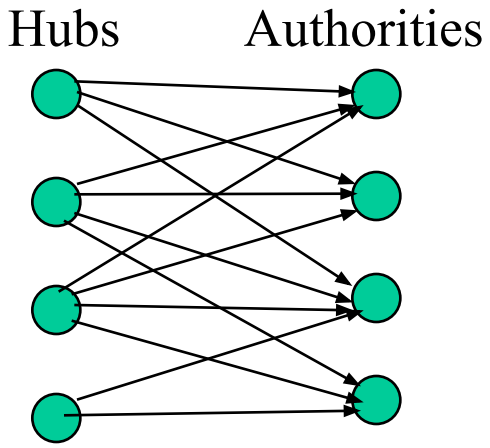
HITS

- Algorithm developed by Kleinberg in 1998.
  - این الگوریتم در همان سالی مطرح شد که الگوریتم گوگل مطرح شد
- Attempts to computationally determine hubs and authorities on a particular topic through analysis of a relevant subgraph of the web.
  - این الگوریتم هاب ها و authority های یک عنوان خاص را محاسبه میکند
- Based on mutually recursive facts:
  - Hubs point to lots of authorities.
  - Authorities are pointed to by lots of hubs.

منطق کلی این الگوریتم به این صورت است که هاب ها به تعداد زیادی authority اشاره میکنند و authority ها هم به هاب های زیادی اشاره دارند

Hubs and Authorities

- Together they tend to form a bipartite graph:

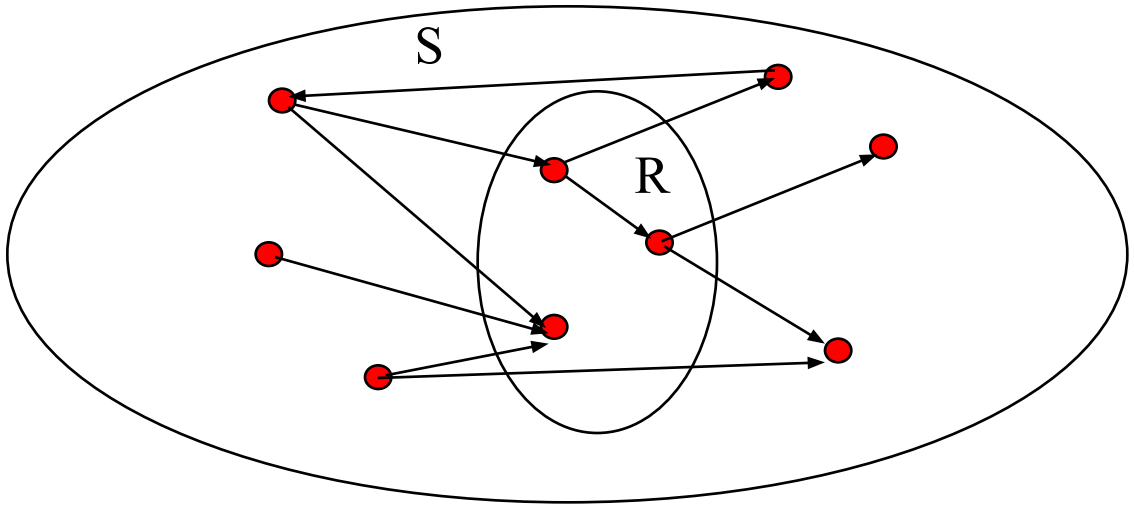


HITS Algorithm

- Computes hubs and authorities for a particular topic specified by a normal query.
- First determines a set of relevant pages for the query called the *base* set *S*.
- Analyze the link structure of the web subgraph defined by *S* to find authority and hub pages in this set.

Constructing a Base Subgraph

- For a specific query *Q*, let the set of documents returned by a standard search engine (e.g. VSR) be called the *root* set *R*.
- Initialize *S* to *R*.
- Add to *S* all pages pointed to by any page in *R*.
- Add to *S* all pages that point to any page in *R*.



