

СОФИЙСКИ УНИВЕРСИТЕТ СВ. КЛИМЕНТ ОХРИДСКИ
ФАКУЛТЕТ ПО МАТЕМАТИКА И ИНФОРМАТИКА

ИЗВЛИЧАНЕ НА ИНФОРМАЦИЯ

Записки към курса по „Извличане на информация“ четен от Иван Койчев

[Type the abstract of the document here.]

Съдържание

1 БУЛЕВО ИЗВЛИЧАНЕ.....	7
1.1 Пример на проблема за извличане на информация.....	8
1.2 Изграждане на обрънат индекс	11
1.3 Обработка на булеви заявки.....	13
1.4 Разширеният булев модел срещу ранкирано извличане	15
2 РЕЧНИК НА ТЕРМИНИТЕ И СПИСЪК НА ПУБЛИКАЦИИТЕ	17
2.1 Анализиране на документи и изграждане на последователности.....	17
2.1.1 Прочитане на документи	17
2.1.2 Избор на части от документ	17
2.2 Определяне на речник на термините.....	18
2.2.1 Процес за установяване на термини	18
2.2.2 Премахване на излишните термини.....	18
2.2.3 Нормализиране или уеднакяване на термините	19
2.2.4 Отчитане на това, че различни форми на думите могат да имат еднакво значение.....	21
2.3 Бързо сливане на постинг списъци чрез прескачащи указатели	21
2.4 Позиционни срещания и фразови заявки	23
2.4.1 Индекси на две думи (<i>Biword indexes</i>)	23
2.4.2 Позиционни индекси (<i>Positional indexes</i>)	24
2.4.3 Пример.	25
2.4.4 Размер на позиционните индекси.	25
2.4.5 Комбинирани схеми.....	26
3 РЕЧНИЦИ И ТОЛЕРАНТНО ИЗВЛИЧАНЕ.....	27
3.1 Структури за търсене в речници	27
3.2 Произволни заявки (WILDCARD QUERIES)	29
3.2.1 Обобщени произволни заявки.....	30
3.2.2 Permuterm индекс (индекс от разместени термини).....	30
3.2.3 К-грамни индекси за произволни заявки.....	31
3.3 Поправка на правописа	32
3.3.1 Прилагане на корекция на правописа.....	32
3.3.2 Видове поправки на правописа.....	33
3.3.3 Редакционно разстояние.....	33
3.3.4 К-грамни индекси за коригиране на правопис.....	35
3.3.5 Коригиране на правописа в зависимост от контекста.....	36
3.4 Фонетични корекции	37
3.5 Използвана и допълнителна литература	38
4 КОНСТРУИРАНЕ НА ИНДЕКСИ	39
4.1 Основи на хардуера	39
4.2 Блоково индексиране базирано на сортиране (BLOCKED-SORT-BASED INDEXING).....	40
4.3 Индексиране в паметта с едно обхождане (SINGLE-PASS-IN-MEMORY INDEXING)	41
4.4 Разпределено индексиране.	43
4.5 Динамично индексиране.....	44
5 КОМПРЕСИРАНЕ НА ИНДЕКС (INDEX COMPRESSION)	46
5.1 Статистически свойства на термините в извличането на информация.....	46
5.1.1 Закон на <i>Hearps</i> за определяне броя на термините	47
5.1.2 Закон на <i>Zipf</i> за разпределението на термините	48
5.2 Компресиране на речник.....	48
5.2.1 Речника като низ (<i>string</i>).	49
5.2.2 Съхранение в блокове.....	50
5.3 Компресия на POSTINGS FILE.....	51
5.3.1 <i>Variable byte codes</i>	51

5.3.2	<i>у кодиране</i>	52
6	ОЦЕНЯВАНЕ, ПРЕТЕГЛЯНЕ НА ТЕРМИНИ И ВЕКТОРНО-ПРОСТРАНСТВЕН МОДЕЛ	54
6.1	ПАРАМЕТРИЧНИ И ЗОНАЛНИ ИНДЕКСИ	54
6.1.1	<i>Оценяване чрез претеглени зони</i>	56
6.1.2	<i>Определяне на теглата</i>	57
6.1.3	<i>Оптималната тежест g</i>	58
6.2	ЧЕСТОТА НА ПОЯВЯВАНЕ НА ТЕРМИН И ПРЕТЕГЛЯНЕ	59
6.2.1	<i>Обърната документова честота (idf)</i>	60
6.2.2	<i>Tf-idf оценяване</i>	60
6.3	ВЕКТОРНО-ПРОСТРАНСТВЕН МОДЕЛ ЗА ОЦЕНЯВАНЕ	61
6.3.1	<i>Скалярно произведение</i>	61
6.3.2	<i>Търсенето като вектор</i>	63
6.3.3	<i>Изчисляване на векторни оценки</i>	63
6.4	ПРОМЕНЛИВИ TF-IDF ФУНКЦИИ	64
6.4.1	<i>Подлинейно tf мащабиране</i>	64
6.4.2	<i>Максимална tf нормализация</i>	64
6.4.3	<i>Схеми за определяне на тежестта на търсене</i>	65
7	ИЗЧИСЛЕНИЕ НА РЕЗУЛТАТИТЕ В ЗАВЪРШЕНА СИСТЕМА ЗА ТЪРСЕНИЕ	66
7.1	ЕФЕКТИВНО ТОЧКУВАНЕ И КЛАСИРАНЕ	66
7.2	Извличане на най-близките (релевантни) към документа (IN EXACT TOP K DOCUMENT RETRIEVAL)	67
7.3	ПРЕМАХВАНЕ НА ИНДЕКСИ (INDEX ELIMINATION)	68
7.4	ШАМПИОНСКИ СПИСЪЦИ (CHAMPION LISTS)	68
7.5	СТАТИЧНИ КАЧЕСТВЕНИ РЕЗУЛТАТИ И ПОДРЕЖДАНЕ (STATIC QUALITY SCORES AND ORDERING)	68
7.6	ПОДРЕЖДАНЕ СПОРЕД ВЛИЯНИЕТО (IMPACT ORDERING)	69
7.7	СЪКРЪЩАВАНЕ НА КЛЪСТЕРИТЕ (CLUSTER PRUNING)	69
7.8	КОМПОНЕНТИ НА СИСТЕМА ЗА ИЗВЛИЧАНЕ НА ИНФОРМАЦИЯ	69
7.8.1	<i>Слоести индекси</i>	69
7.9	Близост на термините от заявката	70
7.10	ПОСТРОЯВАНЕ НА ФУНКЦИИ ЗА ОЦЕНЯВАНЕ И ПРЕДАВАНЕ НА ЗАЯВКАТА	71
7.11	ИЗГРАЖДАНЕ НА ЦЯЛОСТНА ЦЯЛОСТНА СИСТЕМА ЗА ТЪРСЕНИЕ	71
7.12	МОДЕЛ НА ВЕКТОРНОТО ОЦЕНЯВАНЕ И ВРЪЗКАТА му с други оператори за заявки	72
8	ОЦЕНЯВАНЕ В ИЗВЛИЧАНЕТО НА ИНФОРМАЦИЯ	73
8.1	ОЦЕНЯВАНЕ НА СИСТЕМИТЕ ЗА ИЗВЛИЧАНЕ НА ИНФОРМАЦИЯ	73
8.2	СТАНДАРТНИ ТЕСТОВИ НАБОРИ	74
8.3	ОЦЕНЯВАНЕ НА НЕКАТЕГОРИЗИРАНИ МНОЖЕСТВА ОТ ВЪРНATИ РЕЗУЛТАТИ	75
8.4	ОЦЕНЯВАНЕ НА КАТЕГОРИЗИРАНИ ВЪРНATИ РЕЗУЛТАТИ	78
8.5	ИЗМЕРВАНЕ НА ПОТРЕБИТЕЛСКАТА УДОВЛЕТВОРЕНОСТ	81
8.6	РЕЗЮМЕ НА ИЗВЛЕЧЕНИЯ ДОКУМЕНТ (SUMMARY)	83
9	ОБРАТНА ВРЪЗКА И РАЗШИРЯВАНЕ НА ЗАЯВКАТА	84
9.1	ОБРАТНАТА ВРЪЗКА И ПСЕВДО-ОБРАТНАТА ВРЪЗКА	84
9.1.1	<i>Алгоритъмът на Rocchio за Обратната връзка</i>	85
9.1.2	<i>Вероятностна обратна връзка</i>	89
9.1.3	<i>Кога обратната връзка с приложимост работи?</i>	90
9.1.4	<i>Обратна връзка с приложимост в мрежата</i>	91
9.1.5	<i>Оценка на стратегиите за обратна връзка</i>	92
9.1.6	<i>Псевдо-обратна връзка</i>	93
9.1.7	<i>Косвена обратна връзка</i>	93
9.1.8	<i>Резюме</i>	94
9.2	ГЛОБАЛНИ МЕТОДИ ЗА ПРЕФОРМУЛИРАНЕ НА ЗАПИТВАНИЯТА	95
9.2.1	<i>Лексически способи за преформулиране на заявката</i>	95
9.2.2	<i>Разширяване на заявката</i>	95
9.2.3	<i>Автоматично генериране на речник</i>	97
9.3	СПРАВКА И ПО-ЗАДЪЛБОЧЕНО ЧЕТИВО	98

10 XML ИЗВЛИЧАНЕ НА ИНФОРМАЦИЯ.....	99
10.1 Базови XML понятия	101
10.2 Предизвикателства в XML извлечането	104
10.3 Векторен пространствен модел за XML извлечане на информация.....	108
10.4 Оценка на извлечането от XML.....	111
11 ВЕРОЯТНОСТНО ИЗВЛИЧАНЕ НА ИНФОРМАЦИЯ	116
11.1 Преговор на основната теория на вероятностите	116
11.2 Принцип за вероятностна наредба	117
11.2.1 Случаят с 1/0 загуба	117
11.2.2 PRP с цени за извлечане	118
11.3 Двоичен модел на независимост	118
11.3.1 Получаване на подреждаща функция за термините на търсенето	119
11.3.2 Изчисляване на вероятности.....	121
11.3.3 Вероятностни оценки в практиката	122
11.3.4 Вероятностни подходи за релевантността на обратната връзка.....	123
11.4 Оценяване и някои разширения	125
11.4.1 Оценяване на вероятностен модел.....	125
11.4.2 Дървовидна структура на зависимост между термините	125
11.4.3 Okapi BM25: недвоичен модел.....	126
11.4.4 Бейсови подходи на мрежата за ИИ	127
12 ЕЗИКОВИ МОДЕЛИ ЗА ИЗВЛИЧАНЕ НА ИНФОРМАЦИЯ	128
12.1 Видове езикови модели	129
12.2 Полиномно разпределение.....	130
12.3 Използване на модела за вероятност на заявките.....	130
13 КЛАСИФИКАЦИЯ НА ТЕКСТ И НАИВЕН БЕЙСОВ ПОДХОД	137
13.1 Проблемът класификация на текст.....	139
13.2 Текстов класификатор по Наивно-Бейсов подход	141
13.2.1 Връзка с полиномния универсално граматиченезиков модел	145
13.3 Модел на Бернули.....	146
13.4 Свойства на НБ	147
13.4.1 Разновидност на мултиномиалния модел.....	152
13.5 Подбор на Характеристиките	152
13.5.1 Обща информация (OI)	152
13.5.2 χ^2 Подбор на Характеристиките.....	153
13.5.3 Честотно-ориентиран Подбор на Характеристиките.....	154
13.5.4 Подбор на Характеристиките за мулти-класификатори.....	154
13.5.5 Сравнение на Методите за Подбор на Характеристиките.....	154
13.6 Оценка на текстовата класификация	155
14 КЛАСИФИКАЦИЯ С ВЕКТОРНО ПРОСТРАНСТВО	156
14.1 Хипотеза за съседство:.....	156
14.2 Представяния на документи и мерки за свързаност във векторни пространства	157
14.3 Класифициране по Rocccnio	158
14.4 k-ти най-близък съсед (kNN)	161
14.4.1 Времева сложност и оптимизация на kNN	162
14.5 Линейни и нелинейни класификатори.....	162
14.6 Класификация в повече от два класа.....	165
14.7 Размяната „отклонение – разлика“ (THE BIAS – VARIANCE TRADEOFF)	167
15 МЕТОД НА ОПОРНИТЕ ВЕКТОРИ И МАШИННО САМООБУЧЕНИЕ ВЪРХУ ДОКУМЕНТИ	172
15.1 SVM: Случай на линейното разделяне.....	172
15.2 Разширения на SVM модела	174
15.2.1 Класификация.....	174

15.2.2	<i>Многокласови SVMs</i>	175
15.2.3	<i>Нелинейни SVMs</i>	176
15.2.4	<i>Експериментални резултати</i>	177
15.3	ПРОБЛЕМИ ПРИ КЛАСИФИКАЦИЯТА НА ТЕКСТОВИ ДОКУМЕНТИ.....	177
15.4	МАШИННО САМООБУЧЕНИЕ И ИЗВЛИЧАНЕ НА ИНФОРМАЦИЯ	179
16	ПЛОСКО КЛЪСТЕРИРАНЕ.....	182
16.1	Клъстериране в извличането на информация	183
16.2	Определяне на задачата	186
16.3	Кардиналност - броят на клъстерите.....	187
16.4	Оценяване на клъстерирането.....	187
16.5	K-Means (K-средни)	190
16.5.1	<i>Клъстерирана кардиналност при алгоритъма K-средни</i>	195
16.6	Клъстериране базирано на модели.....	197
17	ЙЕРАРХИЧНО КЛЪСТЕРИРАНЕ.....	202
17.1	1. Йерархично агломеративно клъстериране	202
17.2	2. Клъстериране с единично свързване и пълно свързване	204
17.2.1	2.1. Сложност на ЙАК	205
17.3	3. Агломеративно клъстериране чрез осреднени подобия	207
17.4	4. Центроидно клъстериране.....	208
17.5	5. Оптималност на ЙАК	209
17.6	6. Разделящо клъстериране	211
17.7	7. Етикиране на клъстери	212
18	РАЗЛАГАНЕ НА МАТРИЦИ И ЛАТЕНТНО СЕМАНТИЧНО ИНДЕКСИРАНЕ	213
19	ОСНОВИ НА ТЪРСЕНЕТО В УЕБ	218
19.1	Произход и история	218
19.2	Уеб характеристики	219
19.3	Уеб граф	220
19.4	СПАМ	220
19.5	Рекалмиранието като икономически модел	221
19.6	THE SEARCH USER EXPERIENCE. ПОТРЕБИТЕЛСКО ТЪРСЕНЕ	222
19.7	ПОТРЕБИТЕЛСКИ НУЖДИ (USER QUERY NEEDS).....	223
19.8	ИНДЕКСЕН РАЗМЕР И ОЦЕНЯВАНЕ	223
19.9	Близки дубликати и SHINGLING	225
19.10	РЕЧНИК:	226
20	ОБХОЖДАНЕ И ИНДЕКСИРАНЕ НА МРЕЖАТА	227
20.1	Уеб роботи	227
20.1.1	<i>Свойства, които уеб роботите трябва да притежават</i>	227
20.1.2	<i>Свойства, които уеб роботите е препоръчително да притежават</i>	227
20.2	Обхождане	228
20.2.1	<i>Архитектура на уеб робота</i>	229
20.2.2	<i>Разпределение работата на робота</i>	231
20.2.3	<i>Извличане на IP адреса, който седи зад даден домейн</i>	232
20.2.4	<i>Граница от адреси (the URL frontier)</i>	233
20.3	Разпределение на индексите	236
20.4	Свързващи сървъри	237
20.5	БИБЛИОГРАФИЯ.....	240
21	АНАЛИЗ НА ВРЪЗКИТЕ	241
21.1	Уеб мрежата като граф.....	241
21.1.1	<i>Текстът на хипервръзката и уеб графът</i>	242
21.2	РАНКИРАНЕ НА СТРАНИЦАТА (PAGE RANK) ()	243
21.2.1	<i>Вериги на Марков</i>	244

21.2.2	Пресмятане на PageRank (оценяване на страницата).....	246
21.2.3	Тематично оценяване на страница (Topic-specific PageRank).....	248
21.3	Хъбове и авторитети (Hubs and Authorities).....	249
21.3.1	Изборът на подмножество на уеб	251
22	АНАЛИЗ НА МНЕНИЯ	254
22.1	УВОД.....	254
22.2	ПРОБЛЕМИ В АНАЛИЗА НА ЧУВСТВА (SENTIMENT ANALYSIS)	256
22.3	Класификация на мнения и субективностна класификация	257
22.3.1	Класификация на мнения на ниво документи	258
22.3.2	Базиран на характеристики анализ на мнения	259
22.4	SENTIMENT ANALYSIS НА СРАВНИТЕЛНИ ИЗРЕЧЕНИЯ	260
22.4.1	Дефиниция на проблема	260
22.4.2	Тип на сравнителните релации	260
22.4.3	Извличане на обекти и характеристики на обекти в сравнителни изречения	261
22.5	Търсене и извличане на мнения	262
23	СЪЗДАВАНЕ НА КОМПЮТЪРНИ РЕЧНИЦИ.....	264
23.1	Увод.....	264
23.2	КРАТКО РАЗВИТИЕ НА РЕЧНИЦИТЕ ПРЕЗ ГОДИНТЕ. Автоматизиране на лексикографските практики.....	264
23.3	Взаимодействието корпус – речник	265
23.4	Изготвяне на електронен речник с XML.....	266
23.5	Използване на корпуси. Търсене в българския Браун корпус чрез регулярни изрази	269
23.6	ЗАКЛЮЧЕНИЕ	271
23.7	Използвана литература:	271
24	РЕЧНИК: ПРЕВОД НА ТЕРМИНИТЕ.....	272

1 Булево извлечане

Значението на термина “извлечане на информация” е много обширно. Форма на извлечане на информация може да бъде и изваждането на кредитната ви карта от портфейла, за да видите номера ѝ. В академичните среди извлечане на информация може да бъде дефинирано по следния начин:

Извличане на информация е намирането(откриването) на материал (обикновено документи) от неструктурирани данни(обикновено текст) в големи колекции, който задоволява информационна нужда.

Дефинирана по този начин, “извлечане на информация” е била дейност, в която само малко на брой хора са били въвлечени: библиотекари и други подобни професионалисти, които се занимават с търсение. Сега светът е променен и стотици милиони хора са въвлечени в извлечането на информация ежедневно, когато използват уеб търсачки. Извлечането на информация бързо се превръща в доминантна форма на информационен достъп, взимайки преднина пред традиционното търсение в бази от данни (ситуацията в която чиновникът ви казва „Съжалявам, мога да потърся вашата поръчка само ако ми дадете нейния номер“)

Извличането на информация може също така да покрие други видове проблеми с данните и информацията, от тези специфицирани в горната дефиниция. Терминът „неструктурирани данни“ се отнася за данни, които нямат ясна, семантично явна, разбираема за компютър структура. Той е противоположност на термина структурирани данни, най-типичният пример за които са релационните бази данни, които компаниите обикновено използват, за да съхраняват информация за продуктовия си инвентар и своя персонал. Всъщност, почти няма данни, които да са „неструктурирани“. Това определено важи за всички текстови данни, ако вземем предвид латентната лингвистична структура на естествените езици. Но дори и да приемем, че желаната структура е явната, повечето текст има структура като заглавия, параграфи, бележки, които обикновено се представят в документа изрично със специфична маркировка. Извлечането на информация също се използва да улесни „наполовина структурирано“ търсение, като например намиране на документ, в който заглавието съдържа Java, а в съдържанието има линкове и връзки.

Областта на извлечане на информация също така покрива и подпомагане на потребителите в търсения или филтрирането на списъци документи, или по-нататъшната обработка на множество извлечени документи. Ако предположим, че имаме множество документи, тогава задачата на кълстеризацията е да направи подходящо групиране на документите, въз основа на тяхното съдържание. Това наподобява подреждането на книги на лавица, според тяхната тематика. Като имаме предвид определено множество от теми, установените информационни нужди или други категории, задача на класификацията е да определи към кои класове принадлежи всеки документ. Често срещан подход при класификацията е първо ръчно да се класифицират определен брой документи и след това новите документи да се класифицират автоматично, надявайки се че системата е в състояние да направи това сама, въз основа на вече класифицираните документи.

Системите за извлечане на информация могат да бъдат разграничени и по скалата, по която работят и обикновено се разделят на следните три вида. При търсение в *интернет пространството*, системата трябва да осигури търсение в милиарди документи, съхранявани на милиони компютри. Често срещани проблеми са нуждата да се съберат документи за индексиране, възможността да се изградят системи, които да работят ефективно в този огромен мащаб и да могат да се справят с специални аспекти на уеб пространството.

На всички тези проблеми е обърнато внимание в глави 19-21. В другата крайност е извличането на *лична информация*. През последните няколко години, потребителските операционни системи имат вграден модул за извлечение на информация (например Apple's Mac OS X Spotlight или Windows Vista's Instant Search). E-mail програмите обикновено осигуряват не само търсене, но и класификация на текст: най-малкото, което осигуряват е спам филтър. И обикновено също така осигуряват ръчни или автоматични средства за класификация на писма, така че писмото да бъде поставено директно в конкретна папка. Характерни проблеми тук включват обработката на голяма гама различни типове документи съхранявани на персоналния компютър и поддържането на системата за търсене достатъчно олекотена по отношение на стартиране, обработка и използване на дисковото пространство, така че да може да работи на определена машина без да създава пречки на нейния притежател. Между тези два проблема, стои търсенето в специфични области, *данни на компании и институции*, където информацията може да бъде извлечена за колекция от вътрешни за компанията документи, бази данни за патенти или статии за биохимия. В този случай, документите обикновено се съхраняват в централизирана файлова система и една или няколко специално определени за това машини, ще осигурят търсенето в колекцията от документи. Тази книга съдържа похвати за целия този спектър от нужди, но отразяването на някои аспекти на паралелното и разпределено търсене в уеб-мащабни търсещи системи е сравнително съкратено, поради относително малкия брой публикувана литература за детайлите на такива системи. За един софтуерен инженер е по-вероятно да попадне на случай на личното, персонално търсене или търсенето в условия на компания, отколкото на уеб търсене.

В тази глава започваме с един много прост пример на проблема за извлечение на информация и представяме идеята за термин-документ матрицата (раздел 1.1) и структурата обърнатия индекс (раздел 1.2). След това ще разгледаме модела за булево извлечение и как се обработват булеви заявки (раздел 1.3 и 1.4).

1.1 Пример на проблема за извлечение на информация

Дебела книга, която много хора притежават е „Шекспир-събрани съчинения“. Да предположим, че искате да разберете в коя писка на Шекспир се съдържат думите Brutus и Cesar и не се съдържа Calpurnia. Един от начините да направите това е да започнете да четете от началото и да прочетете целия текст, като отбелязвате за всяка писка дали съдържа Brutus и Cesar и да я изключвате от списъка ако съдържа Calpurnia. Най-простата форма на извлечение на тази информация за компютър е да направи линейно сканиране на документите. Този процес се нарича „grepping“, наречен на unix командата „grep“, която изпълнява този процес. „Grepping-тът“ на текст може да бъде много ефективен процес, имайки предвид скоростта на съвременните компютри и често позволява използването на маска модел за съвпадение чрез използването на регулярни изрази. Със съвременните компютри, за елементарна заявка към неголяма колекция (размерът на „Шекспир-събрани съчинения“ е малко под един милион думи), наистина не се нуждаете от нищо повече.

Но за много цели се нуждаем от:

1. Бърза обработка на големи колекции от документи. Количество на онлайн данните се увеличава толкова бързо колкото и скоростта на компютрите и сега ние бихме искали да можем да търсим колекции от порядъка на билиони и трилиони думи.
2. По-гъвкави операции по съвпадение. Например, не е практично да се извърши заявката „Romans NEAR countrymen“ с grep, където NEAR, може да бъде дефинирано както „в рамките на 5 думи“ така и „в рамките на същото изречение.“

3. Ранкирано извличане: в много случаи искаме най-добрият отговор на информационната нужда измежду много документи, които съдържат определена дума.

Начинът, по който може да се избегне линейното търсена на текст за всяка заявка е предварително да се индексират документите. Нека продължим да работим със съчиненията на Шекспир и да използваме този пример, за да представим основите на булевия модел за извличане на информация. Да предположим, че за всеки документ(в случая писците на Шекспир) записваме дали съдържа конкретна дума от общо всички думи, които Шекспир използва (Шекспир е използвал около 32000 различни думи) Резултатът е *бинарна термин-документ матрица* като на фиг. 1.1. Термините са индексираните единици (обсъдени са и в глава 2.2); те обикновено са думи и за момента можем да мислим за тях като за думи, но в литературата по Извличането на Информация се използва понятието термини, защото например „I-9“ или „Hong Kong“ обикновено не се възприемат като думи. Сега в зависимост от това дали ще разглеждаме колоните или редовете на матрицата, можем да направим вектор за всеки термин, който показва документите в които този термин се среща, или вектор за всеки документ, който показва термините, които се срещат в него.

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

Фиг 1.1 Термин-документ матрица, Елемент на матрицата (t,d) има стойност 1, ако писцата в колона d, съдържа думата в ред t и 0 в противен случай.

За да се върне резултат от заявката Brutus AND Caesar AND NOT Calpurnia, взимаме векторите на Brutus, Caesar и Calpurnia, като правим отрицание на последния и след това извършваме операцията побитово „AND“

$$110100 \text{ AND } 110111 \text{ AND } 101111 = 100100$$

Резултатът от тази заявка са „Антоний и Клеопатра“ и „Хамлет“ (фигура 1.2).

Моделът „*Булево Извлечане*“ е модел за извличане на информация, в който можем да представим всяка заявка, под формата на Буев израз на термини, което означава термините да са комбинирани с оператори „AND“, „OR“, „NOT“. Моделът разглежда всеки документ, като съвкупност от думи.

Нека сега да разгледаме по-реалистичен сценарий използвайки възможността да въведем някои терминологии и означения. Да предположим, че имаме $N=1$ милион документа. Под *документ* разбираме всякакви единици, за които сме решили да направим система за извличане. Това могат да бъдат самостоятелни бележки или глави на книги(раздел 2.1.2) Ще наричаме групата от документи, върху които извършваме извлечането *колекция от документи*. Понякога се нарича и *корпус*. Да предположим, че всеки документ има дължина около 1000 думи (2-3 страници)

Antony and Cleopatra, Act III, Scene ii

Agrippa [Aside to Domitius Enobarbus]: Why, Enobarbus,

When Antony found Julius Caesar dead,

He cried almost to roaring; and he wept
When at Philippi he found Brutus slain.

Hamlet, Act III, Scene ii

Lord Polonius: I did enact Julius Caesar: I was killed i' the
Capitol; Brutus killed me.

Фиг 1.2 Резултат от Шекспир за заявката Brutus AND Caesar AND NOTCalpurnia.

Ако предположим, че средната дължина на думата е 6 байта (включвайки интервали и пунктуация), тогава размерът на колекция от документи е около 6 гигабайта. Обикновено може да има около $M=500,000$ различни термина в тези документи. Няма нищо специално в числата, които сме избрали и те могат да варират, но все пак ни дават идея за размерите на проблемите, с които трябва да се справим. Ще дискутираме и моделираме тези предположения за размера в раздел 5.1.

Нашата цел е да се разработи система, която да се справи със задачата за извлечане на *специфична информация*. Това е най-стандартната задача в извлечането на информация. При нея целта на системата е да осигури документи, от колекцията, които отговарят на произволна информационна нужда на потребителя, подадена на системата под форма на заявка. *Информационната нужда* е тема, за която потребителят би желал да знае повече. Тя се различава от *заявката*, която представлява, това което потребителят се опитва да предаде на компютъра в опита си да комуникира информационната нужда. Един документ е *релевантен*, ако е такъв, че да се възприема от потребителят като съдържащ информация, отговаряща на неговата информационна нужда. Нашият пример по-горе беше по-скоро изкуствен, имайки предвид, че информационната нужда беше дефинирана със специфични думи. Обикновено потребителят е заинтересован от тема като например „течове в тръбопровод“ и би искал да намери релевантни документи, независимо от това дали те съдържат тези думи или изразяват връзката с други думи като „пробиване на газопровод“. За да се оцени *ефективността* на системите за извлечане на информация (т.е качеството на върнатият от търсенето резултат), потребителят обикновено би искал да знае две ключови статистически метрики:

Precision: Каква част от върнатите резултати удовлетворяват информационната нужда?

Recall: Каква част от документите, които удовлетворяват информационната нужда са върнати от системата?

Подробна дискусия за метрики за релевантни оценки, включително precision и recall се разглежда в глава 8.

Сега не можем да построим термин –документ матрица по лесен начин. $500K \times 1M$ матрица има половин трилион нули и единици, което е твърде много за да се побере в компютърната памет. Но ключовото наблюдение е, че матрицата е твърде рядка, т.е има няколко ненулеви записи. Понеже всеки документ има дължина 1000 думи, матрицата има не повече от един билион единици, т.е минимум 99,8% от клетките са нули. Много по-добър подход е да се запишат само събитията, които са се случили, т.е позициите с единици. Тази идея е в центъра на първата основна концепция в извлечането на информация, *обърнатия индекс*. Името всъщност е многословно: индексът винаги кореспондира с термини към частите на документите, където те се срещат. Въпреки всичко обърнатия индекс (понякога се използва и понятието *обърнат файл*), е станал стандартен термин в областта на извлечането на информация. Основната идея на обърнатия индекс е показана на фиг. 1.3. Пази се *речник* на термините (понякога се нарича и *лексикон*). За всеки термин имаме списък, който показва в кои документи, този термин се среща. всяка точка в списъка, който показва, че даден термин се среща в документ, наричаме *постинг*. Списъкът се нарича списък с постинги. Речникът на фиг. 1.3 е подреден по азбучен ред и всеки лист от постинги е сортиран по номер на документа. Ще

видим защо това е полезно в раздел 1.3, а по-късно ще разгледаме и някои алтернативи (раздел 7.1.5)

Brutus → 1 2 4 11 31 45 173 174

Caesar → 1 2 4 5 6 16 57 132 ..

Calpurnia → 2 31 54 101



Dictionary Postings

Фиг 1.3 Двете части на обърнатия индекс. Обикновено речникът се запазва в паметта, с показател към всеки лист от постинги, който се съхранява на диска.

1.2 Изграждане на обърнат индекс

За да спечелим от индексиране по време на извличане на информацията, трябва да построим индекса предварително. Основните стъпки за това са:

1. Събиране на документите, които трябва да бъдат индексирани
Friends, Romans, countrymen. So let it be with Caesar ...
2. Токенизация на текста, превръщайки всеки документ в списък от токени.
Friends Romans countrymen So ...
3. Лингвистична обработка, в резултат на която се изготвя списък с нормализирани токени, които се индексират термини friend roman countryman so ...
4. Индексиране на документите, в които всеки термин се среща, създавайки обърнат индекс, който се състои от речник и постинги.

Ще дефинираме и дискутираме стъпки 1-3 в раздел 2.2. Догогава може да мислите за токени и нормализирани токени като за думи. Тук предполагаме, че първите три стъпки са били вече направени и разглеждаме построяването на обърнат индекс чрез базирано на сортиране индексиране.

В рамките на колекцията считаме, че всеки един документ има уникален сериен номер, известен като идентификатор на документа (*docID*). По време на построяването на индекса, можем да назначим последователни числа на всеки един документ, когато го срещнем за първи път. Входните данни за индексирането са списък от нормализирани токени за всеки документ, за които може да мислим като двойка термин и *docID*, както е показано на фиг.1.4. Основната стъпка в индексирането е *сортирането* на този списък така, че термините да са подредени по азбучен ред. Ако един термин се среща много пъти в един документ, тогава тези срещания се сливат. Отделните срещания на един и същ термин се групират и резултатът се разделя в *речник* и *постинги*, както е показано в дясната колона на фиг1.4. Тъй като обикновено един термин се среща в няколко документа, тази организация на данните намалява изискванията за съхранение на индекса. Речникът също така записва някои статистики, като например брой документи, които съдържат даден термин . Тази информация не е от съществено значение за булевите търсачки, но позволява да се подобри ефективността на търсачката по време на изпълнение на заявката и е статистическа метрика, която по-късно се използва в много модели за извличане на информация.

Doc 1

I did enact Julius Caesar. I was killed
i' the Capitol; Brutus killed me.

term	docID
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Doc 2

So let it be with Caesar. The noble Brutus
hath told you Caesar was ambitious:

term	doc.	freq.	→	postings lists
ambitious	2	1	→	2
be	1		→	2
brutus	1	2	→	1 → 2
capitol	1	1	→	1
caesar	2	2	→	1 → 2
did	1	1	→	1
enact	1	1	→	1
hath	1	1	→	2
I	1	1	→	1
i'	1	1	→	1
it	1	1	→	2
julius	1	1	→	1
killed	1	1	→	1
let	1	1	→	2
me	1	1	→	1
noble	2	1	→	2
so	2	2	→	2
the	1	2	→	1 → 2
told	2	1	→	2
you	2	1	→	2
was	1	2	→	1 → 2
with	2	1	→	2

Фиг 1.4 Построяване на индекс чрез сориране и групиране. Редицата от термини във всеки документ, отбелязани чрез техните docID е сортирана по алфавитен ред. Копията на един и същ термин след това са групирани по дума и по docID. След това термините и docID са разделени. В речника се записват термите, за всеки от които има указател към списъка с постинги. Обикновено съхранява и друга обобщена информация, като например честотата на документа за всеки термин. Използваме тази информация, за да подобрим времето за обработка на заявката, а също така както и по-късно ще видим за претегляне в други модели за извлечение на информация. Всеки списък от постинги съхранява списък от документите, в които се среща термина, а може и да съдържа друга информация като например честотата на термина (честотата на всеки термин във всеки документ) или позиция на термина във всеки документ.

Постингите са вторично сортирани по docID. Това осигурява основата за ефективна обработка за заявката. Структурата обрънат индекс е най-ефективната структура при специфичното търсене текст.

В резултатния индекс, „плащаме“ за съхранение на речника и на списъка от постинги. Последният е много по-голям, но речникът обикновено се съхранява в паметта, докато списъците от постинги обикновено се съхраняват на диска, така че големината на всеки един от

тях е важна и в глава 5 ще разгледаме как всеки от тях може да бъде оптимизиран що се отнася до съхранение и достъп. Каква структура от данни трябва да бъде използвана за списъка от постинги? Масив с фиксирана дължина не би бил подходящ, тъй като някои думи се срещат в много документи, а други само я някои. За списъци от постинги, които се съхраняват в паметта, две добри алтернативи са свързани списъци или масиви с променлива дължина. Свързаните списъци позволяват „евтино“ вмъкване на документи в списъка от постинги (следвайки обновяване, като например когато претърсваме уеб пространството за обновени документи) и естествено разширяване към по-напреднали стратегии за индексиране като „skip list“ (раздел 2.3), които изискват допълнителни указатели. Масивите с променлива дължина печелят изискванията за място, защото тяхното използване от оперативната памет увеличава скоростта на съвременните процесори. Допълнителните указатели, могат да бъдат кодирани в списъците с отмествания. Ако промените се случват относително рядко, масивите с променлива дължина ще бъдат по-компактни и бързи за обхождане. Можем да използваме и хибридна схема със свързан списък от масиви с фиксирана дължина за всеки термин. Когато списъците от постинги се съхраняват на диск, те се съхраняват като съседни (като на фиг 1.3), така че да се минимизира размерът на списъци от постинги и броят на опитите на диска да прочете списък от постинги в паметта.

Brutus → [1] → [2] → [4] → [11] → [31] → [45] → [173] → [174]

Calpurnia → [2] → [31] → [54] → [101]

Intersection ⇒ [2] → [31]

Фиг 1.5 Намиране на сечението на списъка с постинги на Brutus и Calpurnia от Фиг 1.3

1.3 Обработка на булеви заявки

Как обработваме заявки, използвайки обърнат индекс и модела за Булеово извлечане?

Да вземем за пример обработка на заявката

Brutus AND Calpurnia

над обърнатия индекс, показан на Фиг1.3

1. Намираме Brutus в речника
2. Извличаме неговите постинги
3. Намираме Calpurnia в речника
4. Извличаме неговите постинги
5. Правим сечение на двата списъка от постинги, както е показано на Фиг1.5

Операцията „сечение“ е от огромно значение: трябва ефективно да направим сечение на списъците от постинги, за да може бързо да намерим документи, които съдържат и двата термина (понякога тази операция се нарича *сливане* на списъци от постинги, тук ние сливаме списъците с логическо „AND“)

Има лесен и ефективен метод за извършване на сечение на списъци от постинги, използвайки *метод на сливането* (фиг 1.6):

Сечение (p1, p2)

```
1 answer ← h i 2 while p1 6= NIL and p2 6= NIL
```

```

3 do if docID(p1) = docID(p2)
4 then ADD(answer, docID(p1))
5 p1  $\leftarrow$  next(p1)
6 p2  $\leftarrow$  next(p2)
7 else if docID(p1) < docID(p2)
8 then p1  $\leftarrow$  next(p1)
9 else p2  $\leftarrow$  next(p2)
10 return answer

```

Фиг 1.6 Алгоритъм за извършване на сечение на два списъка от постинги p1 и p2

Обработваме указатели в двата списъка и обхождаме двата постинга едновременно, за общо всички записи в постинга. На всяка стъпка сравняваме docID, посочено от двата указателя. Ако docID съвпада, тогава го слагаме в резултата, и увеличаваме указателите. Ако не съвпадат, увеличаваме указателя, който сочи по-малкото docID. Ако списъците имат дължина x и y, сечението прави $O(x + y)$ операции. Сложността на заявката е $Q(N)$, където N е броят на документите в колекцията. Нашите методи за индексиране ни дават константа, не разлика в Q сложността, сравнено с линейно сканиране, но в практиката константата е огромна. За да се използва този алгоритъм е от огромно значение, постингите да бъдат сортирани по еднакъв начин. Лесен начин за постигане на това е използвайки сортиране по docID.