Base Limitations

- To limit computational expense:
  - Limit number of root pages to the top 200 pages retrieved for the query.
  - ابتدا ۲۰۰ صفحه ی اول را بررسی میکنیم
  - Limit number of "back-pointer" pages to a random set of at most 50 pages returned by a "reverse link" query.
  - هاب ها و authority ها را محاسبه میکنیم
- To eliminate purely navigational links:
  - Eliminate links between two pages on the same host.
- To eliminate "non-authority-conveying" links:
  - Allow only m ($m \cong 4{-}8$) pages from a given host as pointers to any individual page.

Authorities and In-Degree

- Even within the base set *S* for a given query, the nodes with highest in-degree are not necessarily authorities (may just be generally popular pages like Yahoo or Amazon).
- True authority pages are pointed to by a number of hubs (i.e. pages that point to lots of authorities).

Iterative Algorithm

- برای محاسبه ی HITS بایستی هاب ها و authority ها را ارزیابی کرده و جمع بزنیم
- Use an iterative algorithm to slowly converge on a mutually reinforcing set of hubs and authorities.
- Maintain for each page $p \in S$:
  - Authority score: a_p (vector $\boldsymbol{a}$)
  - Hub score: h_p (vector $\boldsymbol{h}$)
- Initialize all $a_p = h_p = 1$
- ابتدا هاب ها و authority ها را یک در نظر میگیریم و سپش آن ها را آپدیت میکنیم
- Maintain normalized scores:

$$\sum_{p \in S} (a_p)^2 = 1 \qquad \sum_{p \in S} (h_p)^2 = 1$$

HITS Update Rules

- Authorities are pointed to by lots of good hubs:
- جمع هاب ها به ماAuthorities را میدهد و برعکس

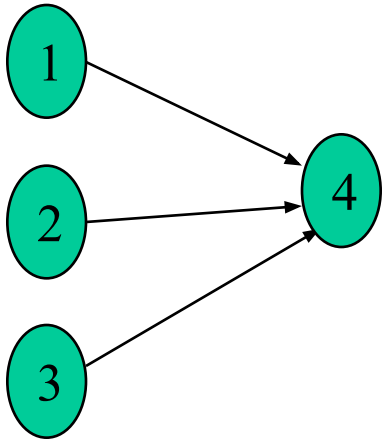
$$a_p = \sum_{q:q \to p} h_q$$

- Hubs point to lots of good authorities:

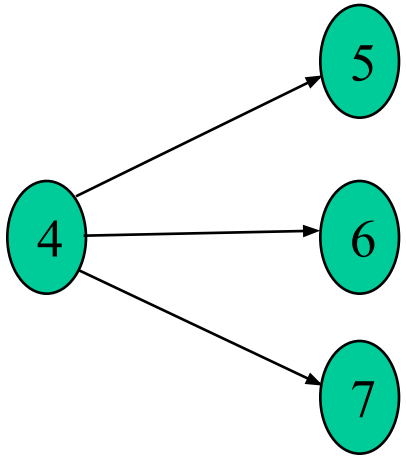
$$h_p = \sum_{q:p \to q} a_q$$

Illustrated Update Rules

- جمع هاب ها به Authoritiesها را میدهد و برعکس



$a_4 = h_1 + h_2 + h_3$



$h_4 = a_5 + a_6 + a_7$

HITS Iterative Algorithm

Initialize for all $p \in S$: $a_p = h_p = 1$

For i = 1 to k:

For all $p \in S$: $a_p = \sum_{q:q \to p} h_q$ (*update auth. scores*)

$$h_p = \sum_{q:p \to q} a_q$$

For all $p \in S$: (*update hub scores*)

$$\sum_{p \in S} \left(a_p / c\right)^2 = 1$$ (*normalize* **a**)

For all $p \in S$: $a_p = a_p / c$ c:

$$\sum_{p \in S} \left(h_p / c\right)^2 = 1$$ (*normalize* **h**)

For all $p \in S$: $h_p = h_p / c$ c:

Convergence

- Algorithm converges to a *fix-point* if iterated indefinitely.
- Define A to be the adjacency matrix for the subgraph defined by S.
  - $A_{ij} = 1$ for $i \in S, j \in S$ iff $i \rightarrow j$
- Authority vector, $\boldsymbol{a}$, converges to the principal eigenvector of $A^T A$
- Hub vector, $\boldsymbol{h}$, converges to the principal eigenvector of AA^T
- In practice, 20 iterations produces fairly stable results.