Можем да разширим операцията „сечение“, за да обработим по-сложни заявки, като например:

(Brutus OR Caesar) AND NOT Calpurnia

Оптимизация на една заявка е процесът по селектиране как да организираме работата по отговорянето на заявката, така че системата да трябва да извърши най-малко количество работа. Важен елемент за това при Булевите заявки е последователността, в която списъците се достъпват. Коя е най-добрата последователност за обработка на заявката? Да вземем за пример следната заявка:

Brutus AND Caesar AND Calpurnia

Трябва да намерим постингите за вски един от термините и после да направим логическо „AND“. Евристичният метод е да обработваме термините, подредени по увеличаваща се честота на документите.

Сечение ($ht1, \dots, tni$)

```

1 terms  $\leftarrow$  SORTBYINCRESSINGFREQUENCY( $ht1, \dots, tni$ )
2 result  $\leftarrow$  postings(first(terms))
3 terms  $\leftarrow$  rest(terms)
4 while terms  $\neq$  NIL and result  $\neq$  NIL
5 do result  $\leftarrow$  INTERSECT(result, postings(first(terms)))
6 terms  $\leftarrow$  rest(terms)
7 return result

```

Фиг 1.7 Алгоритъм на заявки със сечение

Ако започнем да правим сечение на двета най-малки списъка, тогава всички междинни резултати , не би трябвало да са по-големи от най-малкия списък, и поради това ние ще направим най-малкото количество работа. За списъците от фиг.1.3, изпълняваме заявката по следния начин:

(Calpurnia AND Brutus) AND Caesar

Това е първото оправдание за съхраняване на честотата на термините в речника: позволява ни да вземем решение за подредбата, въз основа на данни в паметта, преди да сме достъпили списъци от постинги.

Да вземем за пример по-генерални заявки като:

(madding OR crowd) AND (ignoble OR strife) AND (killed OR slain)

Както досега, ще вземем честотата на всички термини и след това можем да прогнозираме размерът на всеки „OR“ , по сумата на честотата на неговите дизюнкции. След това можем да изпълним заявката в нарастващ ред по големината на всеки дизюнктивен терм.

За случаини Булеви заявки, трябва да оценим и временно да съхраним отговорите за междинни изрази в един сложен израз. В много случаи заявката е чисто конюнктивна. В много от случаите обаче, дали поради естеството на езика за заявки или просто защото това е най-често срещания тип заявки, които потребителя използва, заявката е чиста конюнкция. В този случай вместо да използваме сливането на списъци като функция с два входни параметъра и различен изход, по –ефективно е да направим сечение на всеки извлечен постинг, с текущия междинен резултат в паметта, като инициализираме междинния резултат използвайки списъка на най-рядко срещания термин. Този алгоритъм е показан на фиг 1.7. Операцията “сечение“ е несиметрична: междинният резултат е в паметта, докато списъка с който правим сечение се чете от диска. Сечението на постинги може да бъде направено и по алгоритъма от фиг 1.6, но когато разликата в дължината на списъците е много голяма се появяват опции за използване на алтернативи. Сечението може да бъде направено чрез модификация и маркиране на невалидни елементи в междинния резултат. Също така сечението може да бъде направено като последователност от бинарно търсене в дълъг списък за всеки постинг в междинния резултат. Друга възможност е дългия списък да се съхранява в хештаблица. Но такива алтернативни техники са сложни за комбиниране, още повече че стандартното сечение на списъците остава необходимо когато двета термина на заявката са много близки.

1.4 Разширеният булев модел срещу ранкирано извлечане

Булевият модел за извлечане се различава от моделите за *ранкирано извлечане*, като векторно пространствения модел (раздел 6.3), в който потребителите предимно използват *текстови заявки*, т.е пишат една или две думи, вместо да използват определен език, с оператори, за построяване на заявката. Въпреки десетки години на академично проучване за предимствата на ранкираното извлечане, повечето системи използваха модела за Булево извлечане до началото на 90-те години. (приблизително до появата на World Wide Web). Тези системи използваха не само стандартните булеви операции(AND, OR, NOT), които ви представихме досега. Точните булеви изрази над термини, с незакръглен резултати са твърде ограничени за да отговорят на информационните нужди на потребителите и това въвежда използването на разширени булеви модели, които въвеждат допълнителни оператори като близост на термини. *Операторите за близост* се използват, за да се търсят термини, които се намират на близко разстояние един от друг в даден документ. Близостта, може да бъде измерена, чрез

ограничаване на позволеният брой засегнати думи, или да бъде структурна единица като изречение или параграф.

В тази глава разгледахме структурата и построяването на обрънатия индекс, представяйки речник и списък от постинги. Представихме Булевият модел за извличане и разгледахме как да направим ефективно извличане, чрез сливане на линейно време и прости оптимизации на заявките. В глави 2-7 ще разгледаме в детайли по-богати модели на заявки и сортиране на увеличаващ се индексни структури. Тук само споменахме някои допълнителни неща, които бихме искали да правим:

1. Бихме искали по-добре да определим множеството от термини в речника и да направим извличане, което добре да се справя с правописни грешки и лош избор на думи.
2. Понякога е по-добре да се търси за фраза или словосъчетания, които дефинират концепция като например „операционна система“.
3. Булевия модел само отбелязва присъствието или отсъствието на термините, но често ние искаме да имаме други метрики, които да дават повече тежест на документи, съдържащи един термин няколко пъти. За да можем да използваме това, имаме нужда от информация за честотата на термина в списъка от постинги.
4. Булевите заявки извличат множество от документи, отговарящи на заявката, но обикновено ние искаме да можем ефективно да подредим или ранкираме този резултат. Това изисква да се създаде механизъм, който да определя ранка на документа, т.е. доколко този документ отговаря на заявката.

С тези допълнителни идеи, ще имаме възможност да видим повечето от основните технологии за специализирано търсене в неструктуррирана информация. Специфичното търсене в документи, насокро завладя света, давайки мощ не само на уеб търсачките, но и на сайтове за електронна търговия. Въпреки че основните уеб търсачки се различават, повечето от основните проблеми и технологии за индексиране остават едни и същи. Освен това през годините, уеб търсачките добавят поне частична имплементация на някои от най-популярните оператори от разширения Буев модел: търсенето по фрази е особено популярно. Тези опции са много харесвани от професионалисти, но са малко използвани от повечето хора и не са основният фокус на работа при опит да се подобри производителността на уеб търсачките.

2 Речник на термините и списък на публикациите

The term vocabulary and postings lists

В тази тема ще разгледаме как се определят съставните части на документите и как от тях се определят характерни последователности. Ще се запознаем с това, как се избират термините, които характеризират един документ, как тези термини могат да бъдат групирани във фрази и какви са спецификите, проблемите и техните решения в този процес.

2.1 Анализиране на документи и изграждане на последователности

За да бъдат анализирани документите, трябва да бъде извлечена информацията, която те съдържат и която подлежи на анализ. За да стане това документите трябва да бъдат прочетени.

2.1.1 Прочитане на документи

Прочитането на документите става, като всеки течен символ се прочита последователно. След прочитането на всички символи, те се анализира и на базата на тях, могат да бъдат изградени думи, изречения и други части на речта. След като приключи процедурата по прочитане на един документ и преди да бъдат изградени установени частите на речта, които се съдържат в документа, прочетените символи се съхраняват, като последователност от символи.

След като разполагаме с последователност от символи, извлечени от документа, ние можем да я анализираме и да разграничим частите на речта. За целта избираме един или няколко знака, които представляват разделители и разделяме последователността на по-малки части. Всяка от формираните части се явява потенциална част на речта, като например дума или число. Изградените части на речта могат да бъдат използвани самостоятелно и анализирани, с цел да се открият търсените от нас части на речта и да се намерят търсени документи или информация.

2.1.2 Избор на части от документ

В някой случай анализираните документи са много големи. В тези случаи документите могат да бъдат разделяни на по малки части. Това се прави с цел по-лесното намиране на търсена от нас информация.

Когато един документ е много голям, той може да бъде разделен спрямо неговата структура. За да се раздели правилно документа е необходимо да се направи анализ, на това, как той може да бъде разделен, според глави, секции, параграфи и т.н.. След разделянето на документа, всяка негова част може да бъде обработвана самостоятелна и често може да бъде достъпвана, като индивидуален документ, с което се улеснява търсенето и получаването на информация.

Разделянето на документа на прекалено малки под-документи може да повлияе на логическата връзка между тях и носеният от тях смисъл да бъде загубен или променен. За да бе се достигне до загуба на информация, е необходимо да разделянето да бъде направено с норма и по подходящ начин.

2.2 Определяне на речник на термините

В процеса на търсене на информация се използват части на речта, с който се описва търсената информация. За да се намери търсената информация е необходимо предварително да са установени частите на речта, които описахме в предходната точка, в документите, в които търсим. След като частите на речта, които се съдържат в документите, са известни, е необходимо да подберем тези от тях, които имат пряко отношение към документа или тези които достатъчно съмисъл и биха могли да бъдат търсени. Така подбраните части на речта се приемат за термини. Речникът на термините се прави за всеки документ и включва в себе си всички термини, които са включени в документа.

2.2.1 Процес за установяване на термини

За да изградим речника на термините е необходимо термините да бъдат установени или подбрани и да бъдат записани в речника. За да бъдат установени термините, намерените части на речта се анализират и преценява, кои от тях могат да бъдат термини. В процеса на анализ се премахват препинателните, както и някои други знаци. Процеса на установяване на термините е сложен и има различни специфики, някои от които са:

- Един термин може да бъде получен от различни думи, части на речта или най-общо последователности от знаци. Това се получава в различни случаи, примерно когато се премахват точките в различни съкращения, като в „U.S.A.“. В подобни ситуации премахването на различни знаци може да не доведе до промяна на смисъла, но също така може изцяло да промени смисъла на полученият термин. Често премахването на точки, тирета и други знаци не просто променя значението на термина, с което пречи на намирането на търсената от нас информация, но може да превърне термина в обиден, неприличен или расистки, което не е желателно. Въпреки това че премахването на различни знаци може да доведе до множество проблеми, тази техника се прилага. Това се прави, защото уединява множество думи или различни последователности от символи, които имат един и същи смисъл и могат да бъдат приети с един термин, и по този начин се улеснява намирането на търсената от нас информация.
- Последователности от знаци, които се явяват числа, електронни адреси или други специфични данни, се приемат по начина, по който са въведени в документите, без над тях да се извършват допълнителни обработки. Подобни последователности се приемат за специфични, притежаващи специфично значение и като такива могат да бъдат приети, като термини.

2.2.2 Премахване на излишните термини

Множество думи или последователности от символи могат да бъдат приети, като неприемливи за съставянето на термини. Обичайно това са думи, които се използват често, като части от речта, които допринасят за смисъла на един текст, но използвани самостоятелно не носят конкретен смисъл. Подобни части на речта се приемат за неподходящи за търсене и с тях не могат или не трябва да бъдат изграждани термини. За да се премахнат тези части на речта, се

съставя списък наречен „списък на стоп думите”, в който се вписват думи, които не са подходящи за термини.

Списъкът на стоп думите е може да бъде изграден по различни начини, като всеки от тях има своите особености. Някои от параметрите на списъка на стоп думите са неговата големина и начина на подбор на думите, които се вписват в него.

Според големината си списъка на стоп думите може да има различен брой думи записан в него. Проблеми при изграждането на списъка са случаите, в които думите са прекалено много или много малко. Когато думите са много, проверката за съответствие на анализираните думи спрямо думите от списъка се извършва по-бавно, с което се затруднява работата. Когато думите са много малко, често се случва думи, които не носят самостоятелен смисъл, да не бъдат включвани в списъка и по този начин да се приемат за термини, което може да повлияе на търсенето.

Според начина на подбор на думите в списъка на стоп думите, те могат да бъдат изградени на базата на различни документи или на базата на един документ, както и на базата на един или няколко езика.

Когато списъка на стоп думите се изграждат на базата на един документ, той може да съдържа стоп думи, които се съдържат в документа, с което ще се намали неговият размер, но списъка ще трябва да се прави за всеки документ, което ще отнема време и ресурси. Ако списъка се изгражда на базата на различни документи, той ще бъде изграден веднъж и след това ще бъде използван за анализ на различни документи, което е предимство, но списъка може да бъде поголям и да забави анализа на документите.

При съставянето на списъка на стоп думите, той може да бъде изграден на базата на един език или на базата на множество езици, като проблемите, които се получават със сходни с проблемите при изграждането му на базата на един документ или на базата на повече документи, които разглеждахме по-горе.

2.2.3 Нормализиране или уеднаквяване на термините

Процеса на нормализиране или уеднаквяване на термините се извършва, след като е определен речника на термините и са премахнати излишните термини. Този процес се извършва, защото често се случва да съществуват множество термини, които имат еднакво значение но са записани по различен начин. Предполага се, че при търсене на някои от термините със значение сходно с други термини, може би се търси и останалите термини. За да може да се намират всички тези термини и представянето от тях значение, без да е необходимо те да бъдат записани по точния начин, по който те фигурират документа, е необходимо те да бъдат обработени. Различни начини за обработка на термините са представени по-долу:

- Един от начините за уеднаквяване на термини е чрез премахване на символи от тях. Като пример за премахване на символи можем да разгледаме термините „USA” и „U.S.A.”, които често имат едно и също значение. В случая премахването на точките от вторият термин, ще го уеднакви с първия. Премахването на символи се извършва над термините, за да се намали броят им в списъка на термините, и над термините които въвеждаме в процеса на търсене, за да се уеднакват те с тези, които са записани в списъка. Проблеми на този подход са увеличаването на времето за предварителен

анализ на документи и изграждане на списък на термините, както и възможността да бъдат направени грешки в процеса по търсене, заради това, че от термините са премахнати знаци, които в някои случаи могат да имат специфично значение.

- Друг начин за уеднаквяване на термините е чрез използване на един термин в списъка на термините, които да представя различни други термини, които могат да бъдат негови синоними или контекстни синоними. Пример за подобно уеднаквяване може да се наблюдава при използването на термините „кола“ и „автомобил“. Въпреки използването на един термин представлящ няколко термина, този подход не довежда до намаляване на големината на списъка с термини, защото в списъка, трябва да се изградят допълнителни полета за синонимните термини. При използването на този метод е възможно получаването на грешки в резултатите от процеса на търсене, които са породени от това, че един термин е синоним на друг в определен контекст, не е синоним в спрямо контекста, за които се извършва търсенето.
- Премахването на специални знаци е метод за уеднаквяване, който е много сходен с премахването на символи. Но за разлика от премахването на символи, премахването на специални знаци по-скоро цели премахването на знаци, които нямат значение към термина. Такива знаци са ударенията, пунктуационните знаци, които по погрешка са приети за част от термините в процеса на изграждане на термини, както и други знаци специфични за различните езици. Премахването на специални знаци, както и премахването на символи, може да доведе до промяна на термините и да внесе грешки в резултатите от процеса на търсене.
- Уеднаквяване на буквите в термините е друг метод за уеднаквяване на термините. Уеднаквяването на буквите, представлява използването само на малки или само на главни букви при записването на термините. По този начин се избягва неудобството да бъдат пропуснати резултати, което може да се получи поради причината, че търсените думи могат да бъдат записани с малки и главни букви. Този метод може да внесе грешка в резултатите от процеса на търсене, заради възможността да бъде загубена информация, която се е носила от различните видове букви. Това може да се случи, когато записите на различни наименования, записани само с малки или само с главни букви, съвпадат помежду си или съвпадат с други думи или най-общо с други термини. Предимството на този метод е намаляване на големината на списъка с термини.
- Метод, които се използва за уеднаквяване на термините, но е по специфичен от останалите методи, е разпознаването на изрази и по-специално на идиоми. Този метод се използва, за да разпознава групи от думи, които могат да бъдат заменени с термин представлящ тяхното значение. Метода е по специфичен от останалите, защото изисква допълнителна информация за различните езиците и техните особености. Метода забавя процеса на първоначален анализ на документите и изисква допълнително пространство съхранение на данните за различните езици, но позволява повече възможности в процеса на търсене.

- Отчитането на различните езици и спецификите при работа с тях, също е важно, в процеса на уеднаквяване на термините. Примери за това могат да бъдат случаите в които се използват езици в които се пише от дясно на ляво или по друг специфичен начин. При работа с документи в които се използват такива езици, може да се наложи допълнителен анализ на термините.

Освен изброените методи, съществуват и други методи, както и различни разновидности на изброените. Множеството методи за уеднаквяване могат да бъдат ползвани както самостоятелно, така и комбинирано, като с това се цели подобряване на крайният резултат. Не трябва да се забравя, че методите имат, както положителни, така и отрицателни специфики, и трябва да бъдат използвани по подходящ начин.

2.2.4 Отчитане на това, че различни форми на думите могат да имат еднакво значение

В процеса на определяне на речника на термините се извършват различни анализи. Един от анализите, които се извършва е морфологичен анализ. Морфологичният анализ се използва при определянето на термините, с цел откриването на основната част на думата. Благодарение на този анализ множество думи, които се явяват различни форми на една дума, могат да бъдат представени с един термин.

В процеса на анализ на думите, най-често се махат различни техни окончания. Това може да става благодарение на собствената имплементация на известни алгоритми, на имплементацията на собствени алгоритми, по комбинирани начини или с използването на готови продукти. Често анализа на думите се осъществява с готови продукти, а такива съществуват и често могат да се използват свободно.

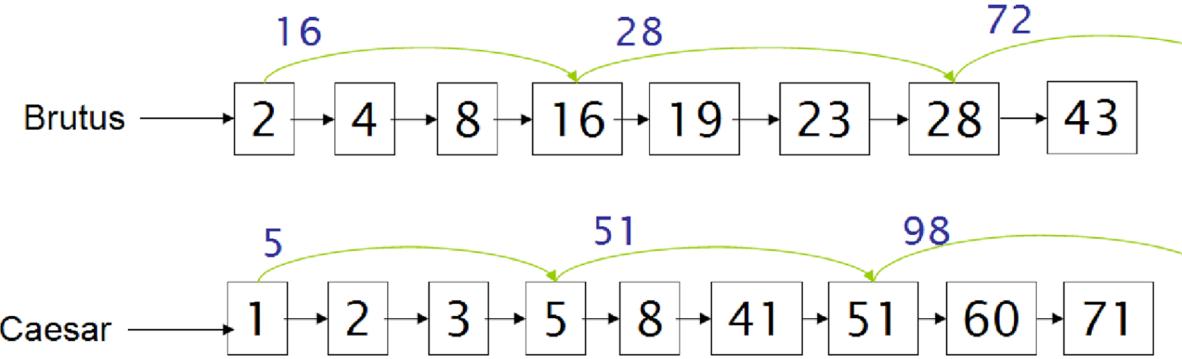
Предимствата от използването на метода са намаляване на големината на списъка с термините и по-бързото намиране на търсени термини в него. Недостатъците на метода са по-бавното първоначално изграждане на списъка и възможността от получаване на грешни резултати в процеса на търсене, които са породени от това, че при търсенето не са използвани точните форми на думите.

2.3 Бързо сливане на постинг списъци чрез прескачащи указатели

(Faster postings list intersection via skip pointers)

В тази глава ще разгледаме разширението на структурите на списъците със срещания и начините, по които можем да ги направим по-ефикасни. Ще разгледаме два списъка, съответно с дължини n и m . Обхождането на двата списъка ще ни отнеме $O(n + m)$ операции. Можем ли да го направим по-бързо?

Единият начин да направим това е да използваме допълнителен лист (skip list), който аргументира списъците със срещания с указатели (индексирани полета), както е показано на фигурата.



Указателите са ефикасни кратки пътища, които ни позволяват да пропуснем тези части от списъците със срещания, които няма да се включат в резултата. Съществуват два въпроса като например къде да поставим тези указатели и как да направим ефикасно сливане с използването на указатели.

Нека да разгледаме фигурата. Представете си че минаваме през списъка, докато не срещнем 8 от всеки лист и го преместваме в резултата. Придвижваме и двата указателя, които ни дават 16 на първия лист и 41 на втория. Най-малкия елемент е элемента 16 от първия лист. Ако искаме да преместим указателя на горния лист, първо ще проверим указателя на допълнителния (skip) лист и ще установим, че 28 също е по малко от 41. В такъв случай ние можем да проследим прескачащия указател и да преместим указателя на горния лист до элемента 28. Така пропускаме минаването през елементите 19 и 23. Съществуват много вариации на сечение на списъци със срещания с използването на пропускащи указатели в зависимост от това, кога точно проверяваме самите указатели. Една версия е показана на следващата фигура. Пропускащите указатели ще са на разположение само за оригиналните списъци със срещания. За междинен резултат в сложна заявка нашия пример винаги ще връща лъжа. Трябва да се отбележи, че пропускащите указатели помагат само за AND заявки, но не и за OR заявки.

INTERSECTWITHSKIPS(p_1, p_2)

```

1  answer ← ⟨ ⟩
2  while  $p_1 \neq \text{NIL}$  and  $p_2 \neq \text{NIL}$ 
3  do if  $\text{docID}(p_1) = \text{docID}(p_2)$ 
4      then ADD(answer,  $\text{docID}(p_1)$ )
5           $p_1 \leftarrow \text{next}(p_1)$ 
6           $p_2 \leftarrow \text{next}(p_2)$ 
7      else if  $\text{docID}(p_1) < \text{docID}(p_2)$ 
8          then if  $\text{hasSkip}(p_1)$  and  $(\text{docID}(\text{skip}(p_1)) \leq \text{docID}(p_2))$ 
9              then while  $\text{hasSkip}(p_1)$  and  $(\text{docID}(\text{skip}(p_1)) \leq \text{docID}(p_2))$ 
10             do  $p_1 \leftarrow \text{skip}(p_1)$ 
11             else  $p_1 \leftarrow \text{next}(p_1)$ 
12         else if  $\text{hasSkip}(p_2)$  and  $(\text{docID}(\text{skip}(p_2)) \leq \text{docID}(p_1))$ 
13             then while  $\text{hasSkip}(p_2)$  and  $(\text{docID}(\text{skip}(p_2)) \leq \text{docID}(p_1))$ 
14             do  $p_2 \leftarrow \text{skip}(p_2)$ 
15             else  $p_2 \leftarrow \text{next}(p_2)$ 
16  return answer

```

Но кога да поставим пропускане? Много пропускания означават по-къси последователности. Но от друга страна също много сравнения с пропускащите указатели и много необходима памет за запаметяването им. По-малко пропускания означава по-малко сравнения, но по-дълги последователности. Трябва да се избере компромисно решение. Прост евристичен метод за поставяне на прескачане, който е бил открит да работи на практика, е за списък със срещания с дължина P да се използват \sqrt{P} равномерно разположени указатели.

Построяването на ефективни прескачачи указатели е лесно ако индекса е относително статичен.

Изборът на оптимално кодиране за обърнатия индекс е задача при проектирането на системния. Преди, процесорите бяха бавни и не използването на високо компресирани техники не бяха оптимални. Сега процесорите са бързи и дисковете са бавни, така че редуцирането на списъците със срещания доминира. Обаче, ако изпълнявате търсеща машина, която пази всичко в паметта, тогава уравнението се променя отново.

2.4 Позиционни срещания и фразови заявки

Много комплексни или технически понятия и много организационни и продуктни имена се състоят от много думи или фрази. Ние бихме желали да имаме възможност да изгълним заявка с “Stanford University”, като го третираме като фраза, така че изречението “The inventor Stanford Ovshinsky never went to university” да не съвпада. Много от търсещите машини поддържат двойните кавички като синтаксис за фрази, които е доказано че е лесно за разбиране от крайния потребител. 10% от учебните заявки са фразови заявки. За да имаме възможност да поддържаме такива заявки, вече не са достатъчни само прости списъци със срещания. Ще предоставим два начина за поддръжка на фразови заявки и техните комбинации. Търсачката не трябва само да предоставя фразови заявки, а да ги имплементира единично.

2.4.1 Индекси на две думи (Biword indexes)

Един подход за търсенето на фрази е да се мисли всяка двойка от последователни термини в документа като фраза. Например текстът “Friends, Romans, Countryman” ще генерира двойните думи:

```
friends romans  
romans countryman
```

В този модел третираме всяка от тези двойни думи като речников термин. По-дългите фрази могат да се раздробят на по-малки термини и след това да се използват като булеви заявки върху всички термини.

STANFORD UNIVERSITY PALO ALTO = “STANFORD UNIVERSITY” AND “UNIVERSITY PALO” AND “PALO ALTO”

Сред възможните заявки, фраза от две съществителни има специален статус. Но свързаните съществителни могат често да се разделят едно от друго с помощта на различни функционални думи във фрази като “the abolition of slavery or renegotiation of the constitution”. Тези нужди могат да се включат в модел на индекси на двойни думи по следния начин. Групираме термините в съществителни, включително и собствени имена, (N) и функционални думи, включително членове и предлози, (X), сред други класове. Сега смятаме всеки стринг от

термини от формата на NX^N да бъде разширена двойда дума. Всяка такава разширена двойна дума прави термин в граматиката. Например:

renegotiation of the constitution

N X X N

За да изпълним заявка, използвайки такива индекси на разширена двойна дума, се нуждаем също да я разделим на N-ове и X-ове и тогава да сегментираме заявката в двойна дума. Този алгоритъм не винаги работи по оптимален начин, когато разделямъ дълги заявки във буlevи такива.

Идеята на индексите за двойни думи може да бъде разширена до по-дълги последователности от думи и ако индекса съдържа променлива дължина на последователностите от думи, обикновено се реферира като фразов индекс. Действително търсениято на единичен термин не се определя като индекс на двойна дума (ще трябва да сканирате речника за всички двойни думи, които съдържат термина) и ние ще се нуждаеме от индекс на единична дума. Въпреки, че винаги има шанс за фалшиво позитивно съвпадение, шансът за такова съвпадение върху фразови индекси с дължина 3 или повече става наистина много малък. Но от друга страна, запаметяването на дълги фрази има потенциал за генерално разширение на големината на речника. Поддържането на изчерпателни фразови индекси за фрази с дължина по-голяма от две е труда перспектива и дори използването на изчерпателен речник на двойни думи, значително ще разшири размера на речника.

2.4 Positional postings and phrase queries

to, 993427:

⟨ 1, 6: ⟨7, 18, 33, 72, 86, 231⟩;
2, 5: ⟨1, 17, 74, 222, 255⟩;
4, 5: ⟨8, 16, 190, 429, 433⟩;
5, 2: ⟨363, 367⟩;
7, 3: ⟨13, 23, 191⟩; ... ⟩

be, 178239:

⟨ 1, 2: ⟨17, 25⟩;
4, 5: ⟨17, 191, 291, 430, 434⟩;
5, 3: ⟨14, 19, 101⟩; ... ⟩

► **Figure 2.11** Positional index example. The word to has a document frequency 993,477, and occurs 6 times in document 1 at positions 7, 18, 33, etc.

2.4.2 Позиционни индекси (Positional indexes)

Поради дадената причина, индекса на двойните думи не е стандартно решение. По-скоро позиционните индекси са по-често използвани. Тук за всеки термин в речника запазваме срещанията на формата docID: (position1, position2, ...), както е показано на фигура 2.11, където всяка позиция е символен индекс в документа. Всяко срещане също ще запише честотата на термина.

За да изпълним фразова заявка все още се нуждаем от достъп до индивидуалните индекси за всеки уникален термин. Както преди, трябва да започнем с най-малко срещания термин и тогава да поработим върху бъдещото ограничение на списъка с възможните кандидати. В операцията на сливане се използва същата техника както преди, но по-скоро отколкото просто да проверите че и двата термина с в документа, трябва също да проверите, че техните позиции на срещане в документа са съвместими с фразовата заявка, която ще се оцени. Това изисква построяване на отмествания между думите.

2.4.3 Пример.

Предполагаме, че списъците със срещания за “to” и “be” са показани на фигура 2.11 и заявката е “to be or not to be”. Списъците със срещания за достъп са : to, be, or, not. Сега ще определим пресичането на списъците за “to” и “be”. Първо търсим за документи, които съдържат и двата термина. След това търсим места в списъците, където има срещане на “be” с токън индекс с едно по-висок от позицията на “to” и тогава ще търсим за друго срещане на всяка дума с токън индекс 4 по-висок от първото срещане. В списъците по-горе, моделът на срещанията, които са възможни е:

```

to: (...; 4: (...; 429, 433); ...)
be: (...; 4: (...; 430, 434); ...)

POSITIONALINTERSECT( $p_1, p_2, k$ )
1  $answer \leftarrow \langle \rangle$ 
2 while  $p_1 \neq NIL$  and  $p_2 \neq NIL$ 
3 do if  $docID(p_1) = docID(p_2)$ 
4     then  $l \leftarrow \langle \rangle$ 
5      $pp_1 \leftarrow positions(p_1)$ 
6      $pp_2 \leftarrow positions(p_2)$ 
7     while  $pp_1 \neq NIL$ 
8         do while  $pp_2 \neq NIL$ 
9             do if  $|pos(pp_1) - pos(pp_2)| \leq k$ 
10                then ADD( $l, pos(pp_2)$ )
11                else if  $pos(pp_2) > pos(pp_1)$ 
12                    then break
13                     $pp_2 \leftarrow next(pp_2)$ 
14                    while  $l \neq \langle \rangle$  and  $|l[0] - pos(pp_1)| > k$ 
15                        do DELETE( $l[0]$ )
16                        for each  $ps \in l$ 
17                            do ADD( $answer, \langle docID(p_1), pos(pp_1), ps \rangle$ )
18                             $pp_1 \leftarrow next(pp_1)$ 
19                             $p_1 \leftarrow next(p_1)$ 
20                             $p_2 \leftarrow next(p_2)$ 
21                        else if  $docID(p_1) < docID(p_2)$ 
22                            then  $p_1 \leftarrow next(p_1)$ 
23                            else  $p_2 \leftarrow next(p_2)$ 
24 return  $answer$ 

```

► **Figure 2.12** An algorithm for proximity intersection of postings lists p_1 and p_2 . The algorithm finds places where the two terms appear within k words of each other and returns a list of triples giving docID and the term position in p_1 and p_2 .

2.4.4 Размер на позиционните индекси.

Приемането на позиционните индекси разширява необходимостта от съхранение на срещанията значително, дори и ако компресираме стойността/отместването на потоците.

Наистина, преминаването позиционни индекси също променя асимптотичната сложност на пресичащите операции, защото броя на елементите за проверка сега граничи не с броя на документите, а с общия брой на символите в колекцията документи T . Сложността на буleva заявка е $\Theta(T)$ а не $\Theta(N)$. Обаче много приложения имат друг избор, освен да приемат това, тъй като повечето потребители очакват да имат функционалността на фразови и точни търсения.

Нека да определим необходимото пространство за да имаме позиционни индекси. Сега е необходимо да имаме запис за всяко срещане на термин. По такъв начин размера на индекса зависи от средния размер на документа. Средната уеб страница има по-малко от 1000 термина, но документи като SEC стокови пазари, книги, и дори някои поеми лесно достигат 100 000 термина. Нека да разгледаме термин с честота 1 на 1000 термина средно.

Document size	Expected postings	Expected entries in positional posting
1000	1	1
100,000	1	100

Докато точният брой зависи от типа на документа и от езика, който ще се индексира, някои правила очакват позиционните индекси да заемат от 2 до 4 пъти големината на непозиционните индекси и да очакват компресирането на позиционните индекси да бъде от една трета до половината от сировия текст (след отстраняване на маркерите) на оригиналните некомпресирани документи.

2.4.5 Комбинирани схеми

Стратегиите за индекси на двойни думи и позиционни индекси могат комбинирани. Ако потребителите често изпълняват заявки върху отделни фрази, като например “Michael Jackson”, е доста неефективно да пазим слети позиционни списъци със срещания. Комбинираната стратегия използва фразови индекси или просто индекси на двойни думи за определени заявки и използват позиционни индекси за останалите фразови заявки. Подходящи заявки за вкаране във фразовите индекси са тези, които са известни и базирани на основата на последните заявки. Но това не е единственият критерий: най-скъпите фразови заявки за оценяване са тези, които индивидуалните думи са общи, но желаната фраза е сравнително рядка. Добавянето на “Britney Spears” като фразов индекс може единствено да даде ускорим фактор на тази заявка от около 3, тъй като повечето документи, в които се споменава едната от думите са валидни резултати, докато добавянето на “The Who” като фразов индекс може да ускори тази заявка с фактор около 1000. Следователно, имайки последната фраза е желателно, дори ако това е относително по-малко обща заявка.

Williams et al. (2004) е оценил още по-сложна схема, в която работят индексите и на двата вида и допълнително частично индекс на следващата дума като по средата между първите две стратегии. За всеки термин, индексът на следващата дума запазва термина, който следва в документа. Те заключават, че такава стратегия позволява типично смесване на уеб фразови заявки да бъдат изпълнени през една четвърт от времето, необходимо при използването само на позиционни индекси, като се вземат 26% повече пространство, отколкото при използването само на позиционни индекси.

3 Речници и толерантно извличане

В Глави 1 и 2 разгледахме идеите, на които се основава обрнатия индекс (*inverted index*), използван за справяне с булеви и заявки за близост. Тук ще разгледаме техники, които се справят добре, както с типографски грешки в заявката, така и с различното изписване на някои думи. В параграф III.1. ще разработим структура от данни, която да ни помогне в търсенията на термини в лексиката (*the vocabulary*) на обрнатия индекс. В параграф III.2. ще разгледаме идеята за произволни заявки (*wildcard queries*) - заявки, съдържащи символа “*”, на чието място в заявката може да стои всякакъв низ (дори и празен). Това позволява на потребителя да търси думи, в чиито правопис не е сигурен или да търси документи, съдържащи няколко варианта на един термин в дадена заявка. Например заявката *automat**, ще търси документи съдържащи термините: *automat*, *automatic*, *automation*.

По нататък, в параграф III.3., ще се спрем на друг вид непрецизно зададени заявки и по-точно – правописните грешки, които потребителите допускат по грешка или защото търсеният термин няма недвусмислен правопис. Ще разгледаме подробно няколко техники за поправяна на допуснатите грешки в заявката, както термин по термин, така и за цели низове от термини. Накрая, в параграф III.4. ще изучим метод за търсене на термини, които са близки фонетично до тези от заявката. Това би било особено полезно в случаите, когато потребителят не знае точното изписване на даден термин (например име или фамилия).

В тази глава ще използваме много варианти на обрнатия индекс, затова под стандартен обрнат индекс, ще разбираме индекса описан в глави 1 и 2.

3.1 Структури за търсене в речници

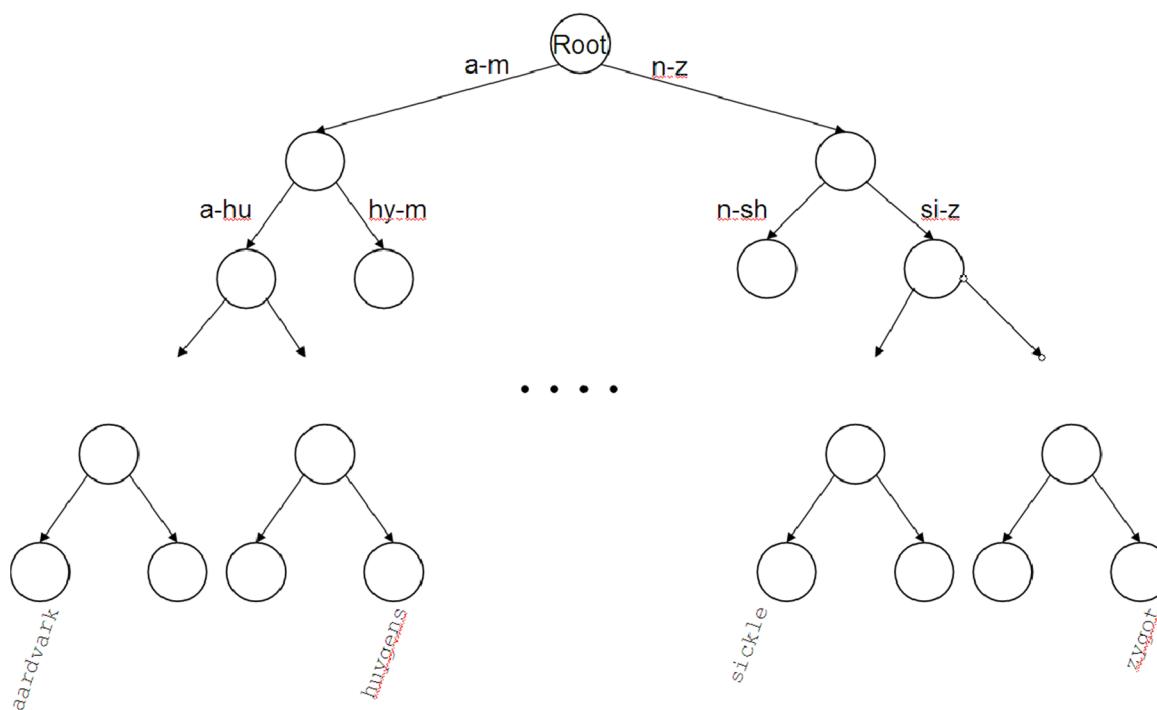
При даден обрнат индекс и заявка, нашата първа задача е да определим дали всеки термин от заявката съществува в лексиката и ако да, то да намерим указателя соченец към съответните постинги (*postings*). За тази проверка (в лексиката) се използва класическата структура наречена речник (*dictionary*) и има 2 обширни класа решения: хеширане (*hashing*) и дървета за търсене. В книгите по структури от данни, элементите на лексиката (в нашия случай термините) често са наричани ключове (*keys*). За да изберем между хеширане и дърво за търсене, трябва да си отговорим на следните въпроси: (1) Колко ключа се очаква да имаме? (2) Броят на ключовете е статичен ли ще бъде или ще се променя често – и в случай на честа промяна само ще бъдат добавяни ключове или и ще се трият такива? (3) Колко често различните ключове ще бъдат достъпвани (търсени)?

Хеширането се използва от някои търсачки, като метод за търсене в речници. Всеки термин от лексиката (ключ) се хешира с естествено число над достатъчно голямо пространство, таке че сблъсъци да са малко вероятни. Ако има сблъсъци, те се решават от спомагателни структури, които изискват грижи, за да се поддържат (така наречените идеални хеш функции, които са проектирани да възпрират сблъсъци, но те са по-сложни за изчисление и по-трудни за прилагане). В момента на заявката, хешираме всеки термин от нея по отделно и проследяваме указателя до съответните постинги, взимайки в предвид всяка логика за разрешаване на хеш сблъсъци. Няма лесен начин да намерим близки разновидности на термините от заявката (например думата „resume“ без и със ударение), тъй като те биха могли да бъдат хеширани с

различни числа. По-специално, ние не можем да търсим всички термини, започващи с представката “automat”, операция която в п. 3.2. ще изискаме от нашата система. И последно, в среда като глобалната мрежа, където обема на лексиката се увеличава, хеш функция проектирана да задоволява сегашните нужди ще бъде далеч недостатъчна след няколко години.

Дърветата за търсене, от своя страна, се справят с много от тези проблеми. Например те ни позволяват да изброим всички термини от лексиката започващи с “automat”. Най-известното дърво от този вид е бинарното (двоичното) дърво, в което всеки вътрешен възел има 2 наследника. Търсенията на термин започва от корена на дървото. Всеки вътрешен възел (включително и корена) представлява двоичен тест. В зависимост от резултата от този тест търсенията продължава по едно от двете под дървета. На фиг. 3.1 е даден пример за двоично дърво използвано за речник.

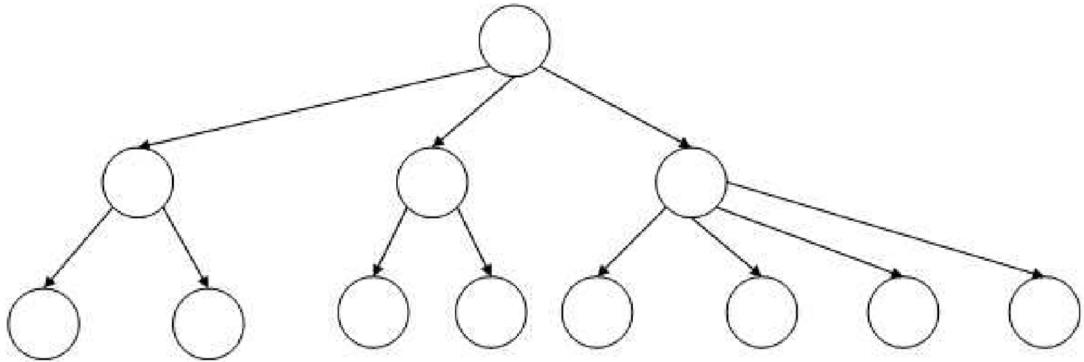
Фиг. 3.1



Ефективността на търсения ($O(\log M)$) на брой бинарни операции, където M е броят на възлите) зависи от това дървото да е балансирано, т.е. броят на възлите под двете поддървета на всеки възел трябва да се различават най-много с едно. Проблемът, който възниква с двоичните дървета е повторното балансиране - при добавяне или изтриване на възли от дървото, то трябва да бъде балансирано, за да се запазят свойствата му.

Един начин да се облекчи балансирането е да се позволи броят поддървета под вътрешен възел да варира във фиксирани граници. Дърво за търсене, което обикновено се използва за речници е В-дървото (Б-дървото). То е дърво за търсене, в което всеки вътрешен възел има брой наследници в интервала $[a, b]$, където a, b са естествени числа и са добре избрани. На фиг. 3.2 е представен пример при $a=2, b=4$.

Фиг. 3.2



Всеки клон под вътрешен възел, представлява отново тест за последователността от символи, както и в примера от фиг. 3.1 . На В-дървото може да се гледа като двоично дърво, на което няколко нива от възлите са се „срутили“ в едно. Това е особено изгодно, когато някой от речниците е локален за даден диск, в който случай това „срутване“ изпълнява функцията на предварително извличане на предстоящи двоични тестове. В такива случаи, числата a и b се определят от големините на дисковите блокове (disk blocks).

Трябва да се отбележи, че за разлика от хеширането, дърветата за търсене изискват символите използвани в документите (колекцията от документи) да имат (предварителна) наредба. Например буквите от А до Я. Други езици обаче, като китайският, не винаги имат строга подредба. Този проблем е преодолян, като в момента такива езици (японски, китайски) са приспособили стандартна система за подредба на своите знаци.

3.2 Произволни заявки (wildcard queries)

Произволните заявки се използват в следните ситуации: (1) Потребителят не е сигурен за правописа на даден термин от заявката (пр. Sydney или Sidney => S*dney); (2) Потребителят е наясно, че даден термин има множество правописи и (съзнателно) търси документи съдържащи коя и да е версия на изписването на термина (пр. colour и color); (3) Потребителят търси документи съдържащи разновидности на термин, които биха били хванати от стеминг, но не е сигурен дали търсачката извършва стеминг или не (пр. judicial или judiciary => judicia*); (4) Потребителят не е сигурен за интерпретацията на чужда дума (пр. заявката Universit* Stuttgart).

Заявка като mon* е известна като крайна произволна заявка (trailing wildcard query), защото символът * се среща само в края на низа. Дърво за търсене върху речника е удобен начин за справяне с крайни произволни заявки : движим се надолу по дървото, следвайки символите m, o, n подред и щом стигнем символа n изброяваме множеството W от всички термини в речника с представка mon. Въщност ние извършваме |W| на брой справки в стандартния обърнат индекс, за да извлечем всички документи съдържащи термини от W.

Но какво да правим с произволни заявки, съдържащи * не само в края? Преди да отговорим на този въпрос, ще се занимаем с едно обобщение на крайните произволни заявки. Първо нека разгледаме водещи произволни заявки (leading wildcard queries). Това са заявки от вида *mon (символът * се среща само в началото на низа). Разглеждаме обърнато В-дърво върху речника,

т.е. всеки път от корен до листо на В-дървото отговаря на термин от речника написан отзад напред. По този начин терминът *lemon* в обрънато В-дърво ще изглежда по следния начин: *root-n-o-m-e-l*. Обхождане от горе надолу на това дърво ще ни даде всички термини R в лексиката с дадената представка.

В същност, използвайки нормално В-дърво заедно с обрънато такова, ние можем да се справим с още по-общ случай: произволни заявки, в които има един символ * на произволна позиция (пр. *se*mon*). За да направим това, използваме нормалното В-дърво, за изброяване на множеството W, състоящо се от термините от речника започващи със *se*, след това използваме обрънатото В-дърво, за да изброим множеството R от речникови термини, завършващи на *mon*. Взимаме сечението на W и R. Накрая използваме стандартния обрънат индекс, за да извлечем всички документи съдържащи се в това сечение. По този начин видяхме как да се справяме в случаите, когато имаме единствен символ * в заявката.

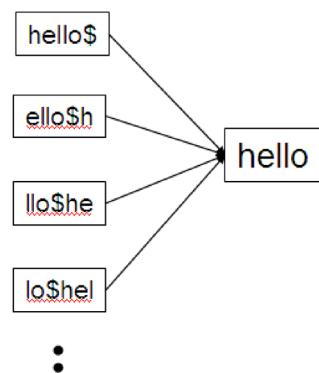
3.2.1 Обобщени произволни заявки

Сега ще разгледаме два начина за справяне с обобщени произволни заявки. И двата подхода имат една и съща стратегия - представяме дадената произволна заявка *qw* като булева заявка Q върху специално конструиран индекс, така че отговорът на Q е надмножество на множеството от термините от лексиката, пасващи на *qw*. След това сравняваме всеки от термините от отговора на Q с *qw*, като изхвърляме тези които не пасват на *qw*. На този етап имаме термините от лексиката пасващи на *qw* и можем да се обърнем към стандартния индекс.

3.2.2 Permuterm индекс (индекс от разместени термини)

Първият специален индекс, който ще конструираме за произволни заявки, е permuterm индексът или индексът от разместени термини. Ще използваме символът „\$“, за да означаваме края на всеки термин. По този начин думата *hello* ще бъде записана като *hello\$*. Конструираме permuterm индексът като в него записваме всички възможни ротации на всеки от термините (записани със символа \$). На фиг.3.3 е показан пример за permuterm индекс за думата *hello*.

Фиг. 3.3



Множеството от така ротираните термини в permuterm индекса ще наричаме permuterm лексика.

Как този индекс ни помага да се справим с произвольните заявки? Нека загледаме заявката *m*n*. Ще ротираме тази заявка, така че символът * да застане в края на низа. Сега имаме низа *n\$m**.

По нататък търсим този низ в permute m индекса, където търсенето на $n\$m^*$ (чрез дърво за търсене) води измежду други и до ротации на термините *man* и *moron*.

Сега, след като permute m индекса ни позволява да идентифицираме оригиналните термини от лексиката пасващи на произволните заявки, остава да потърсим тези термини в стандартния обрънат индекс, за да извлечем съответните документи. По този начин можем да се справим с произволни заявки, които съдържат един единствен символ *. Но остава въпросът, какво да правим със заявки от вида *fi*mo*er*? В този случай първо изброяваме термините от речника, които са в permute m индекса на $er\$fi^*$. Не всички такива термини ще имат низа „*то*“ по средата. Чрез гълъно изброяване ще проверим всеки от кандидатите дали съдържа „*то*“. В нашия пример, терминът *fishmonger* ще мине пресяването, но *filibuster* няма. След това пускаме останалите „оцелели“ термини и през обрънатия индекс за извлечане на документите. Недостатък на permute m индекса е, че речникът става доста голям, заради добавянето на всички ротации.

Забележете тясното взаимодействие между В-дървото и permute m индекса по-горе. Действително, то предполага, че на структурата трябва да се гледа като на permute m В-дърво. Все пак ние следваме традиционната терминология, описвайки permute m индекса като отделна от В-дървото структура, позволяваща ни да изберем ротациите с определена представка.

3.2.3 К-грамни индекси за произволни заявки

Използването на прост permute m индекс води до значително увеличение на термините, заради ротациите. За речник с английски термини това увеличение може да бъде до 10 пъти. Сега ще разгледаме втора техника, наречена к-орков индекс, за справяне с произволните заявки. К-грам се нарича последователност от k символа. По този начин *cas*, *ast*, *stl* са 3-грами от термина *castle*. Ще използваме символът \$, за да отбелнязваме началото или края на термин. Следователно множеството от всички 3-грами на термина *castle* е: $\$ca$, *cas*, *ast*, *stl*, *le\$*.

В к-граммен индекс, речника съдържа всички к-грами, съдържащи се в термините от лексиката. Всеки списък с постинги (posting list) сочи от к-грама към всички термини от лексиката съдържащи този к-грам. Например 3-грама *etr* ще сочи към термини като *metric* и *retrieval*.

Как този индекс ни помага? Нека имаме произвольната заявка *re*ve*. Търсим всички документи съдържащи термин започващ с *re* и завършващ на *ve*. Извикваме двоичната заявка $$re \text{ AND } ve\$$. Това се проверява (търси) в 3-граммния индекс и извиква списък с пасващи термини като *relieve*, *remove*, *retrieve*. Всеки един от тези термини след това се проверява в стандартния обрънат индекс и се извличат документите, в които се срещат.

Тук обаче се появява трудност с к-граммите индекси, която изисква направата на още една стъпка. Нека разгледаме 3-граммния индекс описан по-горе за заявката *red**. По описанния начин, първо извикваме булевата заявка $$re \text{ AND } red$. Това води до връщане на термини като *retired*, което съдържа обединението на *\$re* и *red*, но не отговаря на първоначалната заявка.

За да се справим с този проблем е необходима направата на една допълнителна стъпка – така нареченото след-фильтриране, при което термините върнати от булевата заявка към 3-граммния индекс се проверяват един по един дали отговарят на първоначалната заявка. Това е просто сравняване на низове, което премахва термини като *retired*.

Видяхме, че произволната заявка въсъщност води до множество еднотерминни заявки, към стандартния обърнат индекс. Търсачките позволяват комбиниране на произволни заявки с булеви оператори, например re^*d AND fe^*ri . Тъй като всяка произволна заявка се преобразува до дизюнкция на еднотерминни заявки, то интерпретацията на този пример е, че имаме конюнкция на дизюнкции: търсим всички документи съдържащи кой да е термин пасващ на re^*d и кой да е термин пасващ на fe^*ri .

Дори без булеви комбинации на произволни заявки, процесите за изпълнението им са доста скъпи, заради добавеното търсене в специален индекс, филтрирането и накрая търсенето в стандартния обърнат индекс. Поради тази причина, дори една търсачка да поддържа задаването на произволни заявки, то тази опция най-често е скрита в интерфейса (напр. Advanced Query) и остава неизползвана от повечето потребители. Това се прави с цел да се предотврати хабенето на ресурси за търсене на заявки от вида a^* , които някои потребители биха написали без реална нужда и смисъл.

3.3 Поправка на правописа

Ще разгледаме въпроса за поправяне на правописни грешки в заявките. Целта ни е нашата търсачка да върне резултат за *carrot*, дори когато нашият потребител е написал заявка *carot*. Ще направим две стъпки, за да разрешим този проблем: първата е базирана на **редакционно разстояние** (edit distance), а втората на к-грамно припокриване. Преди да навлезем в алгоритмичните подробности на тези методи, първо ще разгледаме как търсачките осигуряват проверка на правописа, като част от потребителските практики.

3.3.1 Прилагане на корекция на правописа

Има два основни принципа, стоящи в основата на повечето алгоритми за корекция на правописа: (1) От множество правилни правописи на дадена грешно написана заявка, се връща „най-близката“ такава. Това означава, че трябва да имаме критерии за близост между 2 заявки (ще разработим такива мярки за близост в параграф III.3.3.). (2) Когато две правописно правилни заявки са еднакво близко до написаната от потребителя такава, то тогава нашата система трябва да върне по-популярната от двете. Например: *grunt* и *grant* са еднакво вероятни за заявката *grnt*. Ако *grant* се среща по-често в документите, то търсачката трябва да избере него като поправка. Съществува и друг подход, който е по-популярен особено в интернет. При него популярността не зависи от броя срещания в документите, а от броя потребители написали съответните заявки. Така, ако *grunt* е бил по-често търсен (от повече потребители), то търсачката ще предложи него пред *grant*, въпреки че се среща в по-малко документи.

Функционалността на алгоритмите за поправка на правописа се предлагат на потребителя по един от следните начини:

- a) На заявката *carot* винаги се връщат документите съдържащи *carot*, както и всички документи съдържащи всяка правописно поправена версия на *carot* (*carrot*, *tarot*).
- b) Както в (a), но само в случаите, когато термина *carot* не е в речника.
- c) Както в (a), но само в случаите, когато заявката върне по-малко от фиксиран брой резултати (например по-малко от 5).

- d) Когато заявката върне по-малко от фиксиран брой резултати, то търсачката предлага алтернативни поправки на правописа, например „Did you mean carrot?“.

3.3.2 Видове поправки на правописа

Ще се съсредоточим върху два основни вида поправка на правописа, които наричаме изолирано редактиране (isolated-term correction) и контекстово редактиране (context-sensitive correction). В изолираното редактиране се опитваме да поправяме термините от заявката един по един, дори да имаме заявка от множество от термини. Този вид редактиране обаче ще пропусне грешката в заявката: “flew form Heathrow”, тъй като думите разглеждани отделно са правилно написани. Ще започнем с разглеждането на две техники за изолирано редактиране: редакционно разстояние и к-грамно припокриване, след което ще продължим с контекстовото редактиране.

3.3.3 Редакционно разстояние

Дадени са два символни низа s_1 и s_2 , редакционното разстояние (edit distance) между тях е минималния брой редакционни операции (edit operations) необходими за преобразуването на s_1 в s_2 . Най-често редакционните операции позволяни за тази цел са (i) вмъкване на символ в низ, (ii) изтриване на символ от низ, (iii) замяна на символ от низ с друг символ. За тези операции, редакционното разстояние е известно, като разстояние на Левенщайн (Levenshtein distance). Например, редакционното разстояние между cat и dog е 3. В действителност идеята за редакционно разстояние може да бъде обобщена да позволява различни тегла за различните видове редакционни операции, например по-голямо тегло може да се даде на замяната на символ s със символ p , отколкото на замяната със символ a (последното е по-близо до s на клавиатурата). Настройката на теглата по този начин, в зависимост от вероятността на заместването на буквите една с друга е много ефективна в практиката (виж секция III.4 за проблема с фонетичното подобие). Въпреки това, ще се фокусираме върху случая, в който всички редакционни операции имат едни и същи тегла.

Добре известно е как да изчислим сложността $O(|s_1| \cdot |s_2|)$ за намиране на редакционното разстояние между два низа, където $|s_i|$ означава дължината на низа s_i . Идеята е да използваме алгоритъма за динамично програмиране от фиг.3.5, където символите в s_1 и s_2 са дадени като масиви. Алгоритъмът запълва с цели числа матрицата m , чиито размери са равни на дълчините на низовете между които търсим редакционното разстояние. След изпълнението на алгоритъма, елементът на позиция (i,j) на матрицата ще съдържа редакционното разстояние между низовете състоящи се от първите i символа на s_1 и първите j символа на s_2 . Основната стъпка на динамичното програмиране се съдържа в редовете 8-10 на фиг. 3.5, където трите величини, от които вземаме минимума, съответстват на заместване на символ в s_1 , поставяне на символ в s_1 и на поставяне на символ в s_2 .

Фиг. 3.6 показва пример на пресмятане на разстоянието на Левенщайн по фиг.3.5. Всяка клетка $[i,j]$ е разделена на 2×2 по-малки клетки. Долната дясна по-малка клетка е минимума от другите три клетки. Другите три входа са трите входа $m[i-1, j-1] + 0$ или 1 в зависимост от това дали $s_1[i] = s_2[j]$, $m[i-1, j]+1$ и $m[i, j-1]+1$. Клетките с италик числа показват пътя, по който определяме разстоянието на Левенщайн.

Фиг. 3.5

```

EDITDISTANCE( $s_1, s_2$ )
1  int  $m[i, j] = 0$ 
2  for  $i \leftarrow 1$  to  $|s_1|$ 
3  do  $m[i, 0] = i$ 
4  for  $j \leftarrow 1$  to  $|s_2|$ 
5  do  $m[0, j] = j$ 
6  for  $i \leftarrow 1$  to  $|s_1|$ 
7  do for  $j \leftarrow 1$  to  $|s_2|$ 
8    do  $m[i, j] = \min\{m[i - 1, j - 1] + \text{if } (s_1[i] = s_2[j]) \text{ then } 0 \text{ else } 1, f_i, m[i - 1, j] + 1, m[i, j - 1] + 1\}$ 
9
10
11 return  $m[|s_1|, |s_2|]$ 

```

Фиг. 3.6

		f	a	s	t
	0	1 1	2 2	3 3	4 4
c	1	1 2	2 3	3 4	4 5
	1	2 1	2 2	3 3	4 4
a	2	2 2	1 3	3 4	4 5
	2	3 2	3 1	2 2	3 3
t	3	3 3	3 2	2 3	2 4
	3	4 3	4 2	3 2	3 2
s	4	4 4	4 3	2 3	3 3
	4	5 4	5 3	4 2	3 3

Поправянето на правописни грешки обаче, изисква повече от изчисляване на редакционно разстояние. Дадено е множество S от низове (съответстващи на термините в лексиката) и заявка q , търсим низовете във V с най-малко редакционно разстояние от q . Можем да разглеждаме това като разшифроваш проблем, в който кодираните думи (низовете във V) са описани предварително. Явният начин за извършването на това, е да се изчисли редакционното разстояние от q до всеки низ във V , преди да изберем низа с минимално редакционно разстояние. Това източително търсене е прекалено скъпо. Съответно, се ползват различни евристики за ефективно извлечане на термини, освен използването на най-малко редакционно разстояние от терминът или термините на заявката.

Най-лесната такава евристика е да ограничим търсенето до термини, започващи със същата буква като на заявката. Ще се надяваме правописните грешки да не се случват на първия символ на заявката. По-сложен вариант на тази евристика е да използваме версия на *permuteerm* индекс на термин, в който ние пропускаме символа за край на думата $\$$. Смятаме множеството от всички ротации на низа q от заявката. За всяка ротация r от това множество, ние обръщаме В-дървото в *permuteerm* индекс, след това извлечаме всички речникови термини, които имат ротация започваща с r . Например, ако $q=mase$ и считаме ротацията $r=sema$, ние ще извлечем речникови термини като *semantic* и *semaphore*, които нямат малко редакционно разстояние до q . За съжаление, ще пропуснем по-уместните термини като *mase* и *mare*. За да се справим с

това, за всяка ротация пропускаме последните ℓ символа преди да обърнем В-дървото. Това подсигурява съдържанието на „дълъг“ общ низ във всяка колекция R от термини, извлечени от речника. Стойността на ℓ може да зависи от дължината на q. Алтернативно, можем да я фиксираме на стойност 2, например.

3.3.4 К-грамни индекси за коригиране на правопис

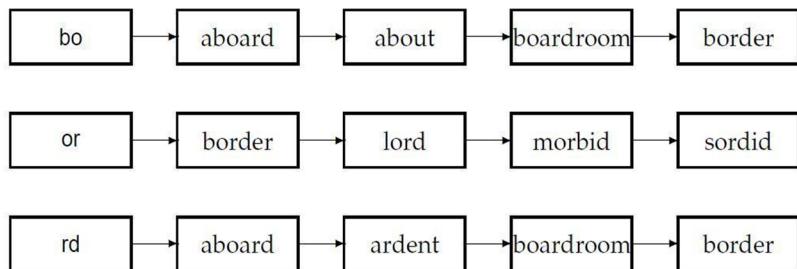
За да се ограничи допълнително множеството от термини в лексиката, за които изчисляваме редакционните разстояния до термина на заявката, сега ще покажем как да извикаме к-граммен индекс (the k-gram index) от секция III.2.2), за да помогнем с извлечането на термини в лексиката с малко редакционно разстояние до заявката q. Веднъж получени тези термини, вече можем да намерим този на най-малко редакционно разстояние от q.

Действително, ние ще използваме к-граммен индекс за извлечане на термини в лексиката, които имат много к-грами като цяло в заявката. Ще поспорим за приемлива дефиниция на „много к-грами като цяло“. Процеса на извлечане се основава на еднократно сканиране през постингите за к-грами в низа q на заявката.

Двутрамният (или биграмният) индекс на фиг.3.7 показва част от постинги за три биграма на заявката bord. Да предположим, че искаме да извлечем термини в лексиката, които съдържат най-малко два от тези три биграма. С едно сканиране на постингите ще можем да изброим всички такива термини. В примера на фиг. 3.7 ние можем да изброим aboard, boardroom и border.

Това директно прилагане на линейното сканиране за еднакви постинги веднага разкрива недостатъкът на обикновеното съвпадение на термини в лексиката, да съдържат фиксиран брой к-грами от заявката q - термини като boardroom, неправилно се броят за коректни. Следователно, ние искаме повече измервания за препокриващите се к-грами между термините в лексиката и q. Пресичането при линейното сканиране може да се нагласи, когато мярката за препокриване е **кофициентът на Жакард** (Jaccard coefficient), дефиниран като $|A \cap B| / |A \cup B|$ за измерване на препокриването между две множества A и B. Двете множества, които считаме за множеството от к-грами на заявката q и множеството от к-грами на термин в лексиката. Докато сканирането протича, ние преминаваме от един термин t в лексиката към друг, изчисляйки в движение кофициента на Жакард между q и t. Ако кофициентът надхвърли определен prag, ние добавяме t към изхода, ако не е, ние се местим на следващия термин в постингите. За да изчислим кофициента на Жакард, се нуждаем от множествата от к-грами в q и t.

Фиг. 3.7



Тъй като сканираме постингите за всички к-грами в q , ние веднага имаме тези к-грами на разположение. Ние можем да изброим к-граммите на t в движение, но в практиката това е не само бавно, но и потенциално неосъществимо, тъй като, по всяка вероятност елементите на самите постинги не съдържат целия низ t , а по скоро някакво кодиране на t . Изключително важно наблюдение за изчисляването на коефициента на Жакард е, че ние се нуждаем от дължината на низа t . За да видим това, ще се върнем на примера от фиг.3.7 и смятаме, че сканирането на постингите за заявката $q=bord$ е достигнало термина $t=boardroom$. Ние знаем, че два биграма съвпадат. Ако постингите пазят броя на биграмите в $boardroom$, ние имаме цялата информация, която искаме, за да изчислим коефициентът на Жакард и той ще бъде $2/(8+3-2)$ - числителят се получава от броя на съвпадали постинги (2 от b и rd), докато знаменателят е сумата на броя на биграмите в $bord$ и $boardroom$ и максимум броя на съвпадали постинги.

Ние можем да заменим коефициента на Жакард с други мерки, които позволяват ефективни изчисления по време на сканирането на постингите. Един метод, който има някаква емпирична поддръжка, е да използваме първо к-грамен индекс за да изброим множество с кандидати за термини в лексиката, които са потенциални поправки на q . Тогава изчисляваме редакционното разстояние от q до всеки термин в това множество и избираме термините от него с малко редакционно разстояние до q .

3.3.5 Коригиране на правописа в зависимост от контекста

Изолирано редактиране няма да се справи с поправянето на грешки, като например „flew form Heathrow”, където всичките три термина на заявката са верни по отношение на правописа. Когато фраза като тази получи малко документи, тогава търсачката може да предложи поправената заявка „flewfromHeathrow”. Най-лесния начин да се направи това, е да изброим корекциите на всеки от трите термина на заявката (като използваме методите от секция III.3.4), въпреки че всеки термин от заявката е правилно написан, правим заместване с всяка възможна корекция във фразата. За примера „flew form Heathrow”, ние изброяваме фрази като „fled form Heathrow” и „flew fore Heathrow”. За всяка заместена фраза, търсачката изпълнява заявката и определя броя на съвпаденията. Това изброяване може да бъде скъпо, ако ние намерим много корекции на отделените термини, тъй като ние можем да отчетем голям брой на алтернативни комбинации. Няколко практики се използват за отсичане на такова пространство. В примера по-горе, както ние разширихме възможностите за *flew* и *form*, ние запазваме най-честите комбинации в колекцията или в логовете на заявката, които съдържат предишните заявки на потребителите. Например, ние ще запазим „flewfrom” като алтернатива, да се опитаме да разширим с до три термина коригирана заявка, но не и „fledfore” или „fleaform”. В този пример фразата „fled fore” е малко вероятно да се сравни с фразата „flew from”. Тогава ние се опитваме да разширим списъка от топ фрази с по две думи, за коригиране на *Heathrow*. Като алтернатива за използване на статистика за фрази от две думи, ние може да запазим заявки на потребители, като това разбира се, може да включи и заявки с правописни грешки.

3.4 Фонетични корекции

Последната техника която ще разгледаме за толерантно извличане се занимава с фонетичните корекции. Грешки при писането се появяват, когато потребителите записват заявка по звучене. Такива алгоритми са специално приложими за търсене на имена на хора. Основната идея е, да генерираме за всеки термин „фонетичен хеш”, така че близките по звучене термини да сочат към една и съща стойност. Идеята е произлязла от работата в международен полицейски отдел за издиране на престъпници, въпреки различния начин по който се пишат имената им в различните страни. Основно се е използвало за поправяне на фонетични грешки за подходящите имена.

Алгоритми за такива фонетични хепириания са известни като soundex алгоритми. Има оригинален soundex алгоритъм, с различни варианти, построени по следната схема:

1. Всеки термин се представя в 4-символна намалена форма. Построяваме индексиране от тези намалени форми на оригиналните термини, което наричаме soundex индекс.
2. Правим същото и с термините на заявката.
3. Когато заявката се извика за soundex съвпадение, търсим по soundex индекса.

Вариациите в различните алгоритми трябва да се справят с обръщането на термините в 4-символна форма. Обикновено обръщанията водят до 4-символен код, започващ с буква, а останалите три са цифри между 0 и 9.

- 1) Оставяме първата буква от термина.
- 2) Променяме всички срещания на следните букви на '0'(нула): 'A', 'E', 'I', 'O', 'U', 'H', 'W', 'Y'.
- 3) Променяме буквите на цифри, както следва:
 - a) 1 - B, F, P, V
 - b) 2 - C, G, J, K, Q, S, X, Z
 - c) 3 - D, T
 - d) 4 - L
 - e) 5 - M, N
 - f) 6 - R
- 4) Премахваме едната цифра от повтарящи се съседни цифри.
- 5) Премахваме всички нули от получения низ. Допълваме резултатния низ с нули и връщаме първите четири символа, които ще се състоят от буква и три цифри.

Например на Hermann се съпоставя H655. За дадена заявка, първо изчисляваме нейният soundex код и тогава получаваме всички термини в лексиката съвпадащи с този soundex код от soundex индекса, преди да получим резултата на заявката от стандартния индекс.

Този алгоритъм лежи на няколко наблюдения: (1) гласните се разглеждат като взаимнозаменяеми в записаните имена; (2) константите с близко звучене са поставени в еквивалентни класове. Това често води до свързани имена да имат едни и същи soundex кодове. Тези правила вършат работа в много случаи, конкретно за европейските езици, но те са зависими от писмената система. Например китайските имена могат да се запишат в уейд-джайлс (Wade-Giles) или пинин (Pīnyīn) транскрипция. Съществуват известни сходства и

разлики в двете транскрипции, като например *hs* от уейд-джайлс и *x* от пинин се съпоставят на *2*, но, *j* от уейд-джайлс и *r* от пинин се съпоставят различни кодове.

3.5 Използвана и допълнителна литература

Knuth (1997) е подробен източник на информация за търсещи дървета, включително В-дървета и тяхната употреба при търсенето в речници от термини.

Garfield (1976) дава едно от първите пълни описания на *permuteerm* индекс. Ferragina и Venturini (2007) дават подход, отнасящ се за намаляване на пространството от *permuteerm* индекси.

Едно от най-ранните официални разглеждания на корекциите при правопис се дължат на Damerau (1964). Идеята за редакционното разстояние идва от Levenshtein (1965), а алгоритъма от фигура 3.5 от Wagner и Fischer (1974). Peterson (1980) и Kukich (1992) разработват варианти на методите, базирани на редакционни разстояния, достигайки до детайлното емпирично изучаване на няколко метода на Zobel и Dart (1995), които показват, че к-грамното индексиране е много ефективно в намирането на кандидати за несъответствие. Gusfield (1997) е стандартна референция за низови алгоритми, отнасящи се за редакционно разстояние.

Вероятностни модели за коригиране на правопис са ръководени от Kernighanetal. (1990) и след това са доразвити от Brill и Moore (2000) и Tuatanova и Moore (2002). В тези модели, грешната заявка се разглежда като вероятностно отклонение от правилната заявка. Те имат подобна математическа основа към езикови методи и също осигуряват начини на обединяваци фонетични прилики, близко разположение върху клавиатурата и данни от истински грешки на потребители. Cucerzan и Brill (2004) показват как тази задача може да бъде разширена до изучаването на модели за коригиране на грешки, базирани на преформулиране на заявки в логовете на търсачките.

Алгоритъмът soundex се приписва на Margaret K. Odell и Robert C. Russelli. Версията описана тук се основава на Bourne и Ford (1961). Zobel и Dart (1996) изчисляват различни фонетични алгоритми и намират, че вариант на soundex алгоритъма не работи добре за всички правописни грешки, но другите алгоритми базирани на фонетични прилики на произнасянето на термин се справят добре.

4 Конструиране на индекси

4.1 Основи на хардуера

Разработката на индексиращи алгоритми се влияе до голяма степен от хардуерните ограничения. Затова, когато се изгражда система за извлечане на информация трябва да се има предвид следното:

- Достъпът до данни в паметта е многократно по-бърз от този в твърдия диск. Следователно трябва да се стремим да държим колкото се може повече данни в паметта, особено тези данни, до които ни трябва редовен достъп. Техниката да държим често използвани дискови данни в паметта ще наричаме *кешране*.
- Когато се извършва операция по четене или писане от диска е нужно известно време за преместване на главата до мястото, където се намират данните. Това време се нарича *време за достъп*, означаваме го с s (*seek time*) и обикновено е около 5 милисекунди. По време на движението на главата от едно място на друго тя не може да чете данни. Затова, с цел да се максимизира скоростта на трансфер на данни, парчетата данни, които ще бъдат прочетени наведнъж трябва да бъдат разположени последователно върху диска. Това може да намали значително времето за трансфер на големи обеми данни.
- Операционните системи обикновено четат и записват данни на блокове (обикновено по 8, 16, 32, или 64KB). Следователно, прочитането на един единствен байт би отнело също толкова време, колкото и прочитането на целия блок. Ще наричаме *буфър* тази част от оперативната памет, в която се държи съдържанието на блок, който бива прочетен или записан.
- Времето за трансфер на данни от диска до паметта, ако главата е на правилната позиция, наричаме *време за трансфер на байт*, описано с b (*transfer time per byte*) и обикновено е от порядъка на $0,02\mu s$, или $2 \times 10^{-8}s$. Тези трансфери се извършват от шина, а не от процесора, следователно през това време процесора е свободен да обработва данни. Можем да се възползваме от това като съхраняваме данните компресирани – обикновено това може да доведе до ускоряване на трансфера на данни.
- Сървърите използвани за извлечане на информация обикновено имат от няколко до няколко десетки гигабайта оперативна памет и с няколко мащаба повече дисково пространство.

От значение са и характеристиките на процесора – *работна честота* (*clock rate*), която е от порядъка на $10^9 s^{-1}$ и време за изпълнение на *операция от ниско ниво* (lowlevel operation), което е от порядъка на $10^{-8}s$ и означаваме с ρ .

4.2 Блоково индексиране базирано на сортиране (Blockedsort-basedindexing)

Основните стъпки в конструирането на индекс са описани в точка 1.1. Първо се прави обхождане на колекцията с документи за да съберем всички двойки term (термин) – docID (номер на документ). След това сортираме таблицата така, че term да стане първичен ключ, а docID – вторичен. Накрая за всеки термин правим списък със срещания (postingslist), съдържащ docID на всеки документ, в който термина се споменава. Можем да добавим и статистически данни, като напр. брой документи, в които се среща термина, брой срещания на термина в даден документ. За малки колекции, всичко това може да се извърши в паметта.

За да бъде ефективно конструирането на индекс на голяма колекция от документи, изискващ ползването на вторичната памет (твърди дискове), термините ще бъдат представени с termID (вместо буквально представяне под формата на символен низ), където *termID* е уникален сериен номер. Можем да построим таблицата, указваща кой termID отговаря на кой термин, докато обхождаме колекцията от документи, както прави описаният алгоритъм, но е възможен и друг подход – да направим две обхождания – при първото обхождане построяваме таблицата с термините а при второто – обърнатият индекс.

За пример използваме колекцията *Reuters-RCV1*, която съдържа около 1GB текст. Основните статистически показатели за нея са:

- Брой документи (N) – 800 000;
- Брой думи (T , tokens) – 100 000 000;
- Среден брой думи в документ (L_{ave}) – 200;
- Брой термини (M , terms) – 400 000;
- Среден брой байтове за дума (вкл. интервали и пунктуация) – 6;
- Среден брой байтове за дума (без интервали и пунктуация) – 4,5;
- Среден брой байтове за термин – 7,5;

В днешни дни често се работи с колекции, които са десетки или стотици пъти по-големи и в такива случаи сортирането на двойки termID–docID в паметта става непосилно дори за големи компютри. Когато първичната памет е недостатъчна се налага да ползваме *външен софтуерен алгоритъм* т.е. такъв, който ползва вторичната памет. За да достигнем приемлива скорост, най-важното изискване към такъв алгоритъм е да минимизира броя достъпи до произволни (непоследователни) сектори на диска. Едно възможно решение е *Блоково индексиране базирано на сортиране* (*Blockedsort-basedindexing*), или *БИБС(BSB)*. БИБС първо (1) разделя колекцията на равни части, после (2) сортира двойките termID–docID на всяка част в паметта, после (3) записва временните сортирани резултати на диска и накрая (4) слива всички временни резултати в крайния индекс.

Алгоритъма обхожда документи и събира в паметта двойки termID–docID, докато запълни един цял блок с определен размер (такъв, че блоковете да могат да се събират и сортират в първичната памет). След това блокът бива обърнат и записан върху диска. *Обърнатото* включва две стъпки. Първо сортираме двойката termID–docID. После събираме всички двойки termID–docID с еднакъв termID в списък със срещания, където всяко *фещане* е просто един docID. Резултатът, обърнат индекс за блока, който току що сме прочели, бива записан на диска.

Извършваме това с примерната колекция, разделяйки я на 10 блока – всеки от тях е обърнат индекс на една част от колекцията.

В последната стъпка алгоритъма едновременно слива десетте блока в един голям индекс. Следият индекс представлява обединението на всички блокове, като на тези termID, които се срещат в повече от един блок им се обединяват списъците със срещания. За да се извърши сливането отваряме едновременно десетте файла, съдържащи блоковете и поддържаме малък буфер за четене от всеки от тях и буфер за запис на крайния слят индекс. На всяка стъпка избираме най-ниския termID, който още не е излязъл от десетте буфера, използвайки приоритетна опашка или друга подобна структура от данни. Всички списъци със срещания за този termID се четат и сливат и следият списък бива записан обратно на диска. Всеки буфер за четене се попълва от своя файл когато е необходимо.

Времевата сложност на алгоритъма БИБС е $\Theta(T \log T)$, защото стъпката с най-голяма времева сложност е сортирането и T (броя думи) е горна граница за броя на различни двойки termID–docID, които трябва да сортираме. Но реалното време за индексиране обикновено се определя основно от времето необходимо за обработването на документите и за крайното сливане.

4.3 Индексиране в паметта с едно обхождане (Single-pass-in-memory-indexing)

Блоковото индексиране базирано на сортиране има отлична мащабируемост, но се нуждае от структура от данни за връзката между термини и termID. За много големи колекции, тази структура от данни може да не се събира в паметта. По мащабируема алтернатива е *IIндексиране в паметта с едно обхождане* (*Single-pass in-memory indexing*) или *ИПЕО* (*SPIMI*). ИПЕО използва термини вместо termID, записва речника на всеки блок на диска и после започва нов речник за следващия блок. ИПЕО може да индексира колекции от всякакъв размер, стига да има достатъчно дисково пространство.

Алгоритъма на ИПЕО е демонстриран с псевдокод:

```
SPIMI-INVERT(token_stream)
1 output_file = NEWFILE()
2 dictionary = NEWHASH()
3 while (free memory available)
4   do token  $\leftarrow$  next(token_stream)
5     if term(token)  $\notin$  dictionary
6       then postings_list = ADDTODICTIONARY(dictionary, term(token))
7       else postings_list = GETPOSTINGSLIST(dictionary, term(token))
8       if full(postings_list)
9         then postings_list = DOUBLEPOSTINGSLIST(dictionary, term(token))
10      ADDTOPOSTINGSLIST(postings_list, docID(token))
11    sorted_terms  $\leftarrow$  SORTTERMS(dictionary)
12  WRITEBLOCKTODISK(sorted_terms, dictionary, output_file)
13  return output_file
```

Частта от алгоритъма, която обработва документи и ги превръща в поток от двойки term-docID, което наричаме токентук, е пропусната. SPIMI-INVERT се повиква многократно, докато цялата колекция бъде обработена.

Думите се обработват една по една (ред 4) при всяко следващо извикване на SPIMI-INVERT. Когато някой термин се срещне за първи път, той се добавя към речника (който се реализира най-добре с хеш таблица) и се създава нов списък със срещания (ред 6). Извикването на ред 7 връща този списък със срещания за последващи появи на термина.

Една разлика между БИБС и ИПЕО е, че ИПЕО добавя срещанията директно в списъка със срещанията (ред 10). Вместо първо да събираме всички двойки termID-docID и тогава да ги сортираме (както направихме в БИБС), всеки списък със срещания е динамичен (т.е., размерът му се променя докато расте) и е непосредствено достъпен за добавяне на още срещания. Това има две преимущества: по-бързо е, защото не изисква сортиране, и пести памет, защото следим за кой термин отнова списъка със срещания, така че termID на срещанията не е нужно да се пазят. В резултат, блоковете, които отделните извиквания на SPIMI-INVERT могат да обработят са много по-големи и като цяло процесът по конструиране на индекса е по-ефективен.

Понеже не знаем колко голям ще бъде списъка със срещанията за даден термин когато го достигнем за първи път, първо ще заделим място за малък списък със срещания и ще го удояваме всеки път когато се запълни (редове 8-9). Това означава, че част от паметта се хаби, което е за сметка на спестената памет от пропускането на termID в междинните структури от данни. Въпреки това, цялостната нужда от памет за динамично конструирания индекс на блок в ИПЕО е по-малка от тази на БИБС.

Когато паметта свърши, записваме индекса на блока (който съдържа речника и списъка със срещанията) върху диска (ред 12). Трябва да сортираме термините (ред 11) преди да направим това, защото искаме да запишем списъците със срещания в лексикографски ред, за да улесним последната стъпка по сливане. Ако списъка със срещанията на всеки блок беше записван в несортиран ред, сливането нямаше да може да се осъществи с просто линейно обхождане на всеки блок.

Всяко извикване на SPIMI-INVERT записва блок върху диска, точно както прави и БИБС. Последната стъпка на ИПЕО (не е показано в псевдокода) следователно е да слее блоковете в крайния обърнат индекс.

В допълнение на конструирането на нова речникова структура за всеки блок и елиминирането на тромавата стъпка по сортиране, ИПЕО има и трети важен компонент: компресия. И срещанията и речника могат да бъдат разположени сбито върху диска, ако използваме компресия. Компресията увеличава ефективността на алгоритъма още повече, защото можем да обработваме още по-големи блокове, и понеже отделните блокове изискват по-малко място на диска. Времевата сложност на ИПЕО е $\Theta(T)$, защото не е нужно сортиране на думи и всички операции са в най-много линейна зависимост от размера на колекцията.

4.4 Разпределено индексиране.

Често колекциите, които трябва да индексираме са толкова големи, че не може да се извърши ефективно построяване на индекс от една физическа машина. Това е особено вярно за World Wide Web, където са ни необходими големи компютърни кълстери за създаването на индексирането. Именно затова, уеб търсачките използват алгоритми за разпределено изграждане на индекс, резултатът от които е индекс разделен между няколко машини, според термините или според документа. Повечето големи търсачки използват „document-partitioned index“, който лесно се генерира от „term-partitioned index“.

Разпределеното индексиране използва MapReduce архитектурата за разпределено изчисляване от големи кълстери. Целта на всеки един кълстер е да пресметне големи изчисления чрез неспециализирани машини (изградени от процесор, оперативна памет, и диск), за разлика от супер компютрите (изградени от специализиран хардуер). В такива кълстери се използват стотици или хиляди машини, които обаче могат да претърпят срив във всеки един момент. Именно затова, работата по разпределеното индексиране се разделя на парчета, които се изпращат за изчисление и в случай на срив – изпращат отново за преизчисление. Управлението на процесът по разпределението на порциите работа се възлага на master node.

Фазите “**map**” и “**reduce**” на **MapReduce** архитектурата разделят работата на порции, които се обработват бързо от крайните станции.

Да разгледаме случай, при който имаме колекция на уеб страници. Първо информацията се разделя на **порции**, всяка от които е толкова голяма, че да може лесно да бъде доставена за обработка (тоест на са големи като размер), а **n** е такова, че общият брой на порциите да не е твърде голям. Порциите не са предварително зададени на машините, а “**master node**” разпределя порциите в процеса на работа. Когато една машина е завършила работата по дадена порция, на нея ѝ се изпраща следваща порция. Ако машина стане недостъпна или се забави, порцията се преназначава на друга машина за обработка.

Като цяло MapReduce разделя работа на отделни порции, които представляват двойки ключ-стойност. За индексиране, ключ-стойност двойката има вида (**termID, docID**). При разпределеното индексиране, свързването на термин с termID е също разпределено индексиране, което прави индексирането по-сложно отколкото, ако беше на една отделна машина.

„**Map**“фазата се състои в свързването на порциите, на които е разделена информацията в началото на ключ-стойност двойки. Това е същата задача която се изпълнява от BSBI и SPIMI, и затова сенарича “**parsers**”. Всеки парсър записва изходната информация в локални „**segment files**“.

При „**Reduce**“фазата искаме всички двойки за даден ключ да са запазени близо, за да могат да бъдат прочетени и предадени бързо. Тази задача се изпълнява от инвертори, като ключовете се разделят на **j** порции от термини, а парсърите разписват всички ключ-стойност двойки за всяка една от порциите термини в отделен сегмент файл. Броя на сегмент файловете е равен на броя на парсърите за всяка една от порциите. Всяка една от порциите е обработена от един инвертор.

Всяка една машина може да бъде парсър, инвертор, да кроува или да изпълнява други задачи. За да се намали времето за запис преди инверторите да намалят информацията, всеки един парсър записва свои сегмент файлове, които се прочитат само веднъж при комуникация с “master node”.

Входа и изхода на MapReduce обикновено са списъци на ключ-стойност, за да могат няколко MapReduce задачи да се изпълняват едновременно. Такъв е бил и дизайнът на Google индексирането през 2004. MapReduce предоставя устойчива и като цяло проста платформа за имплементиране на разпределено индексиране. Разделянето на информацията на по-малки порции, прави архитектурата приложима както при малки, така и при големи задачи.

4.5 Динамично индексиране

Досега приемахме, че документът е статична колекция. Това е приложимо за колекции, които се променят рядко или никога, но повечето колекции се променят често, което означава, че нови термини трябва да се добавя към речника, а постинг списъци трябва да бъдат актуализирани с оглед съществуващите термини.

Най-простият начин за постигане на тази цел е периодично да се реконструира индекс от нулата. Това е добро решение, ако броят на промените с течение на времето е малък, забавяне при добавянето на новите документи към търсенето е приемливо и има достатъчно ресурси за изграждане на нов индекс, докато старият е все още на разположение за заявки.

Ако има изискване новите документи да бъдат включени бързо, едно решение е да се поддържа два показателя: голям основен индекс и малък *спомагателен индекс* за новите документи, като помощният индекс се запазва в паметта. Търсенията се изпълняват и в двата индекса, като резултатите се обединяват. Изтритванията се съхраняват в отделен двоичен вектор. След това филтрираме изтрити документи, преди да се върне резултатът от търсенето.

Всеки път, когато спомагателния индекс стане твърде голям, го сливаме с основния индекс. Сложността на сливането зависи от начина, по който се съхранява индекса във файловата система. Ако съхраняваме всеки постинг списък като отделен файл, сливането се състои от разширяване на всеки постинг списък основния индекс със съответстващияму списък от спомагателния индекс. В тази схема, причината за поддържане на спомагателните индекса е да се намали броя на търсенията. При използване на спомагателен индекс допълнително натоварване има само когато се сливат спомагателния и основния индекс.

За съжаление, схемата „one-file-per-postings-list“ е неосъществима, защото повечето файлови системи, не могат ефективно да се справят с много голям брой файлове. Най-простият вариант е да съхраним индекса като един голям файл, което представлява конкатенация на всички списъци на публикации.

Използването на множество индекси усложнява поддръжката на статистики върху колекциите. С множество индекси и инвалидиращ двоичен вектор, откриването на всички термини не е побавно от обикновено търсене. Всички аспекти на системите за извлечение на информация (поддържане на индекс, обработване на заявки, разпространение и т.н.) са по-сложни при логаритмично сливане.

Заради сложността на динамичното индексиране, някои търсачки използват „reconstruction-from-scratch“ стратегията. Вместо динамично да конструират индекс, такъв бива съставян от нулата през определено време, след което обработването се прехвърля от стария към новия индекс и стария бива изтритан.

5 Компресиране на индекс (Index compression)

В тази глава ще разгледаме няколко техники на компресиране на речници и обърнати списъци (лист от референции към документ за всяка дума), които са много важни за ефективна система за извличане на информация.

Една от ползите на компресирането е напълно ясна - нуждата от повече свободно дисково пространство. Ще видим, че компресия 1:4 лесно се постига.

Има още две предимства от компресирането. Първият плюс е по-голямото използване на caching (складиране). Системите за търсене използват някои части на речника и на листа по-често от други. Чрез компресия ние можем да поставим много от информацията в паметта на компютъра. Вместо да се налага да обхождаме целия хард диск, можем да достъпим така наречения постинг лист от паметта и да го декомпресираме. Както ще видим по-надолу има бързи начини за декомпресия с които не губим прекалено много време. В резултат на това можем да намалим драстично времето за отговор на нашата система за извличане на информация.

Вторият плюс на компресията е по-бързото предаване на информация от диска към паметта. Ефективни алгоритми на декомпресия работят толкова бързо на съвременен хардуер, че да се прехвърли компресирана информация от хард диск и да се декомпресира става по-бързо отколкото да се прехвърли в некомпресиран вариант.

Основната цел на компресирането е да се пести място. На второ място и не толкова важна е скоростта на самите алгоритми.

Тази глава дава статистическа характеристика на разпределението на нещата които искаме да компресираме-термини и постинг в големи колекции. След това разглеждаме компресирането на речника, използвайки метода „речник като string“ и blocked storage. Разглеждаме и две техники за компресия на постинг файловете, кодиране на променливите по байтове и гама кодиране.

5.1 Статистически свойства на термините в извличането на информация

Нека дадем малко информация за термините и постинг статистиките в следващата таблица. “Δ%” отбележва намалението на размера от предния ред. “Г%” е натрупващата се редукция от нефильтрирана.

Таблицата показва термините в различните моменти на действие. Броя термини е това което се гледа при определяне размера на речника. Броя на непозиционни постинги определя очаквания размер на непозиционния индекс на колекцията. Очаквания размер на позиционния индекс определя броя позиции, които трябва да се кодират.

Като изключим производните(case), намаляваме броя на различните термини с 17% и броя на непозиционни постинги с 4%. Обработката на най-често срещаната дума е много важна. Чрез техника за елиминиране на най-често срещаните думи и ползвайки ги като стоп думи можем да намалим с 25%-30% броя на непозиционните постинги.

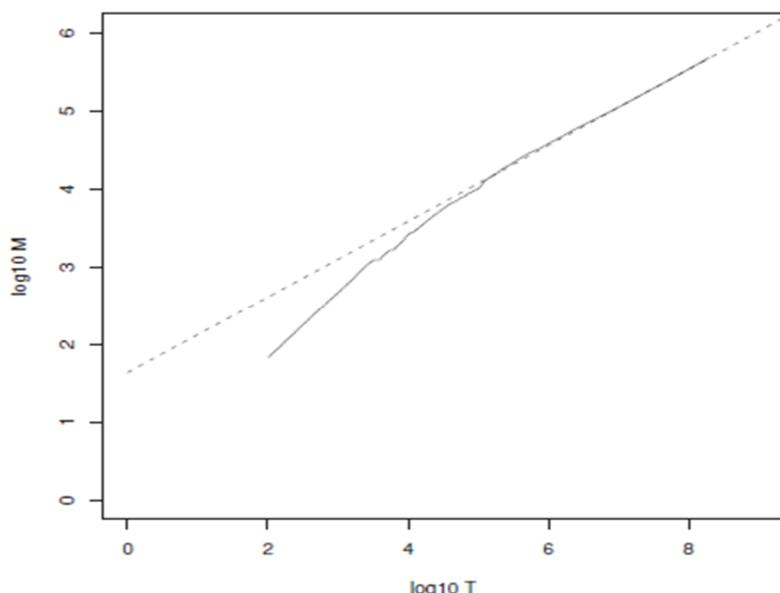
	(distinct) terms			nonpositional postings			tokens (= number of position entries in postings)		
	number	$\Delta\%$	T%	number	$\Delta\%$	T%	number	$\Delta\%$	T%
unfiltered	484,494			109,971,179			197,879,290		
no numbers	473,723	-2	-2	100,680,242	-8	-8	179,158,204	-9	-9
case folding	391,523	-17	-19	96,969,056	-3	-12	179,158,204	-0	-9
30 stop words	391,493	-0	-19	83,390,443	-14	-24	121,857,825	-31	-38
150 stop words	391,373	-0	-19	67,001,847	-30	-39	94,516,599	-47	-52
stemming	322,383	-17	-33	63,812,300	-4	-42	94,516,599	-0	-52

Фигура 5.1

- “ $\Delta\%$ ” отбележва намаляването в размера на следващия ред освен в случаите при 30 стоп думи и 150 стоп думи които ползват „case folding” за справка
- “T%” е общото(натрупващото) намаляване от нефильтрирани

Техниките за компресия които описваме се наричат *lossless*- при тях не се губи информация, Подобри компресии могат да се постигнат с така наречените *lossy* компресии, но при тях част от информацията се губи. Тези компресии са ефективни, когато информацията, която се губи е малко вероятно да е нужна в системата за търсене.

Размера на граматиката M като функция на размера на колекцията T е показана на фигурата долу.



Фигура 5.2

5.1.1 Закон на Heaps за определяне броя на термините

$$M = kT^b$$

M е размерът на лексиката. T е броят на знаците в колекцията, а „ k “ и „ b “ са коефициенти обикновено $30 \leq k \leq 100$ и $k = 0.5$.

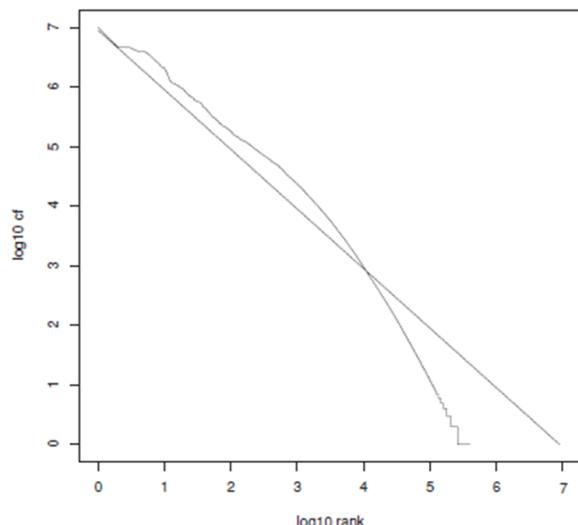
Законът на Heaps показва, че размера на речника се увеличава с увеличаването на броя на документите в колекцията, вместо да бъде достигнат максимален размер на речника в даден момент - размера на речника е сравнително голям за големи колекции.

5.1.2 Закон на Zipf за разпределението на термините

Ако t_1 е най-често срещания термин в колекцията, t_2 е следващия по честота на срещане и така нататък, тогава честотата на колекцията c_f^i на i -тия най-срещан елемент е пропорционално на $1/i$:

$$c_f^i \propto \frac{1}{i}.$$

Ето как изглежда честотата на колекцията от термини като функция на своя ранг за Reuters-RCV1



Фигура 5.3

5.2 Компресиране на речник

Тук ще представим няколко структури от текст наподобяващи речници, при които се постигат високи нива на компресия. Като цяло речниците не заемат много място, но ние ги компресираме с цел да ги поберем в паметта на компютъра и при нужда системата за извличане на информация да ги достъпи по-бързо.

На фигурата долу показваме съхранение на речник като масив от входове с фиксирана ширлина.

term	document frequency	pointer to postings list
a	656,265	→
aachen	65	→
...
zulu	221	→
space needed:		20 bytes 4 bytes 4 bytes

Фигура 5.4

Въпреки че речници от много големи колекции се побират в паметта на стандартен десктоп компютър, това не е вярно при други приложения. Например сървър за търсене на голяма компания, който трябва да достъпва огромна колекция от документи на различни езици. Основна наша цел е да проектираме и системи за търсене за по-ограничен хардуер като мобилни телефони и настолни компютри. Друга причина поради която искаме да пестим памет е по-бързото стартиране на машината и нуждата да се споделят ресурси с други приложения.

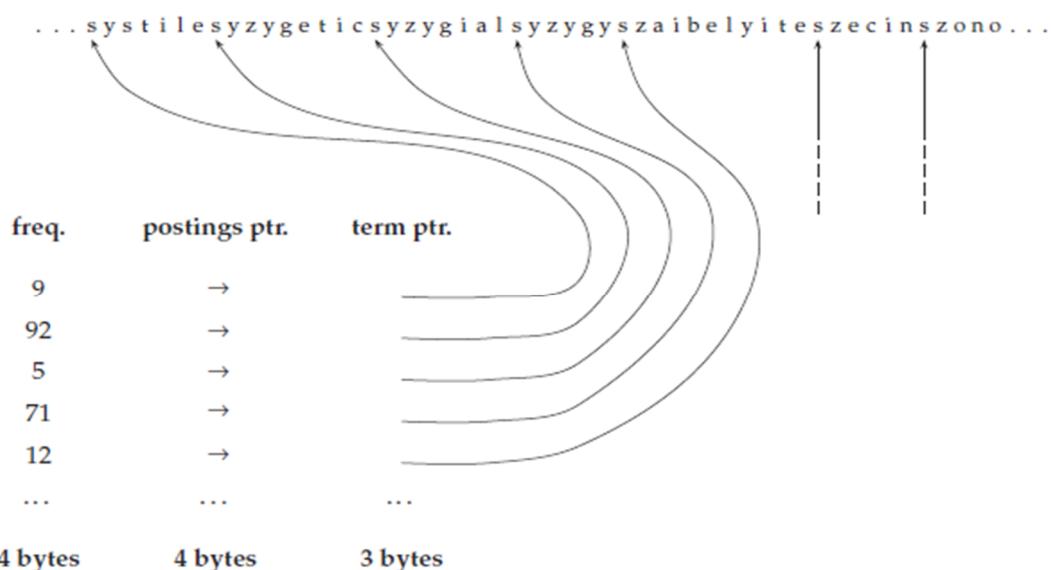
5.2.1 Речника като низ (string)

Най-простата структура от данни за един речник е да се сортира граматиката лексикографски и да се съхрани в масив с фиксирани по ширина входни данни(както показвахме в таблицата по горе). Заделяме 20 байта за самия термин (защото много малко думи се състоят от повече от 20 знака), 4 байта за честотата в документа и 4 байта за пойнтъра към постинг листа. Пойнтъри от 4 байта заемат 4 гигабайта място. За Reuters-RCV1 ние се нуждаем от $M \times (20 + 4 + 4) = 400,000 \times 28 =$

11.2 megabytes (MB) за съхраняване на речника.

Използването на фиксирани широки входове за термини е прахосничество, защото средно думите имат по 8 знака. В същото време обаче ни се налага да съхраняваме термини с повече от 20 знака, което става невъзможно. Можем да се справим с този проблем като запазваме термините в речника като един голям низ(string) от знаци, както е показано по долу .

По тази нова схема ние се нуждаем от $400,000 \times (4 + 4 + 3 + 8) = 7.6 \text{ MB}$ заReuters-RCV1 речник.



Фигура 5.5

Това се нарича „Dictionary-as-a-string storage”. Стрелките показват края на предпоставия термин и началото на следващия. Например първите три термина в този пример са systyle, syzygetic, и syzygial. Тази метод ни съхранява до 60% място в сравнение когато дължината на термина ни е фиксирана.

5.2.2 Съхранение в блокове

Речникът може допълнително да се компресира, като се групират термините в групи с размер k и се пази поинтър само към първия термин от групата. Дължината на всеки термин се пази в самия стринг и се поставя пред него ([Фигура 5.6](#)). Така се елиминират $k - 1$ term pointer-а, но се добавят още k байта за дължина на термин. При $k = 4$ спестяваме $(k-1) \times 3 = 9$ байта от term pointer-и, но добавяме $k = 4$ байта за дължина на термин. Общо печелим 5 байта на всеки блок, състоящ се от 4 термина. За речника Reuters-RCV1 получаваме $400\,000 \times \frac{1}{4} \times 5 = 0.5$ MB, то ест вече е с размер 7,1 MB.

... 7 systyle 9 syzygetic 8 syzygial 6 syzygy11saibelyite6szecin...

Фигура 5.6

Увеличавайки размера на блока k, получаваме по-добра компресия, обаче съществува компромис между компресия и скорост на търсене. В един некомпресиран речник търсим с binary search, а в един компресиран речник първо намираме блока, в който се намира терминът, с binary search, а позицията му в блока – чрез линейно търсене в блока. Търсенето в некомпресиран речник отнема средно 1.6 стъпки при положение, че всички термини имат еднаква вероятност да бъдат потърсени. С блокове от размер k = 4 ни трябват средно 2 стъпки или $\approx 25\%$ повече. Увеличавайки k, можем да достигнем до минималната големина на речника 6.8 MB, но търсенето на термини става прекалено бавно при големи стойности на k.

Друго излишество в речника, от което не сме се възползвали, е фактът, че последователни елементи в списък, който е подреден по азбучен ред, имат обща представка. Това наблюдение

води до *front coding*. Намира се обща представка за дадена последователност и се отбелязва със специален символ ([Фигура 5.7](#)). В случая с Reuters front coding спестява още 1.2 MB.

Един блок при блокова компресия ($k = 4$) . . .

8automata8automate9automatic10automation

↓

. . . допълнително се компресира с front coding.

8automat*a1◦e2◦ic3◦ion

Фигура 5.7 Front coding. Последователност от термине с обща представка („automat“) се компресира, като се отбележи края на представката с * и се замени с ◦ в следващите термини.

Други методи за още по-голяма компресия се базират на минимално перфектно хеширане, то ест хеш функция мапва M на брой термина върху [1, ..., M] без колизии. Този метод обаче не може да се използва инкрементално, защото всеки нов термин ще предизвика колизия и следователно се налага да се създаде нова перфектна хеш функция. Така че тези методи не могат да се използват в динамични среди.

Дори с най-добрите методи за компресия може да се окаже, че е невъзможно да се съхранят целият речник в главната памет, ако боравим с огромни текстови колекции или с хардуер с ограничена памет. Ако ни се наложи да разделим речника на страници, съхранявани върху диск, тогава можем да индексираме първият термин от всяка страница, използвайки В-дърво.

5.3 Компресия на postings file

Да си припомним, че Reuters-RCV1 има 800 000 документа, 200 токена на документ, шест символа на токен и 100 000 000 постинга, като ние дефинираме постинг в тази глава като docID в постинг лист, с други думи изключваме честотата и информацията за позицията. Идентификаторите на документите са дълги $\log_2 800\,000 \approx 20$ бита. По този начин размерът на колекцията е около $800\,000 \times 200 \times 6$ байта = 960 MB, а размерът на некомпресирания постинг файл е $100\,000\,000 \times 20/8 = 250$ MB.

За по-ефикасна репрезентация на постинг файла, използваща по-малко от 20 бита за docID, вместо docID записваме през колко документа (гаповете между тях) се среща даден термин. Тъй като често срещаните термини се срещат в много документи, то гаповете (интервалите) са малки числа, за които са нужни по-малко от 20 бита. Проблем остават рядко срещаните термини, които се срещат в малко документи и гаповете са от същия порядък като docID-тата. За това ни е нужно кодиране с променлива дължина.

За тази цел ще разгледаме два метода: байтова компресия и битова компресия. Те се опитват да кодират гаповете с възможно най-малко байта/бита.

5.3.1 Variable byte codes

Variable byte (VB) encoding използва интегрално количество байтове, за да кодира гап. Последните 7 бита от всеки байт кодират част от гапа. Първият бит е бит за продължение. На последния байт е сетнат на 1, на останалите е 0. За да декодираме такъв код, прочитаме

последователност от байтове, докато стигнем такъв с първи бит 1. Вземаме последните 7 бита от всички прочетени байтове и ги конкатенираме (Фигура 5.8).

docID-та	824	829	215406
гапове		5	214577
VB code	00000110 10111000	10000101	00001101 00001100 10110001

Фигура 5.8

С VB компресия размерът на индекса за Reuters-RCV1 е 116 MB, с повече от 50% по-малко от некомпресирания индекс.

Идеята на VB кодирането може да се приложи върху по-големи или по-малки групи от байтове: 32-битови думи, 16-битови думи и 4-битови думи (nibbles). Колкото по-дълга е думата, толкова по-малко работа с байтове е нужна, но за сметка на компресията. И обратното – с по-къси думи се получава по-добра компресия, но се налага повече работа с байтове.

За повечето системи за извлечение на информация байтовете кодове предоставят отличен компромис между време и памет, те са лесни за имплементиране за разлика от γ кодовете и δ кодовете, които се използват при осъдна памет.

5.3.2 γ кодиране

Кодирането на битово ниво е по-фин начин за кодиране на гаповете. Най-простият код на битово ниво е унарният код. Унарен код от n представлява n единици и една 0 в края.

При условие, че всички 2^n гапове са еднакво вероятни, то тогава оптималният код използва поне $\log_2 2^n$ бита.

Γ кодирането е метод, който се намира на коефициент от оптималното кодиране. Γ кодът е с променлива дължина, той разделя гапа G на две части – *дължина* и *офсет*. Офсетът е G в двоична бройна система, като водещата единица е премахната. Например офсетът на 13 (1101) е 101. *Дължината* е дължината на офсета в унарен код. На 13 дължината на офсета е 3 бита, което е 1110 в унарен код. Като конкатенираме дължината и офсета получаваме γ кода на 13 – 1110101.

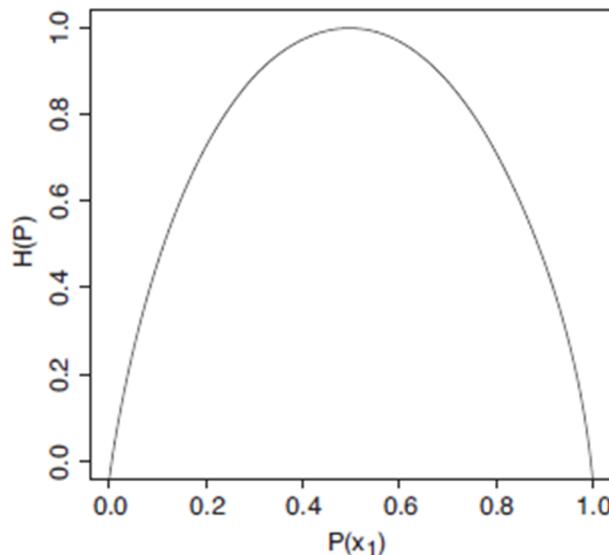
За да разкодираме γ код, първо прочитаме унарния код до нулата, това ни дава дължината на офсета, прочитаме и него и добавяме една 1 отпред.

Дължината на офсета е $\lceil \log_2 G \rceil$ бита, а дължината на *дължината* е $\lceil \log_2 G \rceil + 1$ бита, което общо дава $2 \times \lceil \log_2 G \rceil + 1$ бита. Γ кодовете винаги са с нечетна дължина и са с коефициент 2 по-големи от оптималната дължина $\lceil \log_2 G \rceil$. Този оптимум е при условие, че всички гапове са еднакво вероятни, но това не винаги е така. В общи случаи няма как предварително да знаем вероятностното разпределение на гаповете.

Характеристиката на едно дискретно вероятностно разпределение P , която определя неговите кодиращи свойства (включително дали е кодът е оптимален) е неговата *ентропия* $H(P)$, която се дефинира така:

$$H(P) = \sum_{x \in X} P(x) \log_2 P(x)$$

където X е множеството от всички възможни числа, които трябва да можем да кодираме. Ентропията е мярка за несигурност. При $X = \{x_1, x_2\}$ ентропията се максимира ($H(P) = 1$) за $P(x_1) = P(x_2) = 0.5$, когато несигурността кое x_i ще се случи е най-голяма, и се минимизира ($H(P) = 0$) при $P(x_1) = 1, P(x_2) = 0$ и $P(x_1) = 0, P(x_2) = 1$, когато има абсолютна сигурност ([Фигура 5.9](#)).



Фигура 5.9

Може да се покаже, че долната граница на очакваната дължина $E(L)$ на код L е $H(P)$, ако са изпълнени определени условия. Може още да се покаже, че за $1 < H(P) < \infty$, γ кодирането е на коефициент 3 от оптималното кодиране и се доближава до 2 за големи $H(P)$:

$$\frac{E(L_\gamma)}{H(P)} \leq 2 + \frac{1}{H(P)} \leq 3$$

Забележителното на този резултат е, че той важи за всяко вероятностно разпределение Р. Т. е. γ кодирането винаги се намира на коефициент ≈ 2 от оптимално кодиране за разпределения с голяма ентропия. Кодове, които се намират на коефициент от оптималния код за произволно разпределение се наричат универсални кодове.

Освен че γ кодирането е универсално, то притежава още две свойства, които са полезни при индекс компресията. Г кодовете са *prefix free*, което означава, че никой γ код не е префикс на друг γ код и винаги съществува уникално декодиране на поредица от γ кодове, затова не ни трябват разделители между тях, които биха намалили ефикасността на кода. Второто свойство е, че γ кодовете са *parameter free*. При много други ефикасни кодове трябва да съхраняваме в паметта параметрите на разпределението на гаповете в индекса. Това усложнява имплементацията на компресията и декомпресията. При динамичното индексиране разпределението на гаповете може да се промени и първоначалните параметри да не са вече валидни. Тези проблеми са избегнати при parameter-free кодовете.

6 Оценяване, претегляне на термини и векторно-пространствен модел

При индексите, които поддържат булеви заявки, даден документ или съвпада или не съвпада със заявката. При голям набор от документи, броя на съвпадащите може да надхвърли количеството от документи, което потребителя може да прегледа. Съответно, е важно за една търсачка, да подрежда съвпадащите документи според някаква оценка. Тази глава се състои от три основни идеи.

1. Представяме параметрични и зонални индекси в раздел 6.1, които служат за две цели. Първо ни позволяват да индексираме и извличаме документи чрез метаданни. И второ, ни предоставят средства за оценяване (и следователно подреждане) на документи отговарящи на заявката.
2. В раздел 6.2, развиваме идеята за претегляне важността на даден термин (дума) в документ, според статистиката на появяванията му.
3. В раздел 6.3, разглеждаме всеки документ като вектор от тегла. Така можем да изчислим оценка за заявката спрямо всеки документ. Този преглед е познат като оценяване на векторното пространство.

В раздел 6.4 разглеждаме няколко варианта за претегляне на термин за векторно пространствения модел.

6.1 Параметрични и зонални индекси

До сега разглеждахме документа като последователност от термини. Всъщност повечето документи съдържат определени метаданни. Метаданните са специален вид информация за документа. Те съдържат основно полета, като дата на създаване и формат на документа, автора и може би заглавието.

Разглеждайки заявки от типа „открий документ с автор Уилям Шекспир, от 1601, съдържащ фразата „alas poor Yorick”, обработването на заявката се състои, както обикновено, от пресичане на срещания (постинги), само че можем да обединим срещания от стандартно инвертиране, както и параметрични индекси. Има по един параметричен индекс за всяко поле (да кажем дата на създаване), което ни позволява да избираме само документите, отговарящи на датата указана в заявката. Фигура 6.1 показва потребителския изглед на такова параметрично търсене. Някои от полетата могат да приемат подлежащи на подреждане стойности, като дати. В дадения по-горе пример 1601 година е такава стойност. Търсачката може да поддържа диапазон на заявките от такива подредими стойности. Структура подобна на В-дърво (B-tree) може да бъде използвана за речника на полетата.

Зоните са подобни на полетата, с тази разлика, че съдържанието на зоната може да бъде произволен текст. Докато полето може да има относително малък набор от стойности, зоната може да бъде смятана за произволна, необвързана с количество текст. Например заглавията и резюметата на документ по принцип се смятат за зони. Можем да построим отделен обърнат индекс за всяка зона от документа, за да поддържаме заявки от вида на „открий документи с

търговец в заглавието и Уилям в списъка от автори, както и фразата нежен дъжд в текста". Това построява индекс, който изглежда като фигура 6.2. Докато речникът от параметрични индекси идва от определена лексика (набор от езици или набор от дати), речникът за зонални индекси трябва да използва лексика произтичаща от текста на тази зона. Всъщност, можем да намалим размера на речника, като кодираме зоната, в която термините се появяват. На фигура 6.3 например, показваме как появяванията на Уилям в заглавните и авторни зони от различни документи са кодирани. Такова кодиране е полезно, когато размера на документа е проблем. Но има и друга важна причина - кодирането от фигура 6.3 е полезно за ефективното пресмятане на оценка използвайки техника която ще наречем *оценяване чрез претеглени зони*.

Bibliographic Search

Search category	Value
Author	Example: Widom, J or Garcia-Molina
Title	Also a part of the title possible
Date of publication	Example: 1997 or <1997 or >1997 limits the search to the documents appeared in, before and after 1997 respectively
Language	Language the document was written in English
Project	ANY
Type	ANY
Subject group	ANY
Sorted by	Date of publication
<input type="button" value="Start bibliographic search"/>	
<input type="button" value="Find document via ID"/> <input type="text"/>	

Фигура 6.1 Параметрично търсене. В този пример имаме набор от полета позволяващи ни да изберем публикации по зони като автор и полета като език

Уилям.резюме	11	121	1441	1729
--------------	----	-----	------	------

Уилям.заглавие	2	4	8	16
----------------	---	---	---	----

Уилям.автор	2	3	5	8
-------------	---	---	---	---

Фигура 6.2 Основен зонален индекс; зоните са кодирани като разширения на речникови вписвания.

Уилям	2.автор, 2.заглавие	3.автор	4.заглавие	5.автор
-------	---------------------	---------	------------	---------

Фигура 6.3 Зонален индекс, в който зоната е кодирана в постовете вместо в речника.

6.1.1 Оценяване чрез претеглени зони

До сега в раздел 6.1 се бяхме съсредоточили върху извлечане на документи основано на булеви заявки на полета и зони. Сега преминаваме към второто приложение на зоните и полетата. При дадена булева заявка q и документ d , оценяването на претеглената зона задава на двойката (q, d) оценка в интервала $[0, 1]$, чрез изчисляване на линейна комбинация от *зонови оценки*, където всяка зона на документа добавя булева стойност. По-конкретно взимаме набор от документи, всеки от които има l на брой зони. Нека $g_1, \dots, g_l \in [0, 1]$, така че $\sum_{i=1}^l g_i = 1$. За $1 \leq i \leq l$, нека s_i бъде булева оценка обозначаваща съвпадение (или липса на такова) между q и i -тата зона. Например, булевата оценка от зона може да бъде 1, ако всеки термин от заявката присъства в тази зона и 0 в противен случай; т.е. може да бъде всяка булева функция, която определя присъствието на термина от заявката в зона на 0 и 1. Тогава оценяването на претеглена зона е дефинирано както следва

$$(6.1) \quad \sum_{i=1}^l g_i s_i$$

Оценяването на претеглена зона понякога се нарича също и *ранкирано булево извлечане*.

Пример 6.1: Да вземем заявката Шекспир в колекция, в която всеки документ има три зони: автор, заглавие и съдържание (текст). Булевата оценка на зоната приема стойност 1, ако термина от заявката Шекспир присъства в зоната и 0 в противен случай. Оценяването на претеглена зона в такъв набор изисква три променливи g_1 , g_2 и g_3 отговарящи съответно на зоните автор, заглавие и съдържание. Да предположим, че зададем $g_1 = 0.2$, $g_2 = 0.3$, и $g_3 = 0.5$ (така че сумата на трите променливи да е 1). При този вариант, съвпадение в зоната автор е най-малко важно в общото оценяване, в зоната за заглавие е малко по-важно и в съдържанието има най-висока стойност. По този начин, ако термина Шекспир се появява в заглавието и в съдържанието, но не и в авторовата зона на документ, неговата оценка ще бъде 0.8.

Как пресмятаме оценката на претеглена зона? Прост подход би бил да изчислим резултата на всеки документ, като добавим и приносът на всяка зона. Въпреки това, сега ще покажем как можем да определим оценяването на претеглена зона директно от обрънати индекси. Алгоритъмът от фигура 6.4 се отнася за случая, когато заявката q се състои от два термина q_1 и q_2 и булевата функция е логическо И – 1, ако и двата термина се срещат в зоната, и 0 в противен случай. След описанието на алгоритъма ще преминем към по-сложни заявки и Булеви функции.

ZONESCORE(q_1, q_2)

```

1 float scores[N] = [0]
2 constant g[l]
3 p1 ← postings(q1)
4 p2 ← postings(q2)
5 // scores[] е масив с входна оценка за всеки документ започвайки с нула.
6 // p1 и p2 са въведени да посочват началната точка на техните съответни срещания.
7 // Да предположим, че g[] се инициализира до съответните зонови стойности.
8 while p1 != NIL and p2 != NIL
9 do if docID(p1) = docID(p2)
10 then scores[docID(p1)] ← WEIGHTEDZONE(p1, p2, g)
11 p1 ← next(p1)
12 p2 ← next(p2)

```

```

13else if  $docID(p1) < docID(p2)$ 
14then  $p1 \leftarrow next(p1)$ 
15else  $p2 \leftarrow next(p2)$ 
16return scores

```

Фигура 6.4 Алгоритъм за изчисляване на оценката чрез претеглени зони от дву-постов списък. Функцията WEIGHTEDZONE (не е показана тук) се приема, че изчислява вътрешния цикъл на уравнението 6.1.

Масива scores[], може да се разглежда като съвкупност от *акумулатори*. Причината за това ще се изясни, когато разглеждаме по сложни булеви функции от логическо И. По този начин, можем да зададем не-нулева оценка на документ, дори и ако не съдържа всички термини от заявката.

6.1.2 Определяне на теглата

Как определяме стойностите g за оценяването чрез претеглена зона? Тези стойности могат да бъдат определени от специалист, но все по-често, тези стойности се определят като се използва машинно самообучение и се нарича *машинно научено съответствие*.

1. Разполагаме с набор от тренировъчни примери, всеки от които се състои от двойката - заявка q и документ d , заедно с оценка на съответствието на d спрямо q . В най-проста форма, оценката на съответствието е или „Съответства“ или „Не съответства“. Посложни реализации на методологията, използват по-разнообразни оценки.
2. Стойностите g са „научени“ от тези примери, така че научените оценки да се доближават до оценката на съответствие в тренировъчните примери.

За оценяването на претеглени зони, процесът може да бъде разглеждан като научаване на линейна функция от оценката на булевите съвпадения, като се добави приноса на различните зони. Същият компонент на тази методология е трудното построяване на генериирани от потребителя оценки на съответствие, от които да се обучат стойностите. Сега ще разглеждаме подробно пример, който описва как можем да сведем проблема на самообучението на стойностите g , до прост оптимизационен проблем.

Сега ще разгледам прост случай на оценяване чрез претеглените зони, където всеки документ има зона заглавие и зона съдържание. При дадени заявка q и документ d , използваме дадената булева функция за съвпадение, за изчисляване на булеви променливи $s_T(d, q)$ (за заглавието) и $s_B(d, q)$ (за съдържанието), зависещи от това, дали зоната на заглавието (респективно съдържанието) отговаря на заявката q . Например, алгоритъма на фигура 6.4 използва логическо И върху термините от заявката за тази булева функция. Ще изчислим оценка между 0 и 1 за всяка двойка (документ, заявка) използвайки $s_T(d, q)$ и $s_B(d, q)$, като използваме константа $g \in [0, 1]$, както следва:

$$(6.2) \quad \text{оценка}(d, q) = g \cdot s_T(d, q) + (1-g) \cdot s_B(d, q).$$

Сега ще опишем, как се определя константата g от набор от тренировъчни примери, всеки от които е тройка под формата $\phi_j = (d_j, q_j, r(d_j, q_j))$. Във всеки тренировъчен пример, даден тренировъчен документ d_j и дадена тренировъчна заявка q_j са оценени от човешки редактор, който поставя оценка на съответствие $r(d_j, q_j)$, която е или „Съответства“, или „Не съответства“. На фигура 6.5 са показани седем тренировъчни примера. За всеки тренировъчен пример ϕ_j имаме булеви стойности $s_T(d_j, q_j)$ и $s_B(d_j, q_j)$, които използваме, за да изчислим оценка от (6.2)

$$(6.3) \text{оценка}(d_i, q_j) = g \cdot s_T(d_i, q_j) + (1-g) s_B(d_i, q_j).$$

Сега сравняваме тази изчислена оценка с човешката оценка на съответствие за същата двойка документ-заявка (d_i, q_j) . Ще смятаме всяка оценка за „Съответствие“ за 1 и всяка оценка за „Не съответствие“ за 0. Ако дефинираме грепката от оценяващата функция със стойност g , като $\varepsilon(g, \Phi_j) = (r(d_i, q_j) - \text{оценка}(d_i, q_j))^2$,

Пример	Номер на документа	Заявка	s_T	s_B	Проценка
Φ_1	37	Linux	1	1	Съответства
Φ_2	37	Penguin	0	1	Не съответства
Φ_3	238	System	0	1	Съответства
Φ_4	238	Penguin	0	0	Не съответства
Φ_5	1741	Kernel	1	1	Съответства
Φ_6	2094	Driver	0	1	Съответства
Φ_7	3191	Driver	1	0	Не съответства

Фигура 6.5 Илюстрация на тренировъчни примери.

s_T	s_B		Оценка
0	0		0
0	1		$1 - g$
1	0		g
1	1		1

Фигура 6.6 Четирите възможни комбинации на s_T и s_B , където сме квантували редакторната оценка за съответствието (d_i, q_j) до 0 или 1. Тогава общата грепка на набор от тренировъчни примери се определя от

$$(6.4) \quad \sum_j \varepsilon(g, \Phi_j).$$

Проблемът на научаването на константата g от дадените тренировъчни примери, се свежда до подбирането на стойността на g , която минимизира крайната грепка в (6.4).

Избирането на най-добрата стойност за g в (6.4) в раздел 6.1.3 се свежда до минимизирането на квадратичната функция на g в интервала $[0,1]$.

6.1.3 Оптималната тежест g

Нека отбележим, че за всеки тренировъчен пример Φ_j , за който $s_T(d_i, q_j) = 0$ и $s_B(d_i, q_j) = 1$, оценката изчислена чрез уравнението (6.2) е $(1-g)$. По подобен начин, можем да запишем и оценката

изчислена от (6.2) за другите три възможни комбинации на $s_T(d_j q_j)$ и $s_B(d_j q_j)$ - това е обобщено във Фигура 6.6.

Нека $n_{01r(\text{съответства})}$ (респективно $n_{01n(\text{не съответства})}$) е броя на тренировъчните примери, за които $s_T(d_j q_j) = 0$ и $s_B(d_j q_j) = 1$ и редакторната оценка е „Съответства“ (респективно „Не съответства“). Тогава приносът към крайната грешка в уравнението (6.4) от тренировъчни примери, за които $s_T(d_j q_j) = 0$ и $s_B(d_j q_j) = 1$ е

$$[1 - (1-g)]2n01r + [0 - (1-g)2]n01n.$$

По подобен начин намираме приносът към грешката от примерите на другите три комбинации от стойности за $s_T(d_j q_j)$ и $s_B(d_j q_j)$ и получаваме, че крайната грешка отговаряща на уравнението (6.4) е

$$(6.5)(n01r + n10n)g2 + (n10r + n01n)(1-g)2 + n00r + n11n.$$

Чрез диференциране на уравнението (6.5) по отношение на g и задаване на резултата да е 0, следва, че оптималната стойност на g е

$$(6.6) \frac{n10r + n01n}{n10r + n10n + n01r + n01n}$$

6.2 Честота на появяване на термин и претегляне

До сега, оценяването разчиташе на това, дали термин от заявката присъства или не в зона на документа. Предприемаме следващата логическа стъпка - документ или зона, в който присъства терминът от заявката, по-често има повече общо с тази заявка и следователно трябва да получи по-висока оценка. За да аргументираме това, си припомняме идеята за заявка със свободен текст от раздел 1.4 - заявка в която термините на заявката са написани в свободна форма в интерфейса за търсене, без никакви свързващи оператори (като булеви оператори). Този модел, който е много популярен в Интернет, разглежда заявката като прост набор от думи. Правдоподобен оценяваш механизъм тогава, е да се изчисли оценка, която е сумата от оценките на термините в документа, съвпадащи с термините от заявката.

За тази цел, добавяме за всеки термин в документа тегло, което зависи от броя на появяванията на този термин в документа. Бихме искали да изчислим оценката на термин от заявката t и документа d , въз основа на тежестта на t в d . Най-лесно би било теглото да е равно на броя на появяванията на t в документа d . Това претегляне се нарича *честота на термина* и се означава $tf_{t,d}$ (term frequency), с индекси обозначаващи термина и документа.

За документ d , набора от теглата определени чрез tf , може да бъде разгледана като количествена оценка на този документ. При документ разгледан от тази гледна точка, известна в литературата като *торба с думи*, точното подреждане на думите в документа се игнорира, но броя на появявания на всяка дума е важен. Запазваме информация само за броя на появявания на всяка дума. Така, документа „Мери е по бърза от Джон“ в този вид е идентичен с документа „Джон е по бърз от Мери“. Въпреки това, логично е, че два документа с подобни торби от думи са подобни по съдържание.

Всички думи в документ еднакво важни ли са? Явно не са; в раздел 2.2.2 разгледахме идеята за стоп думи – думи, които решихме да не индексираме въобще, и следователно да не допринасят за оценяването.

6.2.1 Обърната документова честота (idf)

Използването на горната честота на термина (tf) страда от критичен проблем - всички думи се приемат за еднакво важни, когато става въпрос за оценка на съответствието на заявка. Всъщност определени думи имат малка или пренебрежима тежест в определянето на съответствието. Например, в набор от документи на автоВАЗ е вероятно да има „авто“ в почти всеки документ. За тази цел ще въведем механизъм за намаляване значимостта на термини, които се появяват прекалено често в документите, за да имат значение за определянето на съответствието. Веднага идва идеята да се намали тежестта на термините с висока честота в колекцията от документи, определена като броя на появяванията на термин в колекцията. Идеята е да се намали tf тежестта на термина, в зависимост от честотата му в колекцията (cf).

Вместо това, по-често срещано е да се използва за тази цел *документната честота* df_t - броя на документи в колекцията, които съдържат термина t . Това е така, защото в опит да се направи разлика между документите с цел оценяване, е по-добре да се използва статистика на ниво документ (като броя на документите съдържащи термина), отколкото да се използва статистика за появяванията в цялата колекция. Причината да предпочетем документната честота (df) пред колекционна честота (cf) е показана на фигура 6.7, където прост пример показва, че cf и df са доста различни. По-специално, стойностите на cf за двете думи „try“ и „insurance“ са приблизително еднакви, но техните df стойности се различават значително. Искаме малкото документи, които съдържат „insurance“, да получат по-голям резултат за заявкa „insurance“, отколкото многото документи съдържащи „try“ получават за заявка „try“.

Дума	cf	df
try	10422	8760
insurance	10440	3997

Фигура 6.7 Наборна честота (cf) и документна честота (df) се държат по различен начин, като в този пример от колекцията на Ройтерс.

Как се използва документната честота (df) на определен термин, за да се намали теглото му? Обозначавайки крайният брой документи в колекцията с N , определяме обърнатата документова честота (idf) на термин t както следва:

$$(6.7) idf_t = \log_{10} \frac{N}{df_t}.$$

По този начин, idf на рядка дума е високо, докато idf на често срещана дума е вероятно да е ниска. Фигура 6.8 дава пример за idf в колекция на Ройтерс от 806 791 документа; в този пример логаритмите са с основа 10. Всъщност, точната основа на логаритъма не е съществена за оценяването.

6.2.2 Tf-idf оценяване

Сега комбинираме определенията за честотата на появяване на термин и обратната документова честота, за да получим комбинирана стойност за всеки термин, във всеки документ.

Термин	df_t	idf_t
car	18 165	1.65
auto	6723	2.08
insurance	19 241	1.62
best	25 235	1.5

Фигура 6.8 Пример за idf стойности. Тук сме предложили idf на термини с различни честоти в набора от 806 791 документа на Ройтерс.

Претеглящата схема на $tf\text{-}idf$, задава на термин t тежест в документа d получена от

$$(6.8) \quad tf\text{-}idf_{t,d} = tf_{t,d} \times idf_t$$

С други думи, $tf\text{-}idf_{t,d}$ задава на термина t стойност в документа d , която е:

1. най-висока, когато t се появява много пъти в малък набор от документи;
2. по-ниска, когато думата се появява по-малко пъти в документ, или се появява в много документи;
3. най-ниска, когато думата се появява на практика във всички документи.

Тогава можем да разгледаме всеки документ, като вектор с един компонент отговарящ на всяка дума в речника, заедно с тежестта за всеки компонент, която е изчислена от (6.8). За речниково термини, които не се появяват в документ, тази тежест е нула. Тази векторна форма ще се окаже от решаващо значение за оценяването - ще развием тези идеи в раздел 6.3. За начало, представяме мярката за препокрита оценка: оценката на документ d е сумата, по всички думи от заявката, на броя пъти, които всеки термин се появява в d . Можем да подобрим тази идея, като добавим вместо номера на появявания на всеки термин от заявка t в d , $tf\text{-}idf$ стойността на на всяка дума в d .

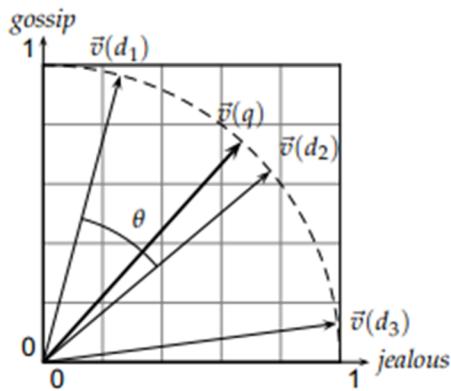
$$(6.9) \quad \text{Оценка}(q, d) = \sum_{t \in q} tf - idf_t, d.$$

6.3 Векторно-пространствен модел за оценяване

Представянето на набор от документи като вектор в общо векторно пространство, е познато като векторно-пространствен модел и е основа на набор от операции за извличане на информация, простиращи се от оценяване на документи спрямо търсене, класификация на документи и кълстериране на документи.

6.3.1 Скаларно произведение

С $V(d)$ означаваме векторът на документ d , с по един компонент във вектора за всеки термин от речника. Наборът от документи може да бъде разглеждан, като набор от вектори във векторно пространство, което има една ос за всеки термин.



Фигура 6.10 Илюстрация на Косинусова мярка за подобие (Cosine similarity) $\text{sim}(d_1, d_2) = \cos\theta$.

Как изразяваме количествено приликата между два документа в това векторно пространство? Първо можем да вземем предвид степента на векторната разлика между два документа вектори. Това измерване има два недостатъка: Два документа с много подобно съдържание могат да имат значително голяма векторна разлика единствено защото единия е много по-дълъг от другия. Следователно относителното разпределение на термините може да е еднакво в двата документа, но абсолютната честота на термините да се различава значително.

За да компенсираме влиянието на дължината на документите, стандартния начин да се представи приликата между документите d_1 и d_2 е да се изчисли косинусовата мярка за подобие на техните векторни представления $V(d_1)$ и $V(d_2)$:

$$(6.10) \quad \text{sim}(d_1, d_2) = \frac{\vec{V}(d_1) \cdot \vec{V}(d_2)}{|\vec{V}(d_1)| |\vec{V}(d_2)|},$$

където числителят представлява скаларно произведение на векторите $V(d_1)$ и $V(d_2)$, а знаменателят е умножението на Евклидовите им дължини. Скаларното произведение x_y на два

вектора се дефинира като $\sum_{i=1}^M x_i y_i$. Нека с $V(d)$ означим документ-вектора на d с M компонента $V_1(d) \dots V_M(d)$. Евклидовата дължина на d се дефинира като $\sqrt{\sum_{i=1}^M \vec{V}_i^2(d)}$.

Ефекта на числителя на уравнение (6.10) е да нормализира дължината на векторите $V(d_1)$ и $V(d_2)$ до единични вектори $\vec{v}(d_1) = \vec{V}(d_1) / |\vec{V}(d_1)|$ и $\vec{v}(d_2) = \vec{V}(d_2) / |\vec{V}(d_2)|$.

Можем да запишем уравнение (6.10) така:

$$(6.11) \quad \text{sim}(d_1, d_2) = \vec{v}(d_1) \cdot \vec{v}(d_2).$$

Разглеждането на набор от N документа, като колекция от вектори води до естествено възприемане на набора/колекцията, като матрица от документ и термини: това е матрица $M \times N$, чийто редове представляват M термини (дименции) на N колони, всяка от които отговаря на един документ.

	Doc1	Doc2	Doc3
car	0.88	0.09	0.58
auto	0.10	0.71	0
insurance	0	0.71	0.70
best	0.46	0	0.41

Фигура 6.11 Евклидови нормализирани tf стойности за документите от Фигурна 6.9

term	SaS	PaP	WH
affection	115	58	20
jealous	10	7	11
gossip	2	0	6

Фигура 6.12 Честота на термините в три романа.

term	SaS	PaP	WH
affection	0.996	0.993	0.847
jealous	0.087	0.120	0.466
gossip	0.017	0	0.254

Фигура 6.13 Вектори на термините за трите романа от Фигурна 6.12

6.3.2 Търсенето като вектор

Има и много по-силна причина да представяме документите като вектори – можем да представим и търсенето като вектор.

Ако разглеждаме търсенето като „торба с думи“, ние можем да го смятаме за много кратък документ. Следователно, можем да използваме косинусовата мярка за подобие между търсения вектор и документ-вектора, като мярка за оценката на този документ спрямо това търсене. Така оценките могат да се използват, за да се изберат документите с най-високите оценки за търсенето. Така имаме оценка

$$(6.12) \quad \text{score}(q, d) = \frac{\vec{V}(q) \cdot \vec{V}(d)}{|\vec{V}(q)| |\vec{V}(d)|}.$$

Един документ може да има висока косинусова мярка за подобие за търсене, дори и да не съдържа всички термини от търсенето.

6.3.3 Изчисляване на векторни оценки

Обикновено имаме колекция от документи, всеки представен от вектор, търсене в свободен текст представено като вектор и положително число K. Търсим K документа от колекцията, с най-високите оценки във вектор-пространството. Обикновено търсим тези K документа по низходящ ред (напр. много търсачки използват K=10, за да върнат и подредят по ранг първата страница с най-добрите резултати).

```

COSINESCORE( $q$ )
1 float Scores[N] = 0
2 Initialize Length[N]
3 for each query term  $t$ 
4 do calculate  $w_{t,q}$  and fetch postings list for  $t$ 
5 for each pair( $d, tf_{t,d}$ ) in postings list
6 do Scores[d] += wft,d × wt,q
7 Read the array Length[d]
8 for each  $d$ 
9 do Scores[d] = Scores[d]/Length[d]
10 return Top K components of Scores[]

```

Фигура 6.14 Основен алгоритъм за изчисляване на векторно пространствените оценки.

Фигура 6.14 показва базов алгоритъм за изчисляване на оценки във вектор-пространството. Масивът Length съдържа дължините за всеки от N-те документа, а масивът Scores - оценката на всеки документ. Когато оценките се изчислят в стъпка 9, в стъпка 10 се избират K документа с най-високите резултати. Най-външният цикъл, започващ от стъпка 3 повтаря актуализирането на Scores, правейки итерации върху всеки термин t от търсенето. В стъпка 5 се изчислява теглото на термин t във вектора на търсенето. Стъпки 6-8 актуализират оценките на всеки документ като добавят теглото на термина t . Процесът на добавяне едно по едно на теглата на термините от търсенето се нарича оценяване термин по термин (term-at-a-time scoring) или акумулиране, а N елементите на масива Scores – акумулатори.

Първо, ако използваме обратната честота на документа, не трябва да изчисляваме idf_t предварително - достатъчно е да запазим N/df_t в началото на срещанията за t (postings for t). Второ, запазваме честотата на термина $tf_{t,d}$ за всяко срещане. Накрая, стъпка 12 извлича топ K оценки - това изисква структура данни heap. Такъв heap отнема не повече от $2N$ сравнения, след които всеки топ K оценки могат да се извлекат от heap-а за $O(\log N)$ сравнения.

6.4 Променливи tf-idf функции

6.4.1 Подлинейно tf мащабиране

Едва ли двадесет срещания на един термин имат повече тежест от едно срещане на този термин в един документ.

Ако използваме логаритъма на честотата на термина, който дава тежестта зададена чрез

$$(6.13) \quad wf_{t,d} = \begin{cases} 1 + \log tf_{t,d} & \text{if } tf_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$$

то в тази форма, можем да заменим tf с друга функция wf, за да получим:

$$(6.14) \quad wf\text{-id}f_{t,d} = wf_{t,d} \times idf_t.$$

6.4.2 Максимална tf нормализация

Една добре изследвана техника е, да се нормализира tf тежестта на всички термини в документа по максималния tf в този документ. За всеки документ d , нека $tf_{max}(d) = \max_{t \in d} tf_{t,d}$ където τ се

мени измежду всички термини в d. Тогава пресмятаме нормализираната честота на термините за всеки термин в документа d като:

$$(6.15) \quad ntf_{t,d} = a + (1 - a) \frac{tf_{t,d}}{tf_{\max}(d)}$$

Термина a в (6.15) е смекчаващ термин чиято роля е да умекоти ролята на втория термин – което може да се разглежда като намаляване на tf по най-високата стойност на tf в d.

Основната идея на максимална tf нормализация е, да намали следната аномалия: наблюдаваме по-висока честота на термините в по-дълги документи, защото по-дългите документи често повтарят едни и същи думи отново и отново.

Пропуснати са три проблема на максималната tf нормализация.

6.4.3 Схеми за определяне на тежестта на търсене

Вариациите между различните вектор пространствени методи за оценяване се базират на избора на тежестите във векторите $V(d)$ и $V(q)$.

Term frequency	Document frequency	Normalization
n (natural) $tf_{t,d}$	n (no) 1	n (none) 1
l (logarithm) $1 + \log(tf_{t,d})$	t (idf) $\log \frac{N}{df_t}$	c (cosine) $\frac{1}{\sqrt{w_1^2 + w_2^2 + \dots + w_M^2}}$
a (augmented) $0.5 + \frac{0.5 \times tf_{t,d}}{\max_t(tf_{t,d})}$	p (prob idf) $\max\{0, \log \frac{N - df_t}{df_t}\}$	u (pivoted unique) $1/u$ (Section 6.4.4)
b (boolean) $\begin{cases} 1 & \text{if } tf_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$		b (byte size) $1/CharLength^\alpha, \alpha < 1$
L (log ave) $\frac{1 + \log(tf_{t,d})}{1 + \log(\text{ave}_{t \in d}(tf_{t,d}))}$		

Фигура 6.15 SMART нотация за tf-idf варианти. Тук CharLength е номера на символите в документа.

Фигура 6.15 изброява някои от основните схеми за определяне на теглото за $V(d)$ и $V(q)$, заедно с мнемониката за представяне на определена комбинация от тежести; тази система от мнемоники е наричана понякога SMART нотация. Мнемониката за представяне на комбинация от тежести има формата ddd.ddd, където първата тройка дава тежестта на термините във вектора на документа, а втората тройка дава тежестта на търсения вектор. Първата буква във всяка тройка задава компонента честота на термините от определената тежест, втората – компонента честота на документа, а третата – използваната форма за нормализация. Честа практика е, да се използват различни нормализации функции за $V(d)$ и $V(q)$. Например, стандартна схема за оценка на тежестта е $lnc.ltc$ където вектора на документа има логаритмично тегло на честотата на термините, няма idf (ако за ефективност, така и за ефикасност) и има косинус нормализация, докато търсения вектор използва логаритмично тегло на честотата на термините, idf тежест и косинус нормализация.

7 Изчисление на резултатите в завършена система за търсене

(Computing scores in a complete search system)

В тази тема ще разгледаме как да подобрим и ускорим изчисленията и алгоритъма за намиране на необходимия документ, въпреки, че често това води до проблеми при избирането на K документа с най-висок резултат.

7.1 Ефективно точкуване и класиране

За да точкуваме и класираме документите, ще използваме следният основен алгоритъм.

```
COSINESCORE( $q$ )
1 float Scores[N] = 0
2 Initialize Length[N]
3 for each query term  $t$ 
4   do calculate  $w_{t,q}$  and fetch postings list for  $t$ 
5     for each pair( $d, tf_{t,d}$ ) in postings list
6       do Scores[d] +=  $tf_{t,d} \times w_{t,q}$ 
7 Read the array Length[d]
8 for each  $d$ 
9 do Scores[d] = Scores[d] / Length[d]
10 return Top K components of Scores[]
```

► Figure 6.14 The basic algorithm for computing vector space scores.

Фиг. 6.14 Основен алгоритъм за изчисление на резултати с векторно пространство

Това е основният алгоритъм за изчисление на косинусовия резултат: q е заявката, а именно всички търсени термини, d е документа, t е термина, а $w_{t,d}$ представлява теглото на термина t в документа d .

Нека имаме заявката q . За нея имаме вектор $v(q)$. За да може да класираме документите, които отговарят на заявката, изчисляваме косинусова мярка за подобие (cosine similarity) на $v(d)$ и $V(q)$ на всеки документ. При $V(q)$ стойностите на всички ненулеви елементи на вектора са единици. Чрез косинусовата мярка за подобие се изчисляват приликите между два документа. Формулата е следната:

$$\text{sim}(d_1, d_2) = \frac{\vec{V}(d_1) \cdot \vec{V}(d_2)}{|\vec{V}(d_1)| |\vec{V}(d_2)|},$$

където d_1 и d_2 са два документа.

Тогава имаме : $\vec{V}(q) \cdot \vec{v}(d_1) > \vec{V}(q) \cdot \vec{v}(d_2) \Leftrightarrow \vec{v}(q) \cdot \vec{v}(d_1) > \vec{v}(q) \cdot \vec{v}(d_2)$, за всеки два документа d_1 и d_2 . За всеки документ d , косинусовата мярка за подобие $V(q) \cdot v(d)$ представлява

претеглената сума над всички термини в запитването q , на теглото на тези термини в d . Като се приложи това правило, алгоритъмът ще изглежда по следния начин :

```

FASTCOSINESCORE( $q$ )
1   float Scores[N] = 0
2   for each  $d$ 
3   do Initialize Length[ $d$ ] to the length of doc  $d$ 
4   for each query term  $t$ 
5   do calculate  $w_{t,q}$  and fetch postings list for  $t$ 
6       for each pair( $d, tf_{t,d}$ ) in postings list
7           do add  $wf_{t,d}$  to Scores[ $d$ ]
8   Read the array Length[ $d$ ]
9   for each  $d$ 
10  do Divide Scores[ $d$ ] by Length[ $d$ ]
11  return Top K components of Scores[]

```

► **Figure 7.1** A faster algorithm for vector space scores.

Фиг. 7.1 По-бърз алгоритъм за изчисление на резултати с векторно пространство

Алгоритъмът преминава през всички срещания (постинги) и изчислява крайния резултат за всеки един документ. На всеки термин даваме idf (inverse document frequency) стойност и за всяко срещане (постинг) - tf (the term frequency) стойност. Idf се изчислява по следната формула:

$$idf_t = \log \frac{N}{df_t},$$

където df представлява броя на документите в колекцията, които съдържат термина t .

Използваме idf, тъй като за по-редки думи той ще даде по-висок резултат, а за по-често срещаните, по-нисък. Честотата на термина (the term frequency) представлява броя на срещанията на търсения термин във всеки документ. Чрез idf и tf се дава тежест на всеки термин и това тегло се използва за изчисляване на резултата и подреждане на документите. Резултатът се изчислява по следния начин:

$$\text{Score}(q, d) = \sum_{t \in q} tf \cdot idf_{t,d}.$$

Алгоритъмът изчислява резултат за всеки документ, за когото и да е търсено условие. Следващата стъпка е да се изберат първите K документа с най-висок резултат. Най-добрият подход е да се използва heap.

7.2 Извличане на най-близките (релевантни) към документа (**In exact top K document retrieval**)

Не е необходимо задължително да изтеглим K документа с най-високи резултати. Достатъчно е да се изтеглят K документа, които са най-вероятни и близки до тези документи. По този начин

се намалява цената на изчислението, тъй като не винаги К документите с най-висок резултат са най-добрите К документа за заявката. Най-голям разход се получава от големия брой документи и косинусовата мярка за подобност. Това може да бъде решено чрез следните предложения:

- Първо избираме множество от A документа, така че $K < |A| \leq N$, като A много вероятно ще съдържа голям брой документи с висок резултат.
- Връщаме K документите с най-висок резултат в A.

7.3 Премахване на индекси (Index elimination)

Когато имаме вектор с много условия, за да съкратим и улесним изчисляването на косинусите следваме две правила:

- Първо ще махнем всички документи, при които idf на търсените думи е по-малък от предварително зададен от нас такъв. По този начин значително ще се намали броя на документите.
- Избираме само документи, които съдържат много от търсените думи. Проблемът тук е, че може да останат по-малко от K документа.

7.4 Шампионски списъци (Champion lists)

Идеята е да се изчисли предварително множество от r (предварително зададено) документа за всеки термин t, където t е с най-висока тежест. След като са избрани подобни множества за всеки от търсените термини, тези множества се обединяват и се изчислява косинуса само на тези документи. Стойността на r трябва да е по-голяма от K и също така не е нужно да е еднаква за всички термини. Проблем може да се появи, ако няма достатъчно документи, поради премахване на индекси, например.

7.5 Статични качествени резултати и подреждане (Static quality scores and ordering)

Повечето търсачки имат статична (независма) мярка за качество g(d) за всеки документ. Това е число между 0 и 1. По следващата формула се изчислява netscore:

$$\text{net-score}(q, d) = g(d) + \frac{\vec{V}(q) \cdot \vec{V}(d)}{|\vec{V}(q)| |\vec{V}(d)|}.$$

Идеята тук е да се поддържа глобален шампионски списък от r документа, които имат най-високи стойности $g(d) + tf-idf_t, d$ за всеки термин t, като r ни е известно. Когато се обработва заявката, тогава се изчислява netscore-a. Друга идея е, да се направят два списъка, като първият, наречен high, ще състои от тези документи, които имат най-високи tf стойности. Другият списък, наречен low, ще съдържа всички останали документи. По този начин, първо ще бъде сканиран списъка с високите стойности.

7.6 Подреждане според влиянието (Impact ordering)

Подреждането на документи става последователно чрез ID или чрез статични качествени резултати. При тях за всеки документ се изчисляват резултатите, когато се срецне този документ. Но има и други начини - при тях изчисляването на резултата става отделно за всеки термин. Това е техника за *in exact top - K retrieval*.

Подреждаме документите за всеки термин в низходящ ред на $tf_{i,d}$. По този начин ние не можем да изчислим резултата. За да се намали броя на документите, в този случай има две идеи. Първата е при преминаване по списъците да се спира, в зависимост от стойността на r или ако стойността на $tf_{i,d}$ е под границата. Втората идея е да подредим термините в низходящ ред по idf , така че първо разглеждаме тези, които най-вероятно ще окажат влияние. Възможно е, когато се разглеждат термините с ниски idf , те да бъдат изпуснати, ако има малки промени в сравнение с предишни термини.

7.7 Съкращаване на клъстерите (Cluster pruning)

Този метод е предварителна стъпка, при която събираме векторите на документите. Изчисляваме косинусовия резултат на тези документи, които са в малки клъстери. Процесът е следният:

- Първо избираме произволно \sqrt{N} документи и се обаждаме на тези лидери.
- За всеки документ, който не е лидер, изчисляване най-близкия такъв.

Наричаме тези документи последователи. Броят им за всеки лидер е \sqrt{N} .

Заявката се обработва по следният начин:

- Предвид заявката q се намира лидера L , който е най-близо до q , като се изчислява косинуса от q до всеки от лидерите.
- Множеството A съдържа L , както и неговите последователи. Изчислява се косинусовия резултат за всеки документ в множеството.

Ако изберем произволно лидерите, процесът е по-бърз и в зависимост от разпределението, и по-фин. Има доста вариации на съкращане на клъстери. Един от тях е да се използва b_1 и b_2 . Те са цели положителни числа и представляват броя на най-близо разположените лидери. Като увеличим b_1 и b_2 увеличаваме шанса да се намерят K документите, които имат най-високи резултати.

7.8 Компоненти на система за извлечение на информация

7.8.1 Слоести индекси

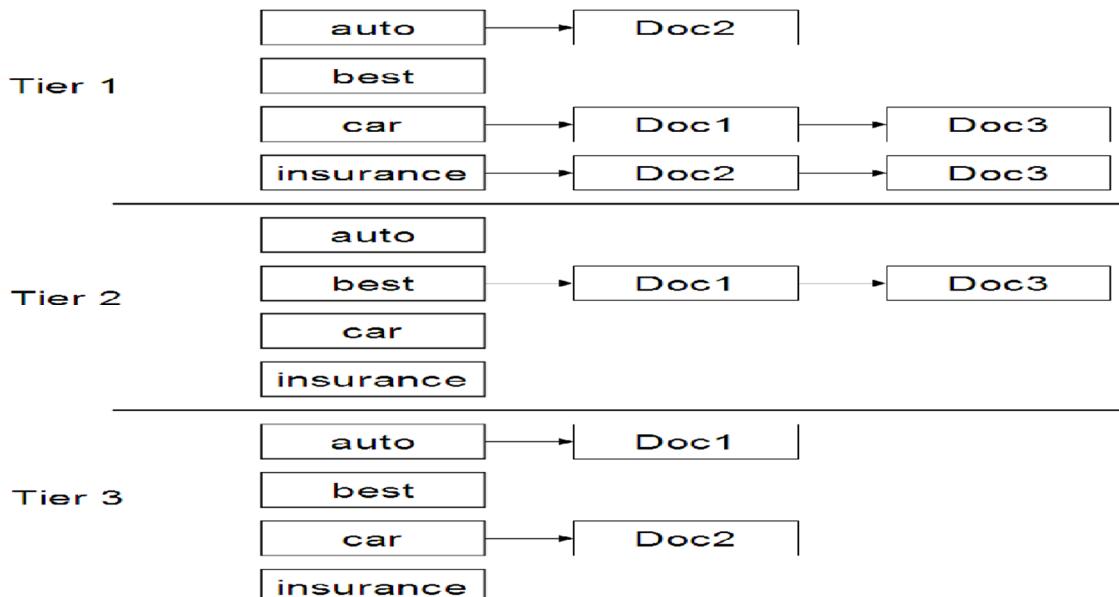
Слоестите индекси най-общо могат да бъдат представени като обобщаване на списъка с най-релевантни за термина документи (champion list, fancy list или top docs). Използването на тези индекси се налага от това, че при употребата на практики като премахване на индекси при извлечането на най-близките към документа (*inexacttop-Kretrieval*), където оценяването се

извършва за всеки термин поотделно, може да се получи множество, което има по-малко от k документа. За пример ще дадем документите и термините в таблица 1, като ще зададем граница от 20 за слой 1 и граница 10 за слой 2. Слой 1 и слой 2, на практика, са две търсения на една и съща заявка със зададени различни граници. При така зададени рестрикции в индекса на слой 1 ще имаме срещания (постинги) с term frequency (tf) над 20, докато в този за слой 2 - само такива с над 10.

	Doc1	Doc2	Doc3
car	27	4	24
auto	3	33	0
insurance	0	33	29
best	14	0	17

Таблица 1

Както е показано на фигура 1, документите във всеки слой са подредени по Document ID. Смисълът на слоевете е, че когато не можем да намерим k резултата, които удовлетворяват заявката, ние „слизаме“ на по-долно ниво, т.е. ако слой 1 не съдържа k резултата, взимаме слой 2, после слой 3 и т.н.



Фигура.1

7.9 Близост на термините от заявката

При заявките, състоящи се от свободен текст, особено при уеб търсенето, потребителите предпочитат документи, в които повечето от термините се срещат близко един до друг, защото това е доказателство за релевантността на намерената информация. Нека разгледаме заявка с

няколко термина: t_1, t_2, \dots, t_k и ω е най-малката дължината на броя думи в текста, в които се срецат всички термини от заявката. За оптимален резултат, може да добавим следните правила:

- колкото по-малко е ω , толкова по-релевантен за заявката е документа;
- ако всички думи не се срецат в дадения документ, на ω присвояваме за стойност достатъчно голямо число;
- друг вариант е, при изчисляването на ω , да се вземат предвид само думи, които не са стоп думи.

Такива изграждания на тегла за близост се отдалечават от модела за косинуса на ъгъла между вектори и се приближават повече до техниките, ползвани при уеб търсенето.

7.10 Построяване на функции за оценяване и предаване на заявката

Повечето от интерфейсите за търсене, скриват от нормалния потребител истинската форма, под която се задава заявката и му предоставят тестово поле за свободен текст с цел улесняване. При такива обстоятелства, оценяването на релевантните документи зависи от информацията, която имаме за дадения потребител, разпределението на заявката и от наличните документи. Предаването на заявката цели да определи ключовите думи от нея и създаването на индекси. Това може да доведе до редица различни заявки:

- Заявка под формата на цялостна фраза, като на термините се дава ранк чрез метода на векторното поле;
- Ако релевантните документи се окажат по-малко от K , делим фразата на подфрази и пак оценяваме посредством метода на векторното поле и т.н.;
- Ако не сме намерили достатъчен брой релевантни документи, разделяме фразата на термини и отново оценяваме с метода на векторното поле.

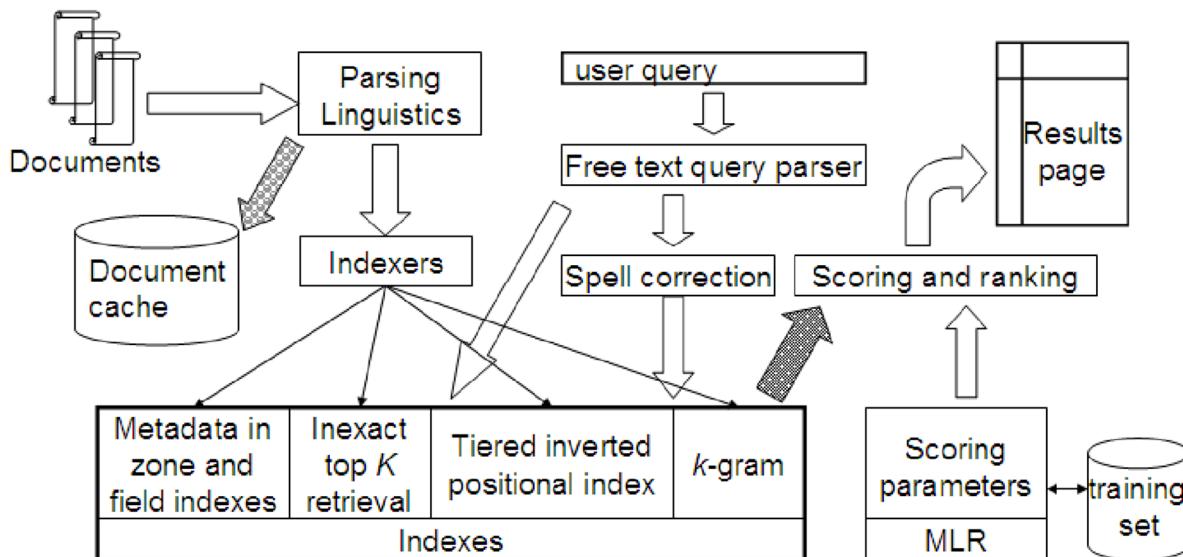
В резултат на тези стъпки се получават различни резултати, които трябва да се съберат и обработят на базата на много фактори, като оценяването с векторно поле, теглата на близост, получаването на един и същи документ в различните стъпки и т.н., което е предпоставка за нуждата от оценяваша функция, която да акумулира резултата от фазите.

Най-трудната задача за решаване при разработката на изискваните parse-и и оценявящи функции е променящата се база от документи, затова доста често се прилага до методи за машинно самообучение, което самостоятелно да оценява измененията.

7.11 Изграждане на цялостна цялостна система за търсене

Фигура.2 изобразява действията, които се извършват при обработката на документите и потребителската заявка. Документите преминават през лингвистически преобразувания, след което подаваме думи (tokens) на индексатор и кеш, където лесно може да се изгради част от документа, представяща неговото съдържание. Изграждат се индекси (указани на фигура.2), отново на базата на получените от лингвистичното преобразуване думи (tokens).

От друга страна, заявката на потребителя минава през предаване като там, както вече споменахме, се образуват ключови думи и различни заявки. Друго преобразуване на първоначалната заявка се извърши с цел проверяване на правописа . След това и двата резултата се индексират. След това получените документи се подават за оценяване, спрямо вече изграден модел за машинно самообучение и най-релевантните се подават към резултатната страница.



Фигура.2

7.12 Модел на векторното оценяване и връзката му с други оператори за заявки

Първо трябва да отбележим, че можем да поддържаме различни оператори за заявки за крайния потребител, които трябва да споделят ресурси и индекси. Предимството на векторния модел е, че той поддържа заявки със свободен текст, както и оценяването на документи, докато при булевия, wildcard и заявките с фрази не се поддържат. Ако разгледаме предимствата и недостатъците на четирите оператора трябва да отбележим, че булевия модел е напълно заменим от векторния при положение, че нямаме нулеви вектори, но обратното не е вярно, поради поддържане на тегла при векторния. Математически е възможно двата модела да се комбинират, но такава система до сега не съществува. Относно векторния модел и wildcards можем да отбележим, че те ползват напълно различни индексирания, освен ако не погледнем имплементацията на основно ниво, където те могат да се представят като срещания (постиинги) и речник. От общ план, векторният модел също е по-добър, защото ще представи същите резултати, но те ще имат и тегла. От друга страна, когато сравняваме заявките по фраза и векторният модел, можем да констатираме, че векторният модел е прахоснически, т.е теглата са ненужни, защото те носят по-силна рестрикция и от друга страна, не е задължително векторният модел да покаже документи, съдържащи търсената фраза.

8 Оценяване в извлечането на информация

(Evaluation in information retrieval)

Разработени са различни алтернативни подходи в дизайна на системите за извлечане на информация, но много важен въпрос свързан с тяхното прилагане е това, колко ефективни ще бъдат бъдещите приложения. Дали трябва да се използват стоп листи, дали трябва да се стемва? Извлечането на информация се е развило като силно емпирична дисциплина, която се нуждае от внимателно и пълно оценяване, за да може да се покажат предимствата на новаторски техники върху представителни колекции от документи.

В тази статия ще бъдат обсъдени измерването на ефективността на системите за извлечане на информация и какво представляват тестовите колекции, които най-често се използват за тази цел. След това ще се въведе терминологията за съответстващи и несъответстващи документи и формалната терминология за оценяване, която е развита за извлечането на некатегоризирани резултати. В това се включват видовете мерки, които обикновено се използват за извлечането на документи и свързаните с това задачи като класифицирането на текст. След това, тези идеи се разширяват и се показват по-детайлни мерки за оценяването на категоризирани резултати и се обсъжда създаването на информативни тестови колекции.

Ще стане въпрос и за понятието за потребителско удобство и как се приближава с употребата на свързаност на документите. Ключовата мярка за ползата е потребителското удовлетворение. Скоростта на отговор и размерът на индекса са част от пътя към това удовлетворение. Най-важната част от него, логично, изглежда да бъде свързаността на резултатите. Обаче, очакванията за качество на потребителите най-често не съвпадат с тези на дизайнера на системата. Например, потребителското удовлетворение много силно зависи и от това как изглежда интерфейса на системата. Много добре се повлиява покриването на потребителското очакване чрез генерирането на качествени резюмиращи откъси от търсените резултати, които не са измервани от основната парадигма на класифициране на свързаността.

8.1 Оценяване на системите за извлечане на информация

За да може да се измерва ефективността на специалното извлечане на информация по стандартен начин, е необходим тестов набор състоящ се от три неща:

1. Набор от документи
2. Тестов набор от информационни нужди, изразени чрез заявки
3. Множество от преценки за уместност, които стандартно се задават двоично, като уместни и неуместни за всяка двойка от заявка-документ.

Стандартният подход за оценяване касае представянето на документите като уместни и неуместни (релевантни и нерелевантни). По отношение на потребителската нужда от информация, един документ от тестовия набор може да бъде представен като уместен или неуместен. Това решение е известно като златен стандарт или основна истина при преценяването за уместност. Тестовия набор от документи и информационни нужди трябва да бъде в достатъчно голям размер – трябва да се постигне добро изпълнение при сравнително

голям брой тестове, като резултатите варират при различните документи и нужди от информация. Като емпирично правило се смята, че 50 информационни нужди са достатъчен минимум.

Уместността се оценява относително спрямо нужда от информация, а не от заявка. Например, една такава нужда от информация може да бъде: *'Информация дали пиещото на червено вино е по-ефективно при намаляването на риска от инфаркт, отколкото бялото вино.'* Това може да бъде преведено като следната заявката: *'вино И червено И бяло И инфаркт И ефективен'*

Един документ е уместен, ако отговаря на изразената нужда от информация, а не защото просто се оказва, че съдържа всички думи, които се съдържат в заявката. Тази разлика, на практика, често се оказва, че не се разбира правилно, защото нуждата от информация не е явна. Но въпреки това, такава нужда е налична. Ако потребителят напише `python` в уеб търсачка, той може би иска да знае от къде може да си купи питон като домашен любимец, повече информация за змията или пък иска информация относно езика за програмиране Python. От една дума като заявка е трудно да се определи за каква нужда става въпрос. Но въпреки това, потребителят има такава нужда и може да оцени върнатите му резултати на база тяхната уместност спрямо нея. За да се оцени система, се нуждаем от изрази, които могат да се използват за оценяването на върнатите документи като уместни или не. Тази уместност може да се мисли като скала, като някои от документите са много на мястото си, а други по-малко, но за улеснение първо ще използваме бинарно решение - уместни или не.

Добрата практика е, да има една или повече тестови колекции и да се изprobват тежестите спрямо тях. По този начин, след като се тества тестовия набор, може да се отчетат безпредубеждени резултати.

8.2 Стандартни тестови набори

Най-известните стандартни тестови набори са:

- коллекцията ***Cranfield*** – една от първите тестови колекции, които позволяват точно количествено измерване на ефективността на информационните системи, но за днешни дни е твърде малка. Събрана е във Великобритания в началото на 1950 и съдържа 1398 текста за аеродинамика и 225 заявки, както и изчерпателно оценяване на всяка двойка документ-заявка.
- ***TREC (Text Retrieval Conference)*** – Американският институт по стандарти и технологии (NIST) е провел много оценки на системи за извлечение на информация от 1992 г. насам. За тези цели, е събрана колекция от множество различни документи, но най-известните са използвани за ***TREC Ad Hoc*** по време на първите 8 TREC оценявания между 1992 и 1999 г. Общо тези колекции се събират на 6 компакт диска, като съдържат 1.89 miliona документа (предимно новинарски статии) и оценяване за свързаност на 450 информационни нужди, наречени теми и подробно описаны в текстови пасажи. Индивидуалните тестови колекции съдържат различни подмножества от тази информация. Ранните TREC, са съдържали 50 информационни нужди, оценявали с различни, но припокриващи се множества от документи. 6-8 версии на TREC предлагат 150 информационни нужди върху около 528,000 телеграфни и

чуждестранни новинарски статии. Това е може би най-добрата част за бъдеща работа, тъй като тя е най-голяма и темите са по-консистентни. Заради големия размер на тестовите колекции, няма изчерпателни оценки за уместност. Вместо това, тези оценки са на разположение само за документите, които са сред първите няколко, върнати от някоя система, подложена на оценка от TREC и за която информационната нужда е била развита.

- **GOV2** – в последните години, NIST проведе оценявания на по-големи колекции от документи, включващи GOV2 уеб колекцията от 25 милиона страници. От начало, тестовите колекции на NIST са по-големи от всичко достъпно за изследователите преди това и в момента GOV2 е най-голямата уеб колекция, лесно достъпна за изследователски цели. Въпреки това, размера на GOV2 е все още с повече от два порядъка по-малък от размера на колекциите от документи, индексирани от големите компании за уеб търсене.
- **NTCIR (NII Test Collections for IR Systems)** – този проект е изграден около различни тестови колекции с размери, подобни на колекциите в TREC, само че са фокусирани върху източно-азиатските езици и междуезиковото извлечане на информация. Запитванията се правят на един език към целия набор от документи, които са на един или повече езици. Повече на http://research.nii.ac.jp/ntcir/data/data_en.html
- **CLEF (Cross Language Evaluation Forum)** – оценяваша серия, насочена към европейските езици и междуезиковото извлечане на информация. Повече на <http://www.clef-campaign.org/>
- **Reuters** – за класифициране на текст, най-използваната колекция е тази на Reuters от 21 578 телеграфни статии. В последно време, Reuters издадоха и по-голям корпус (Reuters Corpus Volume 1), който съдържа 806 791 документа. Размерът и богатото анотиране го прави добра основа за бъдещи изследвания.
- **20 Newsgroups** – друга широко използвана колекция за класифициране на текст, събрана от Кен Ланг. Съдържа по 1000 статии от всяка от 20 Usenet новинарски групи (името на новинарската група се смята за категорията). След премахването на повторенията остават 18 941 уникални статии.

8.3 Оценяване на некатегоризирани множества от върнати резултати

След като представихме основните колекции, как се оценява ефективността на дадена система? Най-често се използват две основни мерки – прецизност и връщане (precision и recall). Първо ще ги дефинираме в простия случай, когато системата за извлечане на информация връща множество от документи за дадена заявка.

Прецизността (P) е частта на получените документи, които са уместни.

$$\text{Прецизност} = \frac{\#(\text{уместни извлечени документи})}{\#(\text{извлечени документи})} = P(\text{уместни} | \text{извлечени})$$

Връщането е частта от уместните документи, които са получени.

$$\text{Връщане} = \frac{\#(\text{извлечени документи})}{\#(\text{уместни документи})} = R(\text{извлечени} | \text{уместни})$$

Тези нотации могат да се изяснят като се погледне следната таблица:

	Уместен	Неуместен
Извлечен	Верен извлечен (true positive - tp)	Неверен извлечен (false positive - fp)
Неизвлечен	Невярно неизвлечен (false negative - fn)	Вярно неизвлечен (true negative - tn)

Тогава:

$$P = \frac{tp}{(tp + fp)} \quad R = \frac{tp}{(tp + fn)}$$

Очевидна алтернатива за измерване на система за извличане на информация, която може да се срещне е *точността (accuracy)* – частта от класификациите, които са коректни. В термините на таблицата по-горе,

$$\text{точност} = (tp + tn) / (tp + fp + fn + pn).$$

Това изглежда приемливо, тъй като има два същински случая – уместен и неуместен, и системата за извличане на информация разделя резултатите в два класа, (тя показва подмножеството от документите, които смята, че са уместни). Това е мярката за ефективност, която често се използва за оценяване на машинно самообучаващи класификационни проблеми.

Има основателна причина, обаче, защо точността не е подходяща мярка за проблеми, относно извличането на информация. В голяма част от всички случаи, данните са изключително несиметрични - нормално 99,9% от документите са неуместни. Система, която е настроена да има максимална точност може да се окаже, че ще се държи добре, просто като при всяка заявка смята всички документи за неуместни. Дори системата да е доста добра и да отбележи някои от документите като уместни, почти винаги има високо ниво от fp (false positive/неверни извлечени). Но отбелязването на всички документи като неподходящи, е напълно нездадоволително за потребителя. Той винаги ще иска да погледне някои от документите и може да се предположи, че ще има определен толеранс да види няколко невярно извлечени, при условие, че ще получи малко полезна информация. Мерките прецизност и връщане (precision & recall), се концентрират върху получаването на вярно извлечени (tp – true positive), питайки за процента от уместните документи, които са открити и колко невярно извлечени са били върнати.

Предимството на двете величини – прецизност и връщане е, че едната може да бъде по-важна отколкото другата при много случаи. Обикновеният уеб потребител би искал всеки резултат на първата страница да е уместен (висока прецизност), но няма голям интерес в това, да разгледа отделно всеки касаещ го документ. От друга страна, различни професионални търсачи, като разпознавателни анализатори са много засегнати от получаването на колкото се може по-голямо връщане и ще толерират сравнително ниска прецизност на резултатите, стига да ги получат. Въпреки това, двете величини просто се компенсират една с друга: винаги може да се получи

връщане R със стойност 1 (но с много ниска прецизност), като се покажат всички документи за което и да е запитване. Връщането R не намалява спрямо това колко документи са получени. От друга страна, в една добра система, прецизността обикновено намалява с увеличаването на броя извлечени документи. Като цяло искаме да получим някакво връщане R, като се толерира само определен процент от невярно извлечени (fp).

Единична мярка, която съчетава прецизността с връщането е F мярката, която е претегленото хармонично средно на прецизността и връщането:

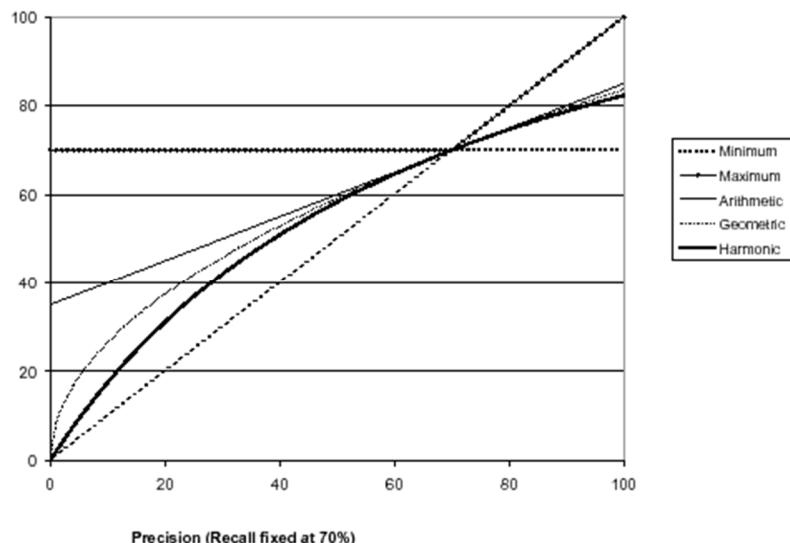
$$F = \frac{1}{\alpha \frac{1}{P} + (1 - \alpha) \frac{1}{R}} = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}, \text{ където } \beta^2 = \frac{1 - \alpha}{\alpha}$$

Където $\alpha \in [0,1]$ и по този начин $\beta^2 \in [0, \infty]$. По подразбиране, F е балансираната мярка, която еднакво оценява прецизността и връщането, което означава да се положи $\alpha = \frac{1}{2}$ или $\beta = 1$. Това често се записва като F_1 , което е на кратко записано $F_{\beta=1}$, въпреки че формулирането в термини на α по-прозрачно отразява F мярката като претеглено средно хармонично. Когато използваме $\beta = 1$, изразът от дясната страна се опростява до:

$$F = \frac{2PR}{P + R}$$

Обаче, даването на еднаква тежест не е единствения избор. Стойностите $\beta < 1$ изразяват прецизността, а стойностите $\beta > 1$ – връщането. Например, стойностите $\beta = 3$ и $\beta = 5$ могат да се използват, за да се даде приоритет на връщането. Прецизността, връщането и F мярката, могат да имат стойности между 0 и 1, но най-често те се представят като проценти, върху скала от 0 до 100.

Зашо да използваме средно хармонично, а не по-простото средно аритметично? Важен момент е това, че можем да постигнем 100% връщане, като просто се връщат всички документи и по този начин постигаме 50% средно аритметично. Това силно предразполага средното аритметично като неподходяща мярка за използване. За сравнение, ако приемем, че един от 10 000 документа е уместен, тогава средното хармонично в този случай ще бъде 0.02%. Средното хармонично е винаги по-малко или равно на аритметичното и геометричното средно. Когато тези стойности за две числа се различават много, средното хармонично е по-близо до по-малкото отколкото средното геометрично.

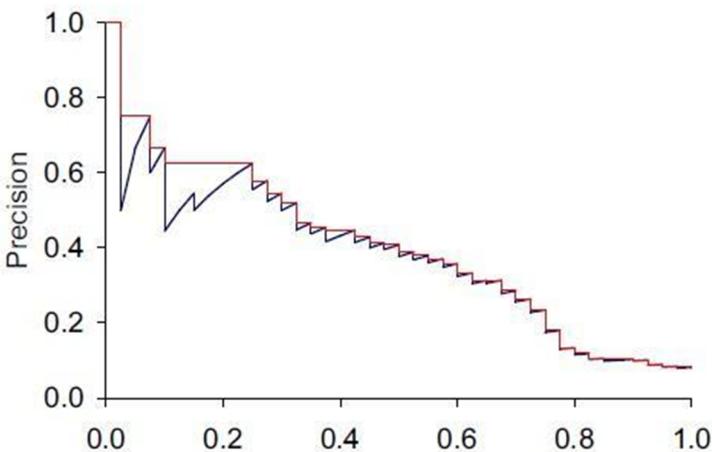


Фиг. 8.1

На фиг.8.1 е показана графика, която сравнява средното геометрично, спрямо другите средни. Показан е отрезък от изчисляването на различни средни на прецизността и връщането при фиксирана стойност на връщането 70%. Хармоничното средно е винаги по-малко от аритметичното или геометричното и е често доста близо до по-малкото от двете числа. Когато прецизността е 70%, всички стойности съвпадат.

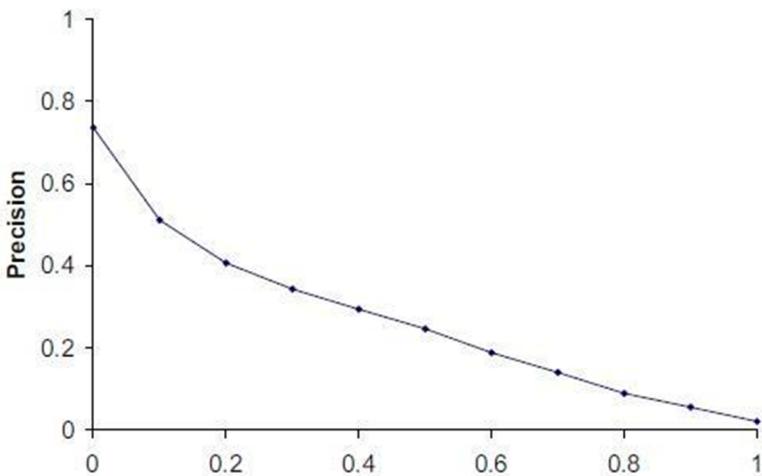
8.4 Оценяване на категоризирани върнати резултати

Нека точността и F-мярката бъдат нашите базови мерки. Те се изчисляват с помощта на неподредени набори от документи. За всеки такъв набор, точните стойности могат да се начертаят и да се изобрази точна крива. Формата на такъв вид крива наподобява зъб на трион. Ако $k+1$ -ия получен документ е неуместен, тогава отговорът е същият както за k -ия документ, но така коефициента на прецизност на самата диаграма пада. Ако документа е уместен, то тогава точността нараства и кривата се назъбва нагоре и надясно. Често, по-полезно е, вместо да представяте чрез зъбици, да използвате интерполиращата точност: интерполиращата точност P_{interp} на дадено средно ниво r е дефинирана като най-точното за всяко средно ниво $r' >= r$ т.e. $p_{\text{interp}} = \max p(r')$ за всяко $r' \geq r$. Обосновката на този вид представяне е, че почти всеки ще е готов да прегледа няколко документа повече, ако това ще увеличи процента на резултата изграден върху приложимите документи. Прецизността на интерполяцията е изобразена с тънката линия на фигурата.



Фиг.8.2

Точно проучване на цялата крива е много информативно, но често е достатъчно информацията да се намали само до няколко числа, или може би дори едно число. Традиционният начин да се направи това е 11 точковата интерполирана средна точност. За всяка нужна информация, интерполираната точност се измерва на 11 нива: 0.0, 0.1, 0.2, . . . , 1.0. За всяко ниво се изчислява точната аритметичната средна стойност на това ниво на интерполяция в тест колекцията. След това, могат да бъдат изобразени със съставната точност като графики чрез крива, показваща 11 точки. Следващата фигура представя примерна графика с подобни резултати, взети от системата TREC 8.



Фиг.8.3

През последните години обаче, други стандарти са станали по-общоприети. Общоприетият от методите в TREC е 'MAP' (означена средна стойност), който представлява определен стандарт за всички нива. Ако наборът от средните стойности на съответните документи е $q_j = \{d_1, \dots, d_m\} \in Q$ и R_{jk} е групата от класирани резултати на възвръщане от горния резултат, докато не се достигне документа d_k то: $MAP(Q)=1/|Q| * \sum_{j=1}^{|Q|} \frac{1}{m_j} \sum_{k=1}^{m_j} Precision(R_{jk})$. Когато даден документ не е извлечен, то стойността на точност е 0. Използвайки MAP, не се избират фиксирани нива и не се наблюдава интерполяция. Стойността на MAP се явява като аритметичното средство за точност на средните стойности на отделните информационни потребности. Вече изчислени,

MAP стойностите на най-нужната информация варират м/у 0.1 и 0.7. Това означава, че наборът трябва да бъде голям и достатъчно разнообразен за да бъде ефективен за системата по различните запитвания. Горепосочените мерки отговарят на всички нива. За много приложения, особено за уеб търсенето, това може да не касае потребителите, но е важно колко добри резултати има на първата страница или на първите три страници например. Това води до измерване точността на фиксиранi ниски нива на резултати. Може да се каже, че това е малък недостатък. Алтернативата, която облекчава този проблем е R-точността. Нужно е да има набор от документи Rel, от които ще пресметнем точността на най-ефективните такива, които са върнати. R-точността се награжда по размера на набора документи. Ще има повече смисъл, обаче ако просто усредним всички. Ако имаме $|Rel|$ уместни документи ние изследваме топ $|Rel|$ резултатите на системата и ще открием, че r (което посочихме по-горе) е приложимо, а това от своя страна по дефиниция е $r/|Rel|$. По този начин R-точността се оказва, че е идентична с т.нар. рентабилност (друга мярка, която понякога се използва, дефинирана за такъв тип равенство). Както точността на k , R-точността описва само една точка на кривата на изземване и до някъде е неясно защо трябва да се интересуваме от рентабилността, вместо просто от най-добрите точки от кривата. Все пак R-точността е тясно свързана с MAP емпиричността, въпреки единствената си точка за измерване от кривата.

Друга концепция, която понякога се използва в изчисленията е ROC кривата. Тук се наблюдават стойности за скорост и в случая грешната позитивна скорост се представя чрез формулата $fp/(fp + tn)$. Този вид крива по принцип тръгва от абсцисата на горе вдясно на графиката, а за една добра система, графиката се изкачва от лявата страна. Тъй като неизвлечените (негативните) стойности обикновено са твърде големи, то ако използваме формулата $tn/(fp + tn)$, стойността и може да бъде най-много 1. ROC кривата може да се използва и за извлечение на пълен спектър, както и друг начин чрез който може да се четат данните. В много случаи, общата мярка се отчита в зоната под кривата, което е аналогично при MAP-кривата.

Последният вид е т.нар. NDCG модел. Той е предназначен за ситуации, когато нямаме двоичност на връзките. Имаме и формулата

$$NDCG(Q,k) = 1/|Q| * \sum Z_{kj} \sum ((2^{R_{j,m}} - 1) / \log_2(1+m)),$$

където Z_{kj} изчислява нормализиращия фактор, за да се получи дадено класиране в k за заявка j .

Давайки нужната информация и съответните документи, трябва да дадем информация и за оценките. Това отнема време много труд. За малки колекции като Cranfield са получени изчерпателни решения, които са от значение за всяка заявка и двойка документи. За големи колекции, каквито са съвременните, обаче, е обичайно значението да се оцени само за подмножество от документи на всяка заявка. Най-стандартен подход е обединяването, където значението се оценява над подмножеството от колекцията, която произхожда от горния клас към документи, които са върнати от няколко различни системи. Човекът, разбира се, не е устройство, което би могло надеждно да докладва стандартната оценка. Хората и техните релевантни решения са доста характерни и променливи, но това не е проблем, който не може да се реши, все пак в заключителния си анализ, успехът на една IR система зависи от това колко добре е обработила дадена информация, която е била необходима в даден момент. В социалните науки е въведена общата мярка „ kappa “. kappa = $P(A) - P(E)/1 - P(E)$, където $P(A)$ е съотношението на времената, а $P(E)$ е съотношението на времената на очакване. Стойността на

ката ще е 0, ако времената съвпаднат и 1 в противен случай. Ако имаме повече от две такива оценки, взимаме средно аритметичното от тях. Предимството в оценка на системата, както е разрешено от стандартния модел на съответните документи е, че имаме фиксирана настройка в която тя може да варира, и се използва от IR системи и системни параметри за извършване на сравнителни опити. Това тестване е много по-скъпо, но гълък позволява по-ясно диагностициране влиянието на промените в системата. Има множество примери за техники, разработени в официалната оценка, които подобряват ефективността в оперативните настройки, като например развитието на методите в контекста на TREC.

8.5 Измерване на потребителската удовлетвореност

Формалните мерки за оценяване, обаче, са по някакъв начин далеч от нашия всеобхващащ интерес към измерването на човешката нужда: по какъв начин остава удовлетворен всеки потребител от резултатите, които системата дава за всяка информационна нужда, която той изпита? Стандартният начин да се измери човешкото удовлетворение е, чрез различни видове потребителски случаи. Това може да включва количествени мерки, които са обективни - напр. времето необходимо за завършване на задача, или субективни - като фактор, който се образува от задоволството от търсачката и количествени величини, например потребителските коментари относно интерфейса на търсене. В тази точка ще се разгледат някои други аспекти, които позволяват количествено оценяване и проблема за потребителското удобство.

Има много практически скали, на които може да се оценява, освен качеството на извлечане на информация от системата и много други важни черти. Те включват:

- Колко бързо се индексира, което означава колко документа за час може да се индексират за определено разпределение, според дължините на документите.
- Колко бързо се търси – какво е забавянето според размера на индекса.
- Колко експресивен е езика за запитвания до системата? Колко бърза е тя при сложни запитвания?
- Колко е голям списъкът с документи, в смисъл на броя документи или на колекцията, имаща информация, разпределена сред широк набор от теми?

Всички тези критерии, безекспресивността на езика за запитвания, са доста лесно измерими – можем да оценим скоростта или размера. Различни видове на списъци с изисквания, може да направят и експресивността на езика за запитвания при оценяване.

Това, което наистина бихме оценили като начин за измерване на събраното потребителско щастие се основава на уместността, скоростта и на потребителския интерфейс на системата. Една част от това е разбирането на разпределението на хората, които искаме да задоволим и това изцяло зависи от нагласите. За една уеб търсачка, щастливите потребители са тези, които намират това, което търсят. Една непряка величина за измерването на това потребителско щастие е доколко те ще се завърнат на същата търсачка. Измерването на честотата, с която потребителите се завръщат, е ефективна мярка, която би могла да бъде още по-ефективна, ако може също и да се определи и доколко потребителите преди това са използвали други търсачки. Но също така и рекламиралите са потребители на съвременните уеб търсачки. Те

остават доволни, ако клиентите стигнат до техния и сайт и направят покупки от там. За един комерсиален уеб сайт е най-вероятно потребителя да иска да купи нещо от него. Затова, могат да се измерят времето до покупката, както и каква част от търсепците се превръща в купувачи. На сайта на даден магазин, може би едновременно при покупка се задоволяват нуждите и на потребителя, и на собственика на магазина. Въпреки това, в общия случай трябва да се реши чие щастие искаме да оптимизираме – това на крайния потребител или на собственика на комерсиалния сайт. Най-често това е собственика на магазина, който плаща за това. За фирмени (на компания, правителство или академична) вътрешна търсачка, уместната метрика изглежда да бъде потребителската производителност – колко време прекарват потребителите в търсене на информацията, от която се нуждаят. Има много други практически критерии, които касаят и такива теми, като информационната сигурност.

Потребителското щастие е неуловимо за измерване и е част от това, защо стандартната методология използва заместването му с уместността на търсения резултати. Стандартния директен начин да се постигне потребителско удовлетворение е, да се използват случаи на употреба (use cases), където на хората се дават различни задачи, като обикновено се измерват различни метрики, наблюдават се участниците, и се използват интервюта, за да се получи количествена информация относно удовлетворението. Случаите на употреба са много полезни при системния дизайн, но освен това те са доста времеотнемащи и скъпи за провеждане. Също така, е трудно те да се проведат правилно и са необходими знания, за да се изградят случаи и да се анализират резултатите. Тук вече става въпрос и за тестването на потребителската използваемост, която е една много обширна тема.

Ако една система за извличане на информация е изградена и е използвана от голям брой потребители, то разработващия екип на системата може да оцени възможните промени, като разполага с различаващи се версии на системата и записва и сравнява измервания, които са свързани с потребителската удовлетвореност. Това е практика, която доста често се прилага при уеб търсачките.

Най-честата версия на това е /В тестването, което е термин зает от рекламната индустрия. За такъв тест, само едно нещо се променя от текущата система в новата система и малка част от трафика (да речем 1-10%) се препраща на случаен принцип към променената нова система, докато повечето потребители използват текущата. Например, ако искаме да изследваме промяна в алгоритъма за оценяване, може да се пробва да се насочват случајна част от потребителите към вариация на системата, за която се използва новия алгоритъм, и да се измерят величини като честотата, с която хората кликат върху най-горния резултат или по който и да е резултат от началната страница. Основата на А/В тестването е използването на няколко тестове на единични променливи (или последователно, или успоредно): за всеки тест един параметър се променя от о, текуща система. По този начин е лесно да се види дали промяната на един параметър има положителен или отрицателен резултат. Такова тестване на работещи системи може лесно и евтино да оцени ефекта от промяната върху потребителите и с достатъчна потребителска бройка, практически да се измери дори много малък положителен или негативен ефект. По принцип, повече анализираща мощ може да се постигне, като се променят няколко неща на веднъж по неконтролиран (случаен) начин, и да се проведе стандартен многопроменлив статистически анализ, като множествена линейна регресия. На практика, обаче, А/В тестването е широко използвано, защото е лесно за разбиране и обяснение на ръководството.

8.6 Резюме на извлечения документ (summary)

Избирайки или класирайки по дадени критерии документи, отговарящи на дадена заявка ние искаме да представим списък с резултати, който би бил полезен за потребителя. В много от случаите, потребителят няма да иска да проучва всички върнати документи. По тази причина, ние искаме да направим резултатите достатъчно информативни, така че потребителят да може да направи крайно отсяване на документите, въз основа на нужната му информация. Стандартен начин за това е, да осигури фрагмент, кратко описание на документа, който е проектиран така, че да позволява на потребителя да вземе решение. Обикновено, откът състои от заглавието на документ и кратко резюме, което автоматично е извлечено. Въпросът е, как да проектираме резюмето, така че да се увеличи неговата полезност за потребителя. Двата основни вида резюмета са

- статични, които винаги са едни и същи за запитването
- динамични, които са пригодени според нуждата от информация на потребителя и се опитват да обяснят защо определен документ е извлечен за заявката.

Статичното резюме обикновено състои от едното или и двете подмножества от документи и метаданни, свързани с документа. Вместо на зони, за даден документ можем да използвате метаданни, които са свързани със съответния документ. Това може да е алтернативен начин, например, за предоставяне на автор или дата, или може да включва елементи, които са предназначени да дават резюме.

Динамичните резюмета показват един или повече по-нататъшни „прозорци“ на документа, с цел представяне на елементи, които са най-добрата помощна програма на потребителя при оценяването на документа, с оглед на нуждата му информация. Ако заявката се намира във фраза, копия на тази фраза в документа ще бъде показана като резюме. В противен случай ще бъдат избрани прозорци в документа, които съдържат няколко термини. Динамичните резюмета, обикновено се разглеждат като значително подобряване на използваемостта на IR системи, но те усложняват дизайна.

От една страна не може да бъде преизчислено динамично резюме, но от друга страна, ако една система има само позиционен индекс, тогава той не може лесно да възстанови контекста в основата на търсенето, за да се генерира динамично обобщение. Това е една от причините за използване на статични резюмета. Една система сканира даден документ, който е на път да се появи в списъка на показаните резултати, за да намери откъси, съдържащи думи за търсene. Това генериране трябва да бъде бързо, тъй като системата обикновено генерира много откъси за всяка заявка за която отговаря. Вместо да се копира целия документ, се копират само тези, които са фиксиран размер в документа. Когато имаме кратки документи е копиран целият документ, а ако даден документ е бил актуализиран, тъй като той последно е бил обработен, тези промени ще бъдат в кеппа, а не в индекса.

9 Обратна връзка и разширяване на заявката

В повечето колекции, едно и също понятие може да се отнася към различни думи. Този въпрос, известен като синонимика, има влияние върху отзоваването на повечето Системи за Извличане на Информация. Например, ако искате да потърсите думата устройство, която да наподобява думата *скелет*(но само в справки, които се отнасят към смисъл на конструкция, а не начовешки скелет), и при търсене на думата *термодинамика*, то тя да съответства на думата *горещина* в съответните обсъждания. Потребителите често се опитват да се справят с този проблем сами като ръчно подобряват заявката, както посочихме в Раздел 1.4; в този раздел ще обсъдим начини, по които системата може да помогне за подобряване на заявката, или изцяло автоматично или с участие на потребителите.

Методите за справяне с този проблем се разделят на два основни класа: глобални методи и локални методи. Глобалните методи са техники за разширяване или преформулиране на изразите в заявката, независими от самата заявка и върнатите от нея резултати, такива че промени в подредбата на думите в заявката биха довели до това заявката да съответства на други семантично подобни изрази.

Глобалните методи включват:

- Разширяване на заявка/преформулиране в подробен речници или WordNet (Раздел 9.2.2)
- Разширяване на заявкачрез създаване на речник (Раздел 9.2.3)
- Техники като поправяне на правописни грешки (обсъдени в Раздел 3)

Локалните методи коригират заявка свързана с документите, които първоначално изглеждат като отговарящи на заявката. Основните методи са:

- Обратна връзка с приложимост (relevance feedback) (Раздел 9.1)
- Псевдо-обратна връзка с приложимост(pseudo relevance feedback), позната като Сляпа обратна връзка с приложимост(blind relevance feedback) (Раздел 9.1.6)
- (Глобална) косвена обратна връзка с приложимост (Раздел 9.1.7)

В тази глава ще споменем всички тези подходи, но ще се концентрираме върху Обратната връзка с приложимост, който е един от най-често ползвани и най-успешни подходи.

9.1 Обратната връзка и псевдо-обратната връзка

Идеята на *обратната връзка с приложимост* (relevance feedback=RF) е да включи потребителя в процеса на извлечане на информация, с цел да се подобри колекцията получена за краен резултат. По-специално, потребителят дава обратна връзка относно приложимостта на документите в първоначалната колекция от резултати. Основният похват е следният:

- Потребителят пуска (кратка, проста) заявка;
- Системата връща първоначална колекция от извлечени резултати;
- Потребителят маркира някои от получените документи като приложими или неприложими;

- Системата изчислява по-добро представяне на информацията на база на обратната връзка, дадена от потребителя;
- Системата показва преработената колекция от резултати;

Обратната връзка с приложимост може да премине през едно или много повторения от тъкъв вид. Процесът разработва идеята, че е вероятно да е трудно да се образува добра заявка, когато нямаш добри познания за колекцията, но е лесно да се прецени по вече определени документи. В този случай има смисъл да се включи повторящо се подобряване на заявката от тъкъв тип. При тъкъв сценарий, обратна връзка с приложимост може да е резултатен за проследяване на развитието на информационните нужди на даден потребител: виждането на даден документ може да доведе до подобряване на разбирането на потребителя за точната информация, от която имат нужда. Търсенията на изображения предоставя добър пример за Обратната връзка с приложимост. Не само е по-лесно да се видят реалните резултати, но също така е област, в която е по-лесно да се отбележат приложимите и неприложимите изображения. След като потребителя въвежде първоначалната заявка за думата колело (bike) в демонстрационната система на страницата:

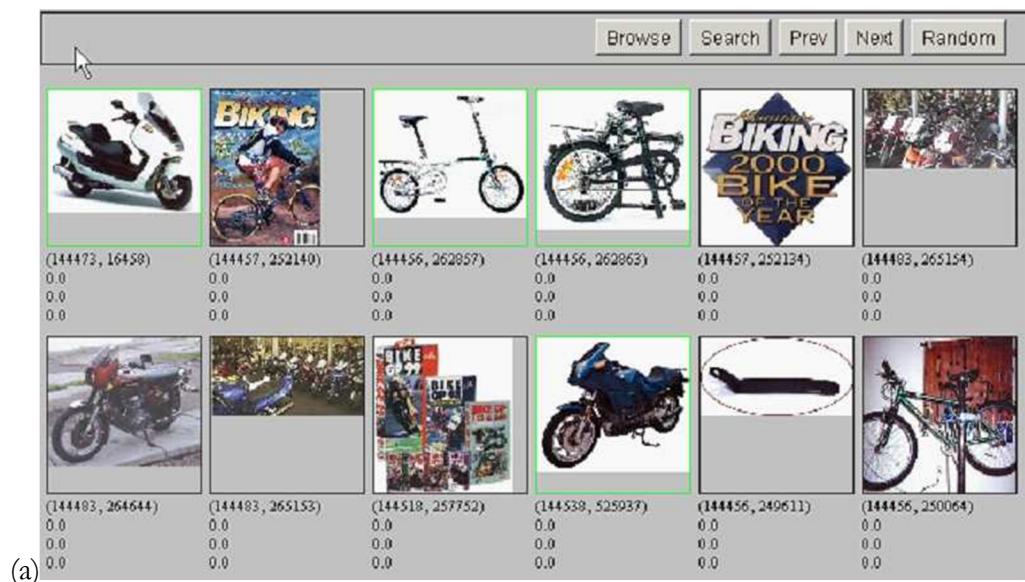
<http://nayana.ece.ucsb.edu/imsearch/imsearch.html>

са получени първоначалните резултати (в този случай, изображения). На Фигура 9.1 (a), потребителят е изbral някои от тях като приложими. Те ще бъдат използвани, за да се подобри заявката, докато останалите показани резултати нямат ефект върху преформулирането. След това Фигура 9.1 (b) показва новите най-добри резултати, изчислени след последната обратна връзка с приложимост.

Фигура 9.2 показва пример за текстово извличане на информация, при което потребителят би искал да открие повече информация относно приложението на космическите спътници.

9.1.1 Алгоритъмът на Rocchio за Обратната връзка

Алгоритъмът на Рохио (Rocchio Algorithm) е класически алгоритъм, прилаган при изпълняване на Обратна връзка с приложимост. Той моделира начинът, по който информацията от Обратната връзка с приложимост се въвежда във векторния пространствен модел на Раздел 6.3.



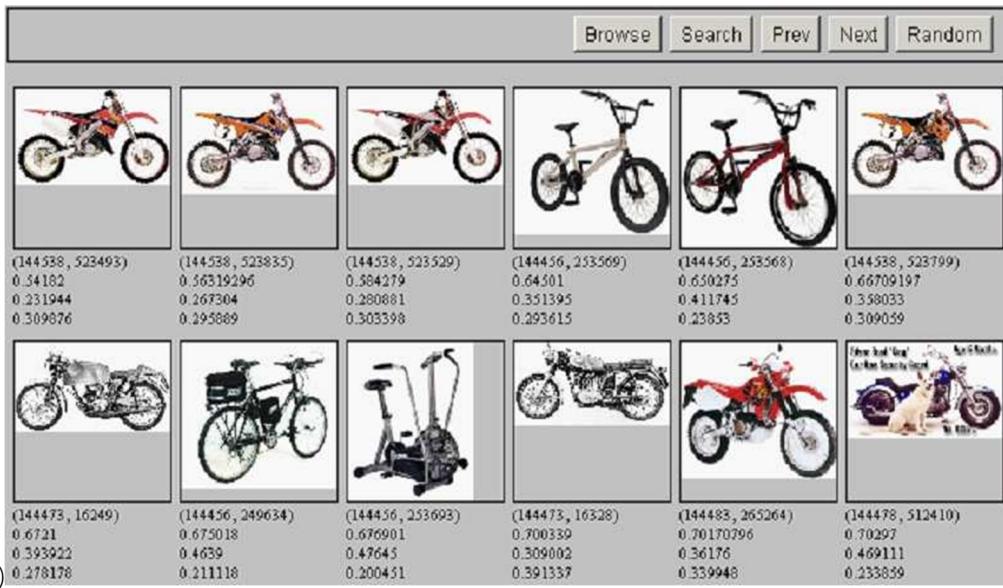


Figure 9.1 Обратнавръзка с приложимост върху търсene на изображения. (a) Потребителят разглежда резултатите от първоначалната заявка заколело (bike), избира първи, трети и четвърти резултат от най-горния ред и четвъртия резултат от най-долния ред като приложими, и ги въвежда като обратна връзка.(b) Потребителят вижда преработената колекция от резултати. Точността на заявката е значително подобрена. Извадката е от вече несъществуващата страница <http://nayana.ece.ucsb.edu/imsearch/imsearch.html> (Newsam и колектив 2001).

(a) Заявка: New space satellite applications(Нови приложения на космическите спътници)

(b)

- +1. 0.539, 08/13/91, NASA Hasn't Scrapped Imaging Spectrometer
- +2. 0.533, 07/09/91, NASA Scratches Environment Gear From Satellite Plan
- 3. 0.528, 04/04/90, Science Panel Backs NASA Satellite Plan, But Urges Launches of Smaller Probes
- 4. 0.526, 09/09/91, A NASA Satellite Project Accomplishes Incredible Feat: Staying Within Budget
- 5. 0.525, 07/24/90, Scientist Who Exposed Global Warming Proposes Satellites for Climate Research
- 6. 0.524, 08/22/90, Report Provides Support for the Critics Of Using Big Satellites to Study Climate
- 7. 0.516, 04/13/87, Arianespace Receives Satellite Launch Pact From Telesat Canada
- +8. 0.509, 12/02/87, Telecommunications Tale of Two Companies

© 2.074 new	15.106 space
30.816 satellite	5.660 application
5.991 nasa	5.196 eos
4.196 launch	3.972 aster
3.516 instrument	3.446 arianespace
3.004 bundespost	2.806 ss
2.790 rocket	2.053 scientist

2.003 broadcast

1.172 earth

0.836 oil

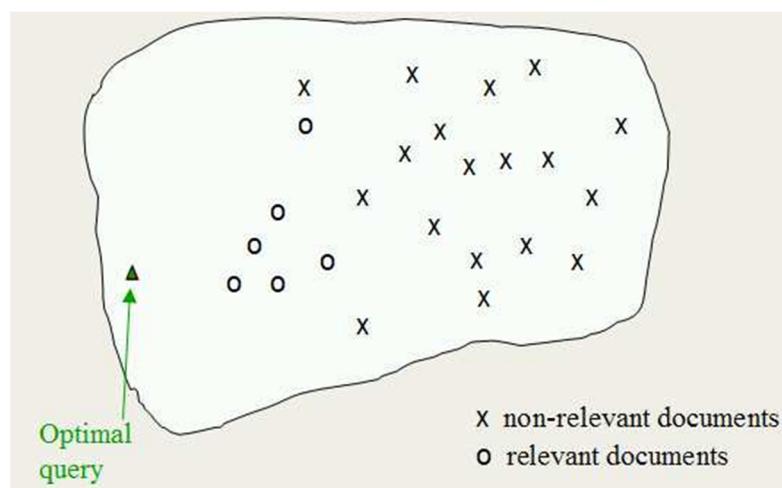
0.646 measure

(d)

- *1. 0.513, 07/09/91, NASA Scratches Environment Gear FromSatel- lite Plan
- *2. 0.500, 08/13/91, NASA Hasn't Scrapped Imaging Spectrometer
- 3. 0.493, 08/07/89, When the Pentagon Launches a Secret Satellite, Space Sleuths Do Some Spy Work of Their Own
- 4. 0.493, 07/31/89, NASA Uses 'Warm' Superconductors For FastCircuit
- *5. 0.492, 12/02/87, Telecommunications Tale of Two Companies
- 6. 0.491, 07/09/91, Soviets May Adapt Parts of SS-20 Missile ForCommercial Use
- 7. 0.490, 07/12/88, Gaping Gap: Pentagon Lags in Race To Matchthe Soviets In Rocket Launchers
- 8. 0.490, 06/14/90, Rescue of Satellite By Space Agency To Cost \$90Million

[△] **Фигура 9.2** Пример за обратнавръзка с приложимост върху текстови колекции.

(a) Първоначалната заявка (b) Потребителят отбелязва някои приложими документи (отбелязани със знакът +) (c) Разширяваме заявката с 18 израза с показаната тежест. (d) След прилагане се появяват преработените най-добрни резултати. Знакът * отбелязва кои от тях са счетени за приложими във фазата на получаване на обратна връзка.



[△] **Фигура 9.3** Оптимална заявка на Rocchio за разделяне на приложими (relevant) от неприложими (nonrelevant) документи.

(9.1)

(9.2)

Основната теория. Искаме да намерим векторът на една заявка, отбелязан като q , който възможно най-много увеличава подобността с приложимите документи и който възможно най-

много намалява подобността с неприложимите документи. Ако C_r е колекцията от приложими документи и C_{nr} е колекцията от неприложимите документи, то тогава ние търсим:¹

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

Където sim е дефинирана в Уравнение 6.10. Под косинусоидното подобие, оптималния вектор \vec{q} от разделяне на приложими от неприложими документи е:

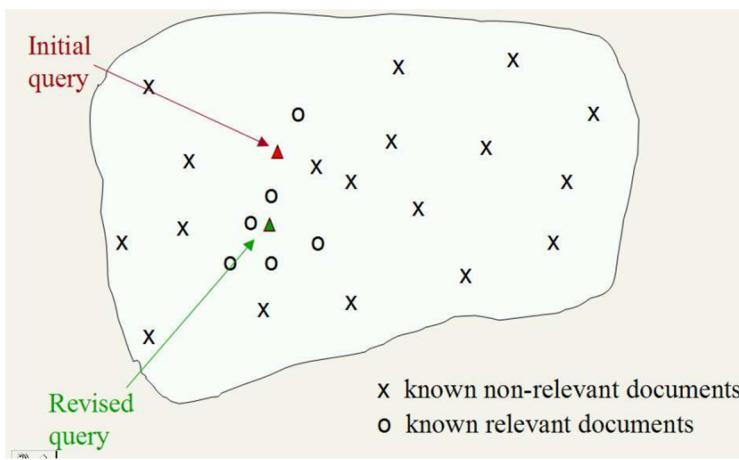
$$\vec{q}_{opt} = \frac{1}{|C_r|} \sum_{\vec{d}_j \in C_r} \vec{d}_j - \frac{1}{|C_{nr}|} \sum_{\vec{d}_j \in C_{nr}} \vec{d}_j$$

Това означава, че оптималната заявка е векторната разлика между инерционните центрове на приложимите и неприложимите документи; виж Фигура 9.3. Това наблюдение обаче не ни е особено полезно, точно понеже пълната колекция от приложими документи не е известна: тя е това, което се стремим да открием.

1. В това уравнение, $\arg \max_x f(x)$ връща стойност x , която възможно най-много увеличава стойността на функцията $f(x)$. Подобно, $\arg \min_x f(x)$ възможно най-много намалява стойността на функцията $f(x)$.

Алгоритъмът на ROCCHIO

Алгоритъмът на Рокио (Rocchio, 1971). Това е механизъмът на обратнавръзка с приложимост, която е представена и популяризирана от SMART системата на Салтън (Salton) около 1970 га година.



[▲] **Figure 9.4** Приложение на алгоритъмът на Rocchio. Някои документи са били отбелязани като приложими или неприложими и първоначалната заявка е била преместена, в зависимост от обратната връзка

(9.3)

В контекста на една истинска заявка за Извличане на информация, имаме потребител и частично познание за известни приложими и неприложими документи. Алгоритъмът предлага използването на подобрената заявка q_m :

$$\vec{q}_m = \alpha \vec{q}_0 + \beta \frac{1}{|D_r|} \sum_{\vec{d}_j \in D_r} \vec{d}_j - \gamma \frac{1}{|D_{nr}|} \sum_{\vec{d}_j \in D_{nr}} \vec{d}_j$$

където q_0 е първоначалният вектор на заявката, D_r и D_{nr} са съответните колекции от приложими и неприложими документи, а α , β , и γ са тежестите съответстващи на всеки израз. Те управляват равновесието между доверяването на вече оценената колекция от документи и заявката: ако имаме много вече оценени документи, то бихме искали да имаме по-високи стойности на β и γ . Започвайки от q_0 , новата заявка ни премества малко по-близо към инерционния център на приложимите документи и малко по-далеч от инерционния център на неприложимите документи. Тази заявка може да бъде използвана за извлечение на Стандартния векторно-пространствен модел (вж Раздел 6.3). Съвсем спокойно можем да оставим положителния квадрант от векторното пространство, като извадим вектора на неприложимите документи. При алгоритъма на Rocchio, отрицателните тегла на изразите не се взимат под внимание. В този смисъл, такива тегла се фиксираят със стойност 0. Фигура 9.4 показва ефекта от прилагане на обратна връзка с приложимост.

Обратнавръзка с приложимост може да подобри както отзоваването (повторното извикване или още recall), така и точността. Практиката обаче показва, че този подход е особено полезен при отзоваване, в случаите където самото отзоваване е от особена важност. Това от части е така, тъй като тази техника разширява заявката, но също от части е ефект от потребителския случай: когато искат високи стойности на отзоваване, от потребителите можеда се очаква да преразгледат резултатите и да повторят търсенето. Положителната обратна връзка също се оказва много по-ценна от отрицателната, така че повечето системи за Извличане на информация избират $\gamma < \beta$. Приемливи стойности могат да се изберат $\alpha = 1$, $\beta = 0.75$ и $\gamma = 0.15$. В интерес на истината, много системи, като например системата за търсене на изображения във Фигура 9.1, позволяват само положителна обратна връзка, което е равностойно на избиране $\gamma = 0$. Друга възможност е да се използват само отбелязаните като неприложими документи, които са получили най-големи стойности от системата за Извличане на информация като отрицателна обратна връзка (тук $|D_{nr}| = 1$ в Уравнение (9.3)). Докато много от резултатите от опити за сравнения на резултатите от различни варианти за обратна връзка с приложимост са доста неубедителни, то някои изследвания предполагат, че този вариант, наречен *Idedec-hi* е най-ефективният или поне най-постоянният като изгънливост

9.1.2 Вероятностна обратна връзка

Вместо да претеглят отново заявката във векторното пространство, ако потребителят ни е посочил някои приложими и неприложими документи, то ние можем да продължим с построяване на класификатор. Един начин това да бъде направено е чрез Наивен Бейс вероятностен модел (Naive Bayes probabilistic model). Ако R е булев номератор, изразяващ

приложимостта на даден документ, то ние можем да определим $P(xt = 1 | R)$, вероятността израз т да се появи в документ, в зависимост от това дали документът е приложим или не:

$$(9.4) P^*(xt = 1 | R = 1) = |V Rt| / |VR|$$

$$P^*(xt = 1 | R = 0) = (d ft - |V Rt|) / (N - |VR|)$$

където N е общият брой документи, $d ft$ е броят от документите съдържащи, $V R$ е колекцията от документи, за които е известно, че са приложими $V Rt$ е подколекцията ѝ съдържат. Въпреки, че колекцията от документи, за които е известно, че са приложими, е най-вероятно много малка част от всички приложими документи, ако приемем, че колекцията от приложими документи, за които е известно, е много малка част от колекцията на всички документи, то нашето предположение би било основателно. Това дава основа на друг начин на променяне на тежестта на изразите една заявка. Тези вероятностни подходи ще бъдат разгледани по-подробно в Раздел 11 и 13, и по-специално ще опишем приложението им към обратната връзка с приложимост в Раздел 11.3.4 (страница 228). За момента, с наблюденятията над този подход само над Уравнение (9.4) като основа за претегляне на изразите вероятно недостатъчно. Уравненията са само събирателна статистика и информация относно разпределението на изразите в документите, счетени за приложими. Те не пазят информацията от първоначалната заявка.

9.1.3 Кога обратната връзка с приложимост работи?

Успехът на обратната връзка с приложимост зависи от някои предположения. На първо място, потребителят трябва да има задоволителни познания, за да може да създаде първоначалната заявка, която да е сравнително близко до желаните от него документи. Това се изисква така или иначе за успешно извлечане информация дори във основния случай, но е от особена важност да се прегледат видовете проблеми, които обратната връзка не може да разреши сама. Случаи, в които обратната връзка сама по себе си не е достатъчна включват:

- Грешки в правописа. Ако потребител напише някой от изразите по начин, различен от този в документа от колекцията, то обратната връзка най-вероятно няма да е приложима. Това може да бъде отнесено към техниките за поправяне на правописа в Раздел 3.
- Междуезиково извлечане на информация. Документите на други езици не се намират близо при разпределение на изразите базирано на векторно пространство. По-точно, документи на един и същи език се групират по-близо един до друг.
- Несъответствие между речникът на търсещия потребител и речникът на колекцията. Ако потребителят потърси думата *laptop*, а всички документи използват изразите *notebook* и *computer*, то тогава заявката би била неуспешна и обратната връзка най-вероятно няма да е ефективна.

На второ място, подходът с обратна връзка изисква приложимите документи да бъдат близки един към друг, т.е. да се групират. В идеалния случай разпределението на изразите в приложимите документи ще бъде близко до това в документите, отбелаяни от потребителите, докато разпределението във всички неприложими документи ще бъде различно от това в приложимите документи. Всичко би сработило добре, ако всички приложими документи са групирани нагъсто около един единствен прототип, или поне ако има различни прототипи, ако приложимите документи имат значително застъпване на речниците, но приликите между

приложими и неприложими документи е малка. Безусловно моделът за обратна връзка на Rocchio се отнася към приложимите документи като към самостоятелна група, която моделира чрез инерционния център на групата. Този подход не работи толкова добре, ако приложимите документи са от разнороден клас, т.е. ако се състои от няколко групи документи във векторното пространство. Това може да се случи, когато:

- Подколекциите от документи, използвани различни речници, като *Бирма спрямо Мианмар*
- Заявка, за която резултатите като колекция по своята същност се противопоставят едни на други, като например: Звезди, които никога са работили в Burger King.
- Примери за общо понятие, коео то често се появява като противоречие на по-точно определените понятия, например *felines*.

Често решение на този проблем може да се намери с добро редакционно съдържание. Например, статия върху становищата на различни групи хора относно положението в Бирма, може да въведе терминология, използвани от различните партии, като по този начин свързва различните групи документи.

Обратната връзка не винаги е особено популярна сред потребителите. Често потребителите не са склонни да предоставят категорична обратна връзка или да удължават времето си за търсене по какъвто и да бил начин. Още повече, често е по-трудно да се разбере защо определен документ е изведен след използването на обратна връзка с приложимост.

Обратната връзка в приложимост може да има и чисто практически ориентирани проблеми. Дългите заявки, генериирани от прилагането на недвусмислените техники на обратната връзка с приложимост са безполезни за една типична система за извлечане на информация. Резултатът от това е висока изчислителна стойност за извлечането и вероятно удължено време за отговор на заявката към потребителя. Частично решение на проблема е да се претеглят само някои определени по-важни изрази в приложимите документи, като например първите 20 най-често използвани израза. Някои експериментални резултати също предполагат, че използването на ограничен брой изрази би довело до по-добри резултати (Harman 1992) докато други трудове предполагат, че използването на повече изрази би довело до по-добро качество на извлечената информация (Buckley и колектив 1994b).

9.1.4 Обратна връзка с приложимост в мрежата

Някои търсачки в интернет предлагат функционалност за търсене на подобни/свързани страници: потребителят отбелязва определен документ от колекцията с резултати като примерен от гледна точка на покриване на необходима информация и правят заявка за повече документи като него. Това може да бъде разгледано като особено прости обратни връзки с приложимост. Въпреки това, като цяло обратната връзка рядко се използва при търсене в мрежата. Едно изключение е търсачката Excite, която първоначално предоставя пълна обратна връзка с приложимост. Поради неизползваемост на това свойство обаче то бива премахнато. Малко хора в мрежата използват интерфейс за търсене за напреднали, като повечето предпочитат да приключват търсенията си от първата интеракция. Липсата на разбиране се дължи вероятно и на други две причини: обратната връзка е трудна за обяснение на средностатистическия потребител и е предимно стратегия за подобряване на отзоваването, а потребителите, които търсят в мрежата рядко се интересуват от получаване на задоволително

отзоваване. Spink и колектив(2000) представят резултати от използването на обратна връзка в търсачката Excite. Само около 4% от потребителските сесии със заявки са използвали опцията за обратна връзка и те в повечето случаи са използвали кратката връзка до всеки резултат за търсене на „Повече като този“ резултати. Около 70% от потребителите са прегледали само първата страница от резултатите и не са продължили към другите резултати. За хората, които все пак са използвали обратна връзка с приложимост, резултатите са били подобрени в около 2/3 от случаите.

Важен нов труд работи върху използването на т.нр. clickstreamdata

(или върху кои кратки връзки цъка потребителят), за да предостави косвена обратна връзка с приложимост. Използването на такъв тип данни е изследвано в подробности при Joachims 2002b, Joachims и колектив 2005.

Използването на особено полезната структура на връзките в мрежата (виж Раздел 21) може също да бъде разгледан като нейна обратна връзка, но предоставена по-скоро от авторите на документа, отколкото от неговите читатели (макар че на практика повечето автори са и читатели).

9.1.5 Оценка на стратегиите за обратна връзка

Интерактивната обратна връзка може да допринесе със значителна полза за производителността на извлечането на информация. Еднократно използвана обратна връзка е изключително полезно. Докато повторно използване може да не допринесе кой знае колко повече. Успешната употреба на обратната връзка изисква достатъчно количество оценени документи, в противен случай процесът може да се окаже нестабилен и може да се отклони от потребителските информационни нужди. Съответно се препоръчва да се разполага с поне 5 оценени документа.

Има някои тънкости при оценяването на ефективността на обратната връзка. Най-първата и очевидна стратегия е да започнем със запитване q_i и да изчислиш *графиката прецизност-отзоваване* (*Прецизност – съотношението (получени резултати) / (значещи резултати), *Отзоваване – съотношението (получени значещи резултати) / (всички значещи резултати)). След първата вълна на обратна връзка с клиента, презчисляваме запитването q_m (модифицирано запитване) и отново пресмятаме *прецизност-отзоваване* графиката. Така правим постоянна оценка на получената информация, спрямо цялата налична, получавайки сравнително праволинейни данни. Правейки това, ще открием значителни ползи от гледна точка на обратната връзка - ръст на средната точност приблизително 50%. За съжаление обаче, това не е съвсем вярно - този ръст се дължи главно на факта, че документите, посочени като приложими от потребителя, биват оценявани по-високо. Коректността изисква да оценяваме само документи, които не са прегледани от клиента.

Друга идея е за повторното извикване да ползваме документи от т.нр. „остатъчна колекция“ (всички документи без тези които са обявени вече за приложими), което изглежда като по-реалистична оценка. За съжаление, замерванията на производителността в този случай могат да бъдат по-ниски дори и от първоначалните. Това е точно случаят, когато има само няколкоприложими документа и значителна част от тях е вече прегледана от клиента по време на първия етап. Относителната производителност на вариантните методи за обратна връзка

може да бъде сравнена, но е трудно това да бъде направено коректно, защото размерът на колекцията от документи и броят на приложимите такива се променя по време на процеса.

Поради тези причини, никой от двата метода не е напълно задоволителен. Трети метод е да вземем две колекции, една от които е използвана в първоначалната заявка и оценка на приложимостта, а другата в последствие се използва за сравнителен анализ. Тогава производителността на q_0 и q_m може да бъде коректно сравнена чрез втората колекция.

Може би най-добрата оценка на значимостта на обратната връзка е да се правят проучвания на потребителя относно нейната ефективност, в частност правейки времево-базирани сравнения: колко бързо клиент намира документи чрез използването на обратна връзка или друга стратегия (като например промяна на запитването) или алтернативно, колко значими документа един клиент намира за даден период от време. Тези статистики са най-справидливи или близки до реалната употреба на системата.

	Прецизност при $k = 50$	
Претегляне на ползите	Без ОВ	Псевдо ОВ
1	64.2%	72.7%
2	74.2%	87.0%

⁴ **Figure 9.5** Резултатите показват как псевдо-обратна връзка може да подобри ефективността. Тези резултати са взети от Cornell SMART system, TREC 4 (Buckley et al. 1995), като ярко различават използването на две различни нормализиращи схеми (L срещу l); виж Фигура 6.15 (стр. 128). Псевдо-обратната връзка се състои в добавяне на 20 изразакъм всяка заявка.

9.1.6 Псевдо-обратна връзка

ПСЕВДО-ОБРАТНА ВРЪЗКА СЛЯПА ОБРАТНА ВРЪЗКА

Псевдо-обратната връзка, или още „*сяпта*“ обратна връзка, представлява метод за автоматичен анализ на локално ниво. Служи за автоматизиране на ръчната част от обратната връзка, така че клиентът да получи по-добро обслужване без допълнително взаимодействие. Методът представлява нормално извлечане на данните с цел да се намери първоначален набор „най-приложими“ документи, след което да се *предположи*, че първите *кот* тях са приложими и да се даде обратна връзка без да се взема предвид това предположение.

В по-голямата си част от случаите тази техника работи. Доказателствата сочат, че има има склонност да работи по-добре от глобалните анализи (Раздел 9.2), също са измерени и подобрения в ефективността. Но не трябва да пропускаме и опасностите от автоматичния процес. Например, ако запитването е за „*медни мини*“ и най-първите резултати са за мини в Чили, тогава може да има тенденция за представяне на документи свързани с Чили.

9.1.7 Косвена обратна връзка

НЕЯВНА ОБРАТНА ВРЪЗКА

Също така можем да ползваме косвени източници на доказателства, пред определена обратна връзка като основа на обратната връзка с приложимост. Това често се нарича „*неявна обратна*

връзка“. Косвената обратна връзка е по-малко надеждна от прямата, но тък е по-полезна от псевдо обратната връзка, която не съдържа никакви доказателства за потребителската оценка. Предвид че потребителите често не желаят да предоставят явна обратна връзка, така е лесно да се събере косвена информация в големи количества при системи с голяма посещаемост, като интернет търсачки.

Също така можем да ползваме косвени източници на доказателства, пред CLICKSTREAM MINING - определена обратна връзка като основа на обратната връзка с приложимост. Също така можем да ползваме косвени източници на доказателства, пред определена обратна връзка като основа на обратната връзка с приложимост. Това често се нарича „нейвна обратна връзка“. Косвената обратна връзка е по-малко надеждна от прямата, но тък е по-полезна от псевдо обратната връзка, която не съдържа никакви доказателства за потребителската оценка. Предвид че потребителите често не желаят да предоставят явна обратна връзка, така е лесно да се събере косвена информация в големи количества при системи с голяма посещаемост, като интернет търсачки.

В мрежата, DirectHit представиха идеята за по-високо оценяване на документи, които биват по-често преглеждани от потребителите. С други думи, връзки, които се отварят по-често се считат за по-свързани със запитването. Този подход прави редица предположения, като например резюмето на документа, което е представено към описанието на връзката (на база на него потребителят решава дали да отвори връзката или не). В оригиналната DirectHit търсачка, данните за кликовете по страниците са събирани в световен мащаб, а не според конкретният потребител или заявка. Това е една по-обща форма на *clickstream mining* или извличане на информация от кликовете. Днес, подобен подход се използва при оценяването на реклами, според заявката за търсене (Раздел 19).

9.1.8 Резюме

Обратната връзка с приложимост се е показвала като много значим метод за повишаване на приложимостта на резултатите от заявката. Успехът на нейната употреба изисква заявки, за които наборът от приложими документи е среден до голям. Пълната обратна връзка често е тежка за потребител и нейното осъществяване е неефективно в повечето системи за извличане на информация. В много случаи, други типове интеракция могат да подобрят ефективността също толкова, с много по-малко работа.

Отвъд специфичните сценарии за добиване на информация, някои други възможни употреби са:

- Следене на изменяща се нужда от информация (например следене на промяната на интереса при моделите коли през времето)
- Поддържане на информационен филтър (news feed – често сменяни актуални новини)
- Активно учене (например подходящ подбор на примери при обучението на ученици, така че да са нужни най-малко коментари за усвояването на материала)

9.2 Глобални методи за преформулиране на запитванията

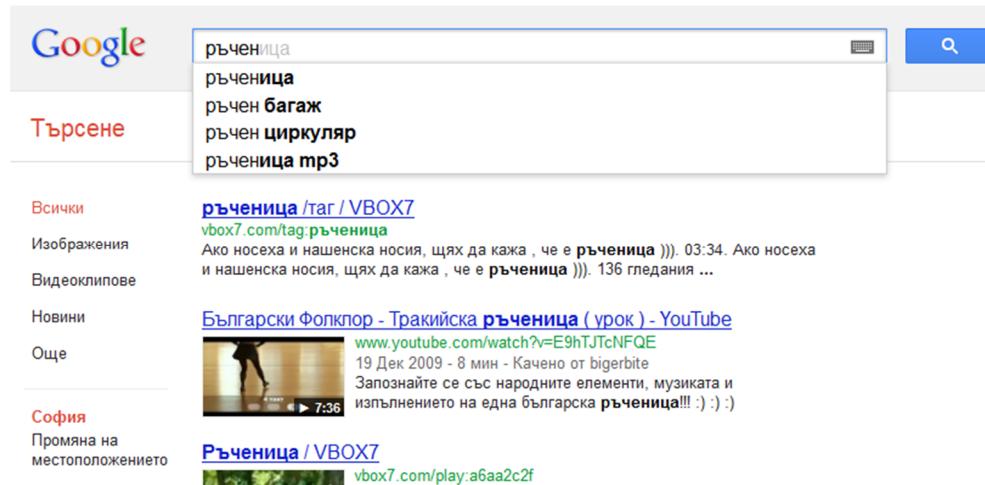
В този Раздел набързо ще дискутираме три глобални метода за разширяване на заявката: подпомагайки потребителят да го направи сам, използвайки ръчен речник и построявайки такъв речник автоматично.

9.2.1 Лексически спосobi за преформулиране на заявката

Различни методи могат да подпомогнат потребителя в процеса на търсене, показвайки му защо неговите запитвания не дават резултат. Това включва информация за пропуснати в заявката думи (понеже са в стоп списък), с кои думи те се коренуват (например „автомобилостроене“ -> „автомобил“, „строй“), брой попадения за всяка дума и фраза и дали някои думи динамично са преведени като фрази. Системите за извличане на информация също могат да предлагат фрази, подходящи за търсене, които са предварително заложени в речник. На потребителят също може да бъде даден избор измежду списък от изрази, с които да подобри търсенето.

9.2.2 Разширяване на заявката

При обратната връзка, потребителят дава допълнителна информация за документи (като ги маркира „приложими“ или не) и така предоставя информация, която служи за претегляне на изразите при запитване за документите. При „разширяването на запитването“, от друга страна, потребителят дава допълнителни думи или фрази, предлагайки и други изрази при търсенето. Някои търсачки (особено интернет-базирани) предлагат подобни заявки в отговор на първоначалната заявка от потребителя, давайки му лесни и бързи алтернативни възможности.



[▲] **Фигура 9.6** Пример за разширяване на заявка в търсачката Google през 2012год. Разширениите избори се появяват точно под текстът, който сме написали до момента.

Основният въпрос при тази форма на разширяване на запитването е как да създадем алтернативите (или разширениите заявки) за потребителя. Най-често срещаната форма е глобалният анализ, използвайки някакъв речник. За всеки термин *t* в запитването, то може да бъде автоматично разширено със синоними на *t* и думи свързани с *t*. Използването на такъв речник може да бъде комбинирано с идеи като тегло на изразите: например, някои изрази могат да бъдат „претеглени“ като по-тежки от първоначалните, други по-леки.

Методи за построяване на речник за разширяване на заявка са:

- Употребата на контролиран от хора речник. Тук всеки изразе именован канонично – Десетичната система на Дюи или Тематичната Заглавна Система на Конгресната Библиотека. Употребата на контролиран речник е доста често ползвана за добре организирани домейни. Добре познат пример е унифицираната система за медицински език (Unified Medical Language System – UMLS) използвана с MedLine за търсене на биомедицинска изследователска литература. Тя използва автоматично разширение на запитването и освен ако потребителят не реши да изследва предадената заявка, може и да не разбере, че тя е била разширена. Например на Фигура 9.7
- Заявка от потребителя: cancer
- PubMed заявка: (“neoplasms”[TIAB] NOT Medline[SB]) OR “neoplasms”[MeSH Terms] OR cancer[Text Word]
- Заявка от потребителя: skin itch
- PubMed заявка: (“skin”[MeSH Terms] OR (“integumentary system”[TIAB] NOT Medline[SB]) OR “integumentary system”[MeSH Terms] OR skin[Text Word]) AND ((“pruritus”[TIAB] NOT Medline[SB]) OR “pruritus”[MeSH Terms] OR itch[TextWord])

⁴ **Фигура 9.7** Пример за разширяване на заявка чрез речникът на PubMed. Когато потребителят въведе заявка през интерфейса на PubMed към Medline на страницата <http://www.ncbi.nlm.nih.gov/entrez/>, заявката се завежда през речникът на Medline както е показано.

- Ръчен речник. Тук, хората редактори трябва да изградят набор от синоними за определени думи и изрази, без да определят каноничен термин. Statistics Canada поддържа речник от предефинирани термини, синоними, термини с по-широко или по-тясно значение за теми, по които правителството събира статистическа информация, като например стоки и услуги. Този речник е двуезичен – Английски и Френски.
- Автоматичен речник на произлизашите термини. Тук, статистики за съвпадение и съвместно наличие на думи над колекция от документи, служат за автоматично построяване на речник. (виж Раздел 9.2.3)
- Преформулиране на запитването на базата на извлечения от предишни заявки. Тук използваме ръчните реформулировки на предните потребителите, за да направим предложение на текущия. Този способ изисква голям обем от запитвания и за това е подходящ при уеб-базираните търсачки.

Разширяването на запитването на базата на речник има предимството да не изисква допълнителна входяща информация от потребителя. Ползването на този метод увеличава рейтингът на „отзоваване“ и се използва значително в много изследователски и инженерни области. Освен от глобалните аналитични способи, разширяване на запитването може да бъде ползвано и в локални анализи, например при анализирането на документи измежду резултатите. Обикновено се изисква потребителска намеса, но остава разликата между това потребителят да дава оценка относно документите които търси или запитването което отправя.

9.2.3 Автоматично генериране на речник

Като алтернатива на скъпо-струващите речници поддържани от хора, можем да опитаме да създадем автоматичен такъв, анализирайки колекция от документи. Има два основни подхода – единият е просто да се изследва многократното повторение на думите. Казваме, че думи, които се повтарят в един документ или дори параграф от него, имат по-висок шанс да са близки или свързани по смисъл и просто броим статистически повторенията, намирайки най-близките думи. Другият подход включва използването на повърхностен граматически анализ на текста, целящ изследването на граматическите връзки и зависимости. Например, казваме че обекти които са „отгледани“, „сгответни“, „изядени“ и „погълнати“ имат по-голям шанс да са някакъв вид храна. Употребата на първият подход има по-малък шанс за грешки, тъй като се избягват грешки при разбора, но пък граматическият подход е по-точен.

Най-простият начин да построим речник на повторенията е като използваме прилики израз-израз. Започваме с матрица на изрази в документа A , където всяка клетка $A_{i,d}$ е претеглен брой $W_{i,d}$ за израза i в документа d , със тегло A , така че A имаме дължинно нормализирани редове. Ако след това изчислим $C = A * A^T$, тогава $C_{i,v}$ е оценка на сходството между i и v (по-голямо число, по-добре).

Фигура 9.8 показва речник, направен по този начин с намаляне на размерността чрез Скрито Семантично Индексиране (виж Раздел 18)

Дума	Най-близки съседи
Absolutely	absurd, whatsoever, totally, exactly, nothing
Bottomed	dip, copper, drops, topped, slide, trimmed
Captivating	shimmer, stunningly, superbly, plucky, witty
Doghouse	dog, porch, crawling, beside, downstairs
Makeup	repellent, lotion, glossy, sunscreen, skin, gel
Mediating	reconciliation, negotiate, case, conciliation
Keeping	hoping, bring, wiping, could, some, would
Lithographs	drawings, Picasso, Dali, sculptures, Gauguin
Pathogens	toxins, bacteria, organisms, bacterial, parasite
Senses	grasp, psyche, truly, clumsy, naive, innate

[▲] **Фигура 9.8** Пример за автоматично генериран речник, базиран на труда на Schütze(1998), който използва скрито семантично индексиране.

Може да се забележи, че някои попадения са добри, докато други имат малък или дори отрицателен принос. Качеството на асоциациите обикновено е проблем при този подход. Двусмислието на термините също създава незначими, но статистически свързани данни. Например търсенето на “Apple computer”, може да се разшири до “Apple red fruit computer”. Като цяло, тези речници страдат от „фалшиви позитивни“ и „фалшиви негативни“

результати. Освен това, изразите от автоматичния речник са силно свързани в документите (често колекцията използвана за съставянето на речника е съща като индексираната), този вид разширение на запитването може да не получи много допълнителни резултати.

Разширението на запитванията е често ефективно за увеличаване на отзоваването. За съжаление, има доста висока цена при ръчното производството на такива речници и последващото им опресняване за научни и терминологични развития в областта. Като цяло, общите речници дават твърде малко покритие на фона на богатите речници построени специално за контекста (например за някоя научна сфера). Също така, разширението на заявката може значително да намали прецизността, особено когато запитването съдържа неедносмислени изрази. Като цяло, разширението на запитването е по-малко успешно от обратната връзка с приложимост, може да е по-добро от псевдо-обратната връзка и има предимството да е по-разбираемо за системния потребител.

9.3 Справка и по-задълбочено четиво

Работата по извличане на информация бързо се сблъска с проблема на вариантните изрази, което означава, че думите в запитването може да не присъстват в документа, въпреки че той е свързан със самото запитване. Ранен експеримент около 1960, проведен от Суонсън ([Swanson, 1988](#)), показва че едва 11 от 23 документа, индексирани като тематично свързани с думата „токсичност”, съдържат срещания на думи с корен „токси”. Не рядко срещани са и проблемите с превода. Блерър и Марун ([Blair and Maron, 1985](#)) са заключили, че е „просто невъзможно трудно за потребителите да предвидят точния набор от думи и фрази, които се ползват от всички или повечето свързани документи”.

Основните първоначални книжа за обратна връзка с приложимост използвайки векторния пространствен модел се в трудовете на Салтън ([Salton, 1971b](#)), включвайки представянето на Алгоритъма на Рохио ([Rocchio 1971](#)) с Ide dec-hi варианта заедно с оценка на няколко варианти ([Ide 1971](#)). Друг вариант е се разглеждат всички документи в колекцията като неприложими, вместо да се разглеждат само тези, които изрично са оценени като неприложими. Schütze и колектив (1995) и Singhal и колектив (1997) обаче показват, че са получени по-добри резултати, когато се разглеждат документи близки до желаната заявка, а не всички документи. Други по-късно развити трудове включват [Salton и Buckley \(1990\)](#), [Riezler и колектив \(2007\)](#) (подход към обратната връзка със статистическа обработка на естествени езици) и най-скорошното проучване на [Ruthven и Lalmas\(2003\)](#).

Ефективността на интерактивните системи за обратна връзка е разисквана при [Salton 1989](#), [Harman 1992](#), [Buckley и колектив 1994b](#). [Koenemann и Belkin \(1996\)](#) провеждат потребителски изследвания на ефективността на обратната връзка с приложимост.

По принцип най-добрият речник между речниците на английски език е този на Роже ([Roget 1946](#)). При неотдавнашни изчислителни дейности, хората почти винаги използват WordNet ([Fellbaum 1998](#)), не само защото е безплатен, но и поради богатата структура от кратки връзки, с които разполага. WordNet може да бъде разгледан на: <http://wordnet.princeton.edu>.

[Qiu и Frei\(1993\)](#) заедно с [Schütze\(1998\)](#) обсъждат автоматичното създаване на речник. [Xu и Croft \(1996\)](#) проучват както локално, така и глобално разширяване на заявки.

10 XML извлечение на информация

Системите за извлечение на информация често се противопоставят на релационните бази от данни. Традиционно, тези системи извличат информация от *неструктуриран текст*, което означава „сиров“ текст, без никакви означения. Базите от данни са проектирани за търсене на *свързани данни*: множество от записи, които имат стойности за атрибути като брой на работници, заглавие и заплата. Има основни разлики между извлечането на информация и базите от данни, които се отнасят за модела на извлечение, структурата на данните и заявките, и са показани в талица 10.1¹.

Някои проблеми за търсене в силно структуриран текст се обработват ефективно с релационни бази от данни, например, ако таблицата на служителите съдържа атрибут, който накратко описва длъжността и вие искате да намерите всички служители, които се занимават с фактури. В този случай SQL заявката:

```
select lastname from employees where job_desc like ?invoic%;
```

може да е достатъчна да задоволи вашата информационна нужда със висока прецизност (precision) и отзоваване (recall).

Въпреки това, много структурирани източници на данни, съдържащи текст, са най-добре моделирани като структурирани документи, а не релационни данни. Наричаме търсенето над такива структурирани документи *структураторно извлечение* на информация. Заявките при структурираното извлечение могат да бъдат структурирани или неструктурни, но ние ще приемем в тази глава, че колекцията се състои само от структурирани документи. Приложението на структурираното извлечение включват цифрови библиотеки, патентни бази данни, блогове, текстове, в които личности и места са маркирани (наричаме го маркиране на именовани единици), и офис пакети (OpenOffice), които записват документи като маркират текст. Когато работим с тези приложения, ние искаме да сме способни да направим заявка, която комбинира текстов критерий със структурен критерий. Пример за такава заявка е: дай ми цялата дължина на статията с трансформация на Фурье (цифрови библиотеки), дай ми патентите, които имат RSA публичен ключ за криптиране.

В най-съвременните бази от данни, е възможно търсене на текст в текстови колони. Това обикновено означава, че е създаден обърнат индекс и булево или търсене във векторно пространство е възможно, като ефективно се комбинират ядрото на базата от данни и технологиите за извлечение на информация.

	RDB search	unstructured retrieval	structured retrieval
objects	records	unstructured documents	trees with text at leaves
model	relational model	vector space & others	?
main data structure	table	inverted index	?
queries	SQL	free text queries	?

Таблица 10.1 RDB (релационни бази от данни) търсене, неструктуррано извличане на информация и структурирано извличане на информация. Все още няма общо мнение кои методи работят най-добре за структурираното извличане на информация, въпреки че много изследователи смятат че Xquery (215стр.) ще се превърне в стандарт за структурирани заявки.

И това включва US патенти 4,405,829 (патента), или дай ми статии за разглеждане на забележителности във Ватикана и Колизеума (текст с маркиране на именовани единици). Това са структурирани заявки, на които не може да отговори добре система за некласирана информация. Такива модели като Булевия модел страдат от лошо отзоваване (recall). Например, една некласирана система ще върне голям брой статии, които споменават Ватикана, Колизеума и обиколка на забележителностите, без класиране на тези, които са най-подходящи за първата заявка. Много потребители са също пословично зле да посочат структурирани ограничения. Например, потребителите може да не знайт за кои структуриран елемент системата прави търсенето. В нашия случай, потребителят е несигурен дали заявката да бъде за страната Ватикана и забележителността Колизеум или за институцията Ватикана и сградата Колизеум. Потребителите също могат да бъдат напълно незапознати със структурираното търсене и усъвършенстваните интерфейси за търсене или не желаят да ги използват. В тази глава, ние наблюдаваме как методите за извличане на информация да се адаптират към структурираните документи за справяне с тези проблеми.

Ние ще разгледаме само стандарт за кодиране на структурирани документи: Extensible Markup Language или XML, който в момента е най-широко използван като стандарт. Няма да се спирате на спецификата, които отличават XML от други видове езици за маркиране като HTML и SGML.

В контекста на извличането на информация, се интересуваме от XML само като език за кодиране на текст и документи. Може би по-широкото използване на XML е да се кодират не-текстови данни. Например, ние може да искаме да изнесем данни в XML формат от система за планиране на ресурсите на компания и след това да ги прочетеме в програма за анализ на производството, която е необходима за презентация. Този вид приложение на XML се нарича *данни-ориентирано*, защото доминират числови и не-текстови атрибут-стойност данни и текстът обикновено е една малка част от общите данни. Повечето данни-ориентирани XML се съхраняват в бази от данни- за разлика от обрънатия индекс методи за текст-ориентирани XML приложения, които представяме в тази глава.

В тази глава наричаме XML извличането *структуррано извличане* на информация. Някои изследователи използват термина *полуструктурирано извличане*, за да разграничат XML извличането от заявките към бази от данни. Приехме терминологията, която е широко разпространена в XML извличане общността. Така например, стандартния начин да се прави XML заявка е структурирана заявка, а не полуструктуррана заявка. Терминът структурирано извличане се използва рядко за заявки към бази от данни и той винаги се свързва със XML извличане в тази книга.

Има и втори тип проблем на извличането на информация, който е междуинен между неструктурното извличане и заявките към релационни бази от данни: параметрично и търсене в зона, което обсъждахме в раздел 6.1 (стр 110). При модела от данни на параметрично и зоново търсене, има параметрични полета (релационни атрибути като *дата* или *размер на файл*) и зони –текстови атрибути, които вземат парче от неструктурирани текст като стойност.

Например, автор и заглавие на фиг.6.1 (стр.111). Моделът на данните е плосък, което означава, че няма влагане на атрибути. Броя атрибути е малък. Обратно, XML документите имат по-сложна дърводидна структура, която виждаме на фиг. 10.2, в която атрибутите са вложени. Броят на атрибутите и възлите е по-голям, отколкото в параметричното и търсенето по зона.

След като представи базовите концепции на XML в секция 10.1, тази глава ще дискутира първо предизвикателствата на XML извлечането (секция 10.2). После ще опишем векторно-пространствения модел за XML извлечане (секция 10.3). Секция 10.4 представя INEX, която е текущо най-важната част от XML извлечането на информация. Ще дискутираме разликите между данни-ориентирани и текст-ориентирани подходи за XML в секция 10.5.

10.1 Базови XML понятия

XML документът е подредено дърво, което има етикети. Всеки възел от дървото е един XML *елемент* и е написан с отварящ и затварящ се *tag*. Един елемент може да има един или няколко XML *атрибути*. В XML документа на фигура 10.1, елементът сцена е затворен между два тага <scene ...> и </scene>. Той има атрибут брой със стойност vii и два наследника, заглавие и сценарий.

Фигура 10.2 показва фигура 10.1 като дърво. Възлите на дървото се състоят от текст – Шекспир, Макбет, и Замъка на Макбет. Вътрешни възли на дървото кодират структурата на документа (заглавие, акт, и сцена) или метаданните (автор).

Стандартът за достъп и обработка на XML документите е XML Документен Обектен Модел (Document Object Model) или DOM. DOM представя елементите, атрибутите и текста, в рамките на элемента, като възли в дървото. Фигура 10.2 е опростено DOM представяне на XML документа от фигура 10.1². С DOM приложението, ние можем да обработваме един XML документ като започнем от корена и слезаме надолу по дървото от родител към наследник.

XPath е стандарт за изброяване на пътища в една XML документна колекция. Ние ще се отнасяме към пътищата като към XML контексти или просто контексти в тази глава. Само малко подмножество от XPath е необходимо за нашите цели. XPath изразът избира всички възли след това име. Последователните елементи на пътя са разделени с наклонени черти, така act/ scene избира всички елементи сцена, чийто родител е действие елемента. Двойна наклонена черта указва, че произволен брой от елементи може да се намесят по пътя: play//scene избира всички елементи сцена, които се срещат в елемента пьеса. На фигура 10.2 това множество се състои от един сцена елемент, които се намира на пътя пьеса, действие, сцена от корена. Наклонената черта стартира пътя от корена на дървото. /play/title избира заглавието на пьесата от фигура 10.1, /play//title избира множество от две заглавия (заглавие на пьесата и заглавие на сцената), /scene/title няма елементи. За удобство ние поставяме на крайния елемент от пътя символа #, въпреки че това не отговаря на стандарта на XPath. Например, title# "Macbeth" избира всички заглавия, съдържащи термина Макбет.

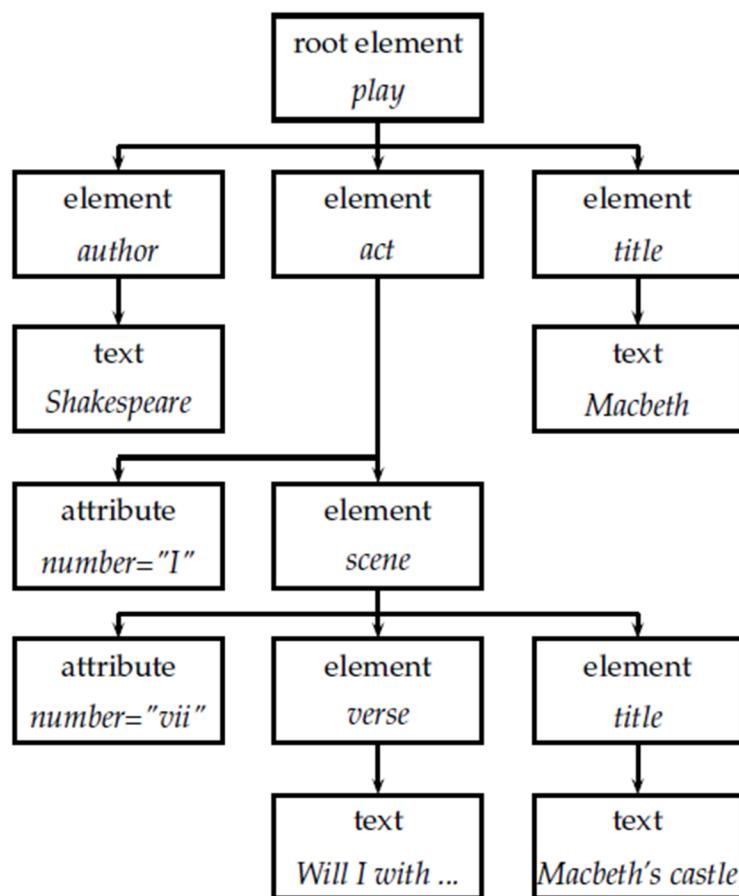
Ние също се нуждаем от понятието схема в тази глава. Схемата поставя ограничения за структурата на допустимите XML документи за конкретно приложение. Схемата за Шекспировите пьеси може да предвиди, че сцената може да настъпи като наследник на действието и че само действия и сцени имат атрибут брой. Два стандарта за схеми за XML документи са XML DTD (Document Type Definition) и XML Schema. Потребителите могат само

да пишат структурирани заявки в XML извличаща информация система, ако имат минимално знание за схемата на колекцията.

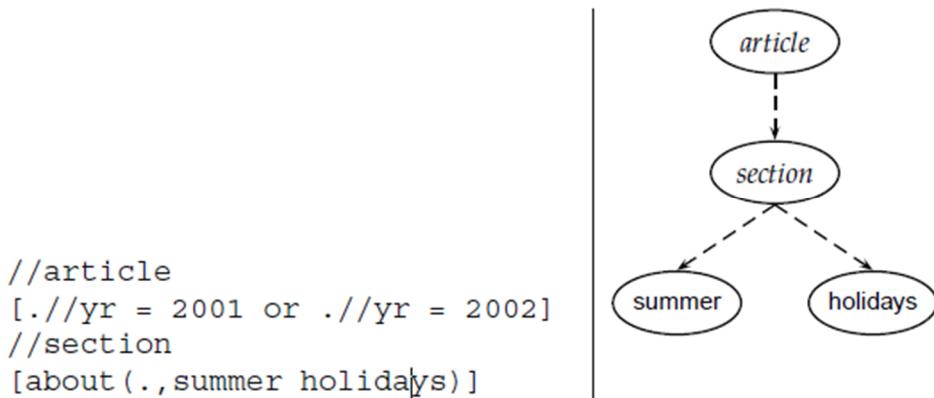
2. Представянето е опростено в редица отношения. Например, ние не показваме корена и текста не е заложен в текстовите възли. Вижте <http://www.w3.org/DOM/>.

```
<play>
  <author>Shakespeare</author>
  <title>Macbeth</title>
  <act number="I">
    <scene number="vii">
      <title>Macbeth's castle</title>
      <verse>Will I with wine and wassail ...</verse>
    </scene>
  </act>
</play>
```

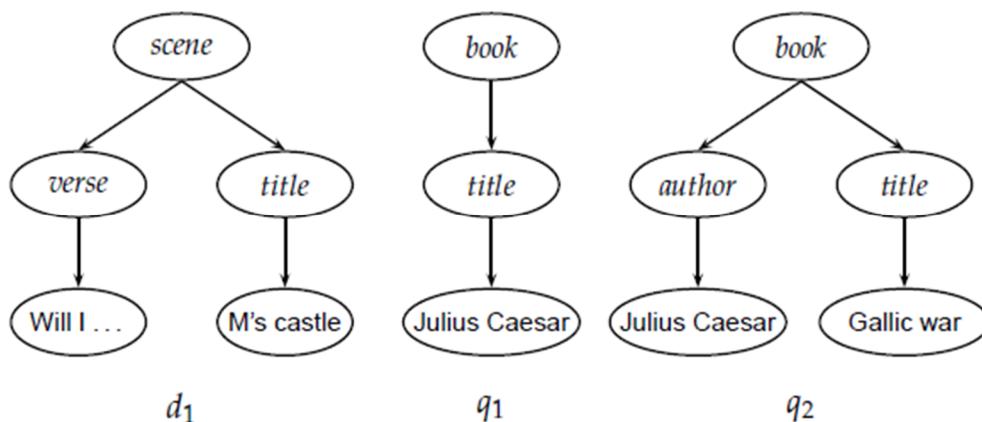
Фигура 10.1 показва един XML документ



Фигура 10.2 XML документът от фигура 10.1 като опростен DOM обект



Фигура 10.3 Една XML заявка в NEXI формат и частично представяне в дървото



Фигура 10.4 Дърво представяще XML документи и заявки.

Общ формат за XML заявки е NEXI (Narrowed NEXI Extended XPathI). Ще вземем примера от фигура 10.3. Ние представяме заявката на четири реда за удобство, но тя е предназначена за един ред. По-специално, $// section$ е вградена под $// article$.

Заявката от фиг.10.3 прави търсене за раздели за лятната ваканция, които са част от статиите от 2001 или 2002 година. В XPath двойните наклонени показват, че произволен брой елементи могат да се намират в пътя. Точката в клаузата с квадратни скоби се отнася за елемента, който променя клаузата. Клаузата $[././/yr = 2001 \text{ or } ././/yr = 2002]$ променя $// article$. Точката се отнася до $// article$ в този случай. По същия начин точката в $[about(., summer holidays)]$ се отнася до раздела, който променя клаузата.

Двете „`yr`“ условия са релационни ограничителни атрибути. Само статии, чиято година на атрибута е 2001 или 2002 (или които съдържат елемент с атрибут „`yr`“ 2001 или 2002) ще бъдат разглеждани. About клаузата е ограничителна: Разделите, които възникват при правилен тип на статиите, ще бъдат класирани според това как са уместни за темата лятната ваканция.

Ние просто премахваме от резултата всички елементи, които не отговарят на ограниченията на атрибутите. В тази глава, няма да се спирате как да направим това ефективно и ще се съсредоточим върху основната проблематика извличане на информация с XML извличане, а

именно как да се класират документи в зависимост от значимостта на критериите, написани в NEXI заявката.

Ако отхвърлим релационните атрибути, можем да представим документите като дървета само с един тип възел: възли елементи. Премахваме всички възли атрибути от XML документа, също и атрибутите *цикла* от фигура 10.1. Фигура 10.4 показва поддърво на документа от фиг. 10.1 само с възли елемент(маркирани с d1).

Ние можем да представим заявките като дърво по същия начин. Потребителите поставят заявки като създават обекти, което задоволява същото формално описание за документите. На фиг. 10.4 q1 е търсене за книги, чиито заглавия имат най-високи точки за думите Julius Caesar. q2 е търсене за Julius Caesar и заглавие Gallic war³.

10.2 Предизвикателства в XML извлечането

В този раздел, ние ще разгледаме редица предизвикателства, които правят структурираното извлечане на информация по-трудно, отколкото неструктурнираното извлечане.

Първото предизвикателство на структурираното извлечане е,че потребителите искат да се върнат части от документите (XML елементи), а не цели документи, както системите за извлечане на информация обикновено правят при неструктурно извлечане. Ако правим заявка „Шекспирови писеси за замъка на Макбет“, трябва ли да се върне сцената, действието или цялата писеса от фиг.10.2? В този случай, потребителят вероятно търси сцената. От друга страна, неуточнено търсене за *Макбет* трябва да върне писесата с това име.

Един критерий за избор на най-подходяща част от документ е принципа на структурираното документно извлечане:

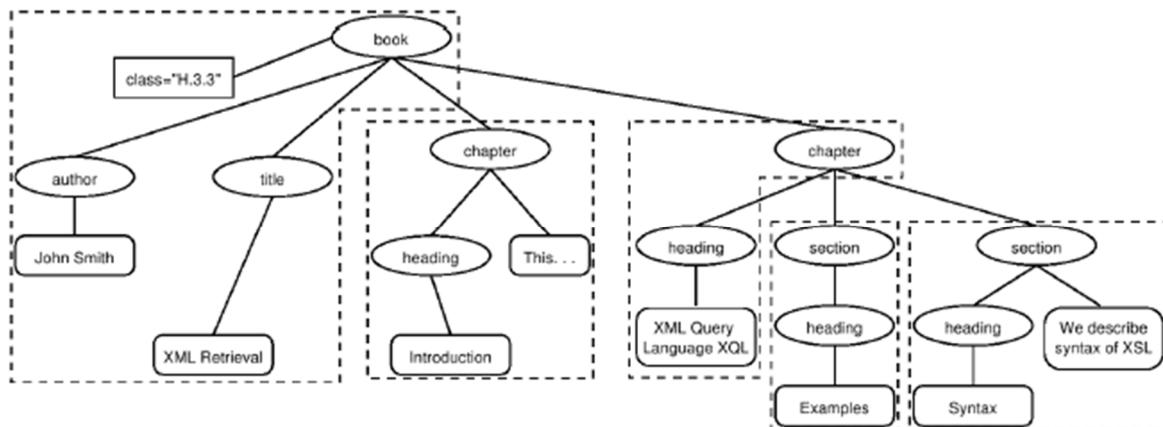
Принцип на структурираното документно извлечане. Системата винаги трябва да извлича най-вътрешната част от документ в отговор на заявка.

3. За да представим семантиката на NEXI заявки, трябва да изберем един възел от дървото за „целеви възел“, например, *раздел* в дървото на фиг. 10.3 . Без да изберем „целеви възел“, дървото на фиг. 10.3 няма да бъде търсене на раздели, вградени в статии (специфицирани от NEXI), но ще бъде търсене за статии, които съдържат раздели.

Този принцип мотивира стратегия, която връща най-малки части, които съдържат необходимата информация, и не излиза от това ниво. Въпреки това, може да бъде трудно да се приложи този принцип алгоритмично. Да разгледаме заявката title#"Macbeth" от фиг. 10.2 . Заглавието на трагедията, *Макбет*, и заглавието на Първо действие, Сцена VII, *Замъка на Макбет*, са две добри попадения, защото съдържат термина *Макбет*. Но в този случай, заглавието на трагедията, по-високия възел, е предпочитан. Да решим, кое ниво от дървото е правилното за отговор, е трудно.

Паралелно на въпроса, кои части от документа да се върнат на потребителя, е и въпроса, кои части от документа да се индексират. В раздел 2.1.2.(20 стр.), ние дискутирахме необходимостта от това да се определи какво е единица документ или *единица за индексиране*. При неструктурнираното извлечане на информация е ясно каква е правилната документна единица: файл на вашия работен плот, съобщение по пощата, уеб страница в мрежата и т.н. В

структурите на XML документа, има няколко различни подходи за определяне на индексираната единица.



Фигура 10.5 Разделяне на XML документ на неприпокриващи се единици за индексиране.

Един от подходите е да се групират възли в неприпокриващи се псевдо-документи както е показано на фигура 10.5. В примера, книгите, главите и разделите са проектирани да бъдат индексирани единици, но без да се припокриват. Например, най-лявата пункирана индексирана единица съдържа само онези части от дървото доминирани от *книга*, които не са части от друга единица за индексиране. Недостатъкът на този подход е, че псевдо-документите може да нямат смисъл за потребителя, защото те не са съгласувани единици. Например, лявата индексирана единица на фиг. 10.5 обединява три коренно различни елемента *клас*, *автор* и *заглавие*.

Можем също така да използваме един от най-големите елементи като индексирана единица, например, елемента *книга* в колекцията от книги или *пиеса* елемента за Шекспировите пиеси. Ние можем да намерим резултати при търсене на всяка книга или пиеса, която има успех. Например, заявката *замъка на Макбет* може да върне пиесата Макбет, след това можем да намериме Първо действие, сцена vii . За съжаление, това двустепенно търсене няма да успее да върне най-добрите поделементи за много заявки, защото значението на цялата книга не може да предскаже значението на малките части от книгата.

Вместо извлечане на големи единици и идентифициране на поделементи (отгоре надолу), ние също можем да намерим всички листа, да изберем най-подходящите от тях и след това да ги прибавим към по-големи дялове в последващата обработка (отдолу нагоре). За заявката „*замъка на Макбет*“ на фиг.10.1, ще извлечем Замъка на Макбет на първо минаване и след това да решим да върнем заглавието, сцената, действието или пиесата. Този подход има проблеми като: листото може да не бъде релевантно на заявката.

Най-малко ограничаващ подход е да се индексират всички елементи. Това също е проблемно. Много XML елементи не представляват смислени резултати при търсене, например, елементи като *definitely* или ISBN, който не може да се тълкува, без контекст. Също индексирането на всички елементи означава, че в резултатите от търсения ще има много излишна информация. За заявката *Замъка на Макбет* и документа на фиг.10.1, ще бъдат върнати всички елементи *пиеса*, *действие*, *сцена*, *заглавие* от пътя между възела корен и *Замъка на Макбет*. Възелът листо ще се появи четири пъти в резултатното множество, веднъж директно и три

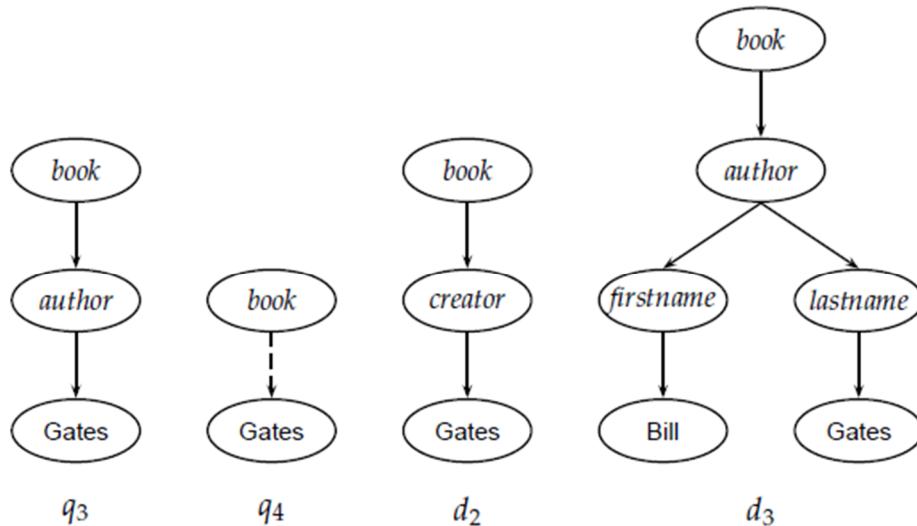
пъти като част от други елементи. Наричаме елементите, които се съдържат в рамките на всеки друг *вложен*. Да се върнат излишно вложени елементи не е разбираемо за потребителя.

Заради излишеството, причинено от вложените елементи, е обичайно да се ограничи множеството от елементи, които могат да се върнат. Ограничителната стратегия включва:

- Да се премахнат всички малки елементи
- Да се премахнат типове елементи, от които потребителите не се интересуват (това изисква работеща XML система за търсене, която пази тази информация)
- Да се премахнат всички типове елементи, за които оценителите предценяват, че не са от значение (ако оценките за значимост са налични)
- Да се задържат само тези типове елементи, за които дизайнът на системата или библиотеката предценяват, че са използвани резултати от търсене

В повечето от тези подходи, резултатното множество ще съдържа вложени елементи. Ние можем да поискаме да премахнем някои елементи в етап на последваща обработка за намаляване на излишството. Алтернативата е да се разпаднат някои вложени елементи в резултатния списък и да се използва *открояването* на термини от заявката, за да се привлече вниманието на потребителя на съответните пасажи. Ако думите в заявката са откроени, тогава сканирането на средния по големина елемент (например раздел) отнема малко повече време от сканирането на малък поделемент (например параграф). По този начин, ако секцията и параграфа се появят в списъка с резултатите, достатъчно е само да покажем раздела. Предимството на този подход е, че параграфът се представя заедно с неговия контекст (т.е. секцията). Тази връзка може да е от полза при тълкуването на параграфа (докладвано е от източника на информация) и параграфът да е от полза за заявката.

Ако потребителят знае схемата на колекцията и е в състояние да определи искания тип на елемента, тогава проблемът с излишството се свежда до вложени елементи от един и същи тип. Но както ние дискутирахме във въведението, потребителите често не знайт какво е името на елемента в колекцията (Ватикана страна ли е или град?) или те не знайт как да правят структурирани заявки.



Фигура 10.6 Схема хетерогенност: възпителни възли и несъответстващи имена.

Предизвикателствата в XML извличането свързани с влагане са, че ние може да се нуждаем да разграничаваме различни контексти на един термин, когато изчисляваме статистика за термина, за да определим ранг, и статистики за честота на обърнат документ (*idf*), дефинирано в секция 6.2.1 (стр.117). Например, терминът *Gates* под възела *автор* не е свързан с появя под възел, като *раздел*. В този пример няма смисъл да се изчислява честотата на срещане на термина в един документ.

Едно решение е да се изчисли *idf* за XML съдържание/термин двойка, например да се изчислят тежести за *author#/"Gates"* и *section#/"Gates"*. За съжаление, тази схема ще се сблъска с проблема разредени данни – това е когато много XML-контекстни двойки се срещат твърде рядко, за да се направи надеждна оценка *df* (раздел 13.2, страница 260, за обсъждане на разредени данни). Компромисът е само да се наблюдава родителския възел *x* на термина и да не се наблюдава останалия път от корена към *x*, за да се разграничат контекстите. Тази схема също има недостатък. Например, ние не различаваме имената на автори и имената на компании, ако имат родителски възел *име*. Но по-важни различия като примера преди малко *author#/"Gates"* и *section#/"Gates"*, ще бъдат уважени.

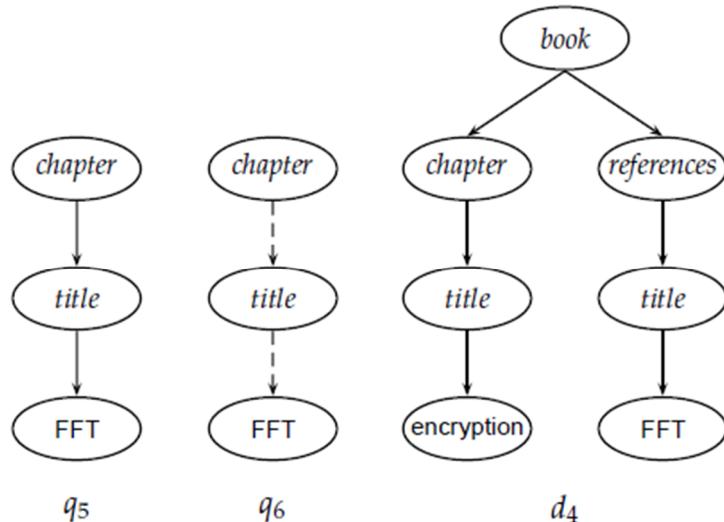
В много случаи, няколко различни