Results

- Authorities for query: "Java"
  - java.sun.com
  - comp.lang.java FAQ
- Authorities for query "search engine"
  - Yahoo.com
  - Excite.com
  - Lycos.com
  - Altavista.com
- Authorities for query "Gates"
  - Microsoft.com
  - roadahead.com

Result Comments

- In most cases, the final authorities were not in the initial root set generated using Altavista.
- Authorities were brought in from linked and reverse-linked pages and then HITS computed their high authority score.

Finding Similar Pages Using Link Structure

- Given a page, *P*, let *R* (the root set) be *t* (e.g. 200) pages that point to *P*.
- Grow a base set *S* from *R*.
- Run HITS on *S*.
- Return the best authorities in *S* as the best similar-pages for *P*.
- Finds authorities in the "link neighbor-hood" of *P*.

Similar Page Results

- Given “honda.com”
  - toyota.com
  - ford.com
  - bmwusa.com
  - saturncars.com
  - nissanmotors.com
  - audi.com
  - volvocars.com

HITS for Clustering

- An ambiguous query can result in the principal eigenvector only covering one of the possible meanings.
- Non-principal eigenvectors may contain hubs & authorities for other meanings.
- Example: “jaguar”:
  - Atari video game (principal eigenvector)
  - NFL Football team (2nd non-princ. eigenvector)
  - Automobile (3rd non-princ. eigenvector)

PageRank

- Alternative link-analysis method used by Google (Brin & Page, 1998).
- Does not attempt to capture the distinction between hubs and authorities.
- Ranks pages just by authority.
- Applied to the entire web rather than a local neighborhood of pages surrounding the results of a query.

Initial PageRank Idea

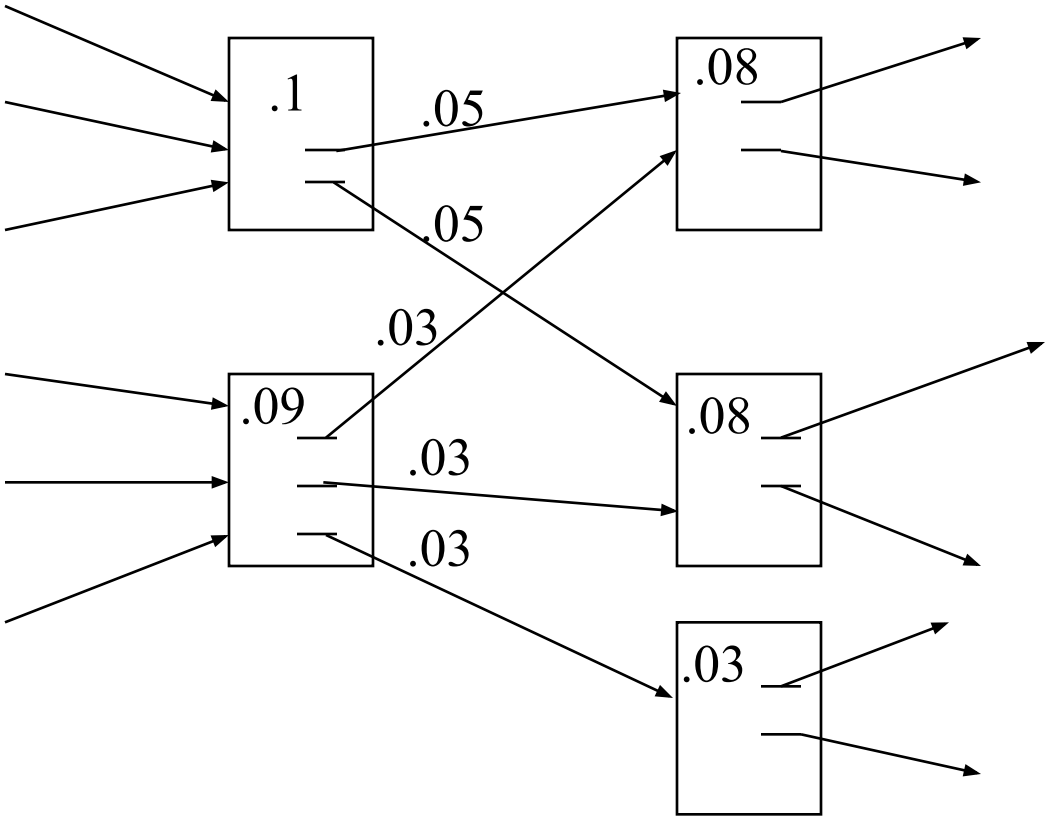
- Just measuring in-degree (citation count) doesn't account for the authority of the source of a link.
- Initial page rank equation for page p:

$$R(p) = c \sum_{q:q \to p} \frac{R(q)}{N_q}$$

  - N_q is the total number of out-links from page q.
  - A page, q, "gives" an equal fraction of its authority to all the pages it points to (e.g. p).
  - c is a normalizing constant set so that the rank of all pages always sums to 1.

Initial PageRank Idea (cont.)

- Can view it as a process of PageRank "flowing" from pages to the pages they cite.



Initial Algorithm

- Iterate rank-flowing process until convergence:

  Let S be the total set of pages.

  Initialize $\forall p \in S: R(p) = 1/|S|$

  Until ranks do not change (much) *(convergence)*

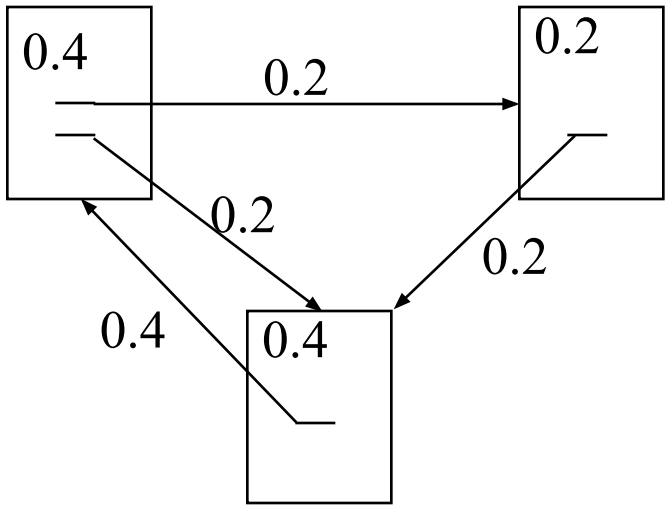
  For each $p \in S$:

$$R'(p) = \sum_{q:q \to p} \frac{R(q)}{N_q}$$

$$c = 1 / \sum_{p \in S} R'(p)$$

  For each $p \in S: R(p) = cR'(p)$ *(normalize)*

Sample Stable Fixpoint

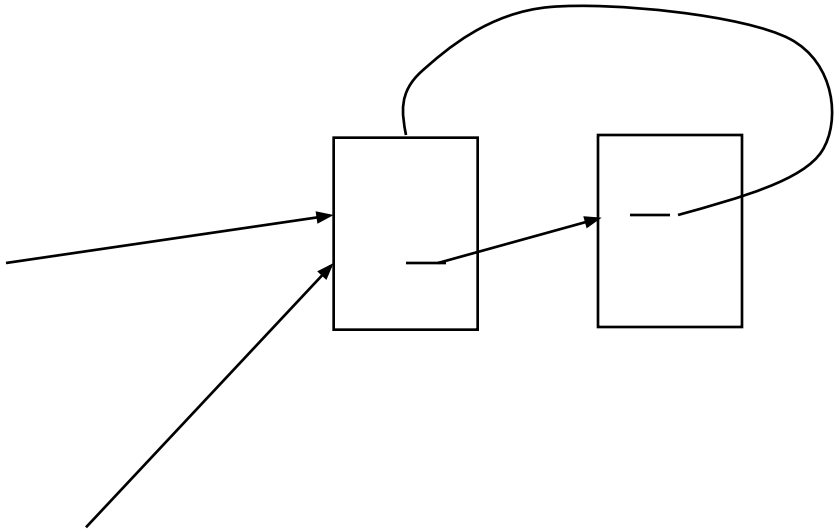


Linear Algebra Version

- Treat **R** as a vector over web pages.
- Let **A** be a 2-d matrix over pages where
  - $\mathbf{A}_{vu} = 1/N_u$ if $u \rightarrow v$ else $\mathbf{A}_{vu} = 0$
- Then $\mathbf{R} = c\mathbf{A}\mathbf{R}$
- **R** converges to the principal eigenvector of **A**.

Problem with Initial Idea

- A group of pages that only point to themselves but are pointed to by other pages act as a "rank sink" and absorb all the rank in the system.



Rank flows into
cycle and can't get out

Rank Source

- Introduce a "rank source" E that continually replenishes the rank of each page, p, by a fixed amount $E(p)$.

$$R(p) = c\left(\sum_{q:q \to p} \frac{R(q)}{N_q} + E(p) \right)$$

PageRank Algorithm

Let S be the total set of pages.

Let $\forall p \in S: E(p) = \alpha/|S|$ (for some $0<\alpha<1$, e.g. 0.15)

Initialize $\forall p \in S: R(p) = 1/|S|$

Until ranks do not change (much) (*convergence*)

For each $p \in S$:

$$R'(p) = \left[(1-\alpha) \sum_{q:q \to p} \frac{R(q)}{N_q} \right] + E(p)$$

$$c = 1 / \sum_{p \in S} R'(p)$$

For each $p \in S: R(p) = cR'(p)$ (*normalize*)

Linear Algebra Version

- $\mathbf{R} = c(\mathbf{AR} + \mathbf{E})$
- Since $\|\mathbf{R}\|_1 = 1$: $\mathbf{R} = c(\mathbf{A} + \mathbf{E} \times \mathbf{1})\mathbf{R}$
  - Where $\mathbf{1}$ is the vector consisting of all 1's.
- So $\mathbf{R}$ is an eigenvector of $(\mathbf{A} + \mathbf{E} \times \mathbf{1})$

Random Surfer Model

- PageRank can be seen as modeling a "random surfer" that starts on a random page and then at each point:
  - With probability $E(p)$ randomly jumps to page p.
  - Otherwise, randomly follows a link on the current page.
- $R(p)$ models the probability that this random surfer will be on page p at any given time.
- "E jumps" are needed to prevent the random surfer from getting "trapped" in web sinks with no outgoing links.

Speed of Convergence

- Early experiments on Google used 322 million links.
- PageRank algorithm converged (within small tolerance) in about 52 iterations.
- Number of iterations required for convergence is empirically O(log n) (where n is the number of links).
- Therefore calculation is quite efficient.

Simple Title Search with PageRank

- Use simple Boolean search to search web-page titles and rank the retrieved pages by their PageRank.
- Sample search for “university”:
  - Altavista returned a random set of pages with “university” in the title (seemed to prefer short URLs).
  - Primitive Google returned the home pages of top universities.

Google Ranking

- Complete Google ranking includes (based on university publications prior to commercialization).
  - Vector-space similarity component.
  - Keyword proximity component.
  - HTML-tag weight component (e.g. title preference).
  - PageRank component.
- Details of current commercial ranking functions are trade secrets.

Personalized PageRank

- PageRank can be biased (personalized) by changing **E** to a non-uniform distribution.
- Restrict "random jumps" to a set of specified relevant pages.
- For example, let $E(p) = 0$ except for one's own home page, for which $E(p) = \alpha$
- This results in a bias towards pages that are closer in the web graph to your own homepage.

Google PageRank-Biased Spidering

- Use PageRank to direct (focus) a spider on "important" pages.
- Compute page-rank using the current set of crawled pages.
- Order the spider's search queue based on current estimated PageRank.

Link Analysis Conclusions

- Link analysis uses information about the structure of the web graph to aid search.
- It is one of the major innovations in web search.
- It was one of the primary reasons for Google's initial success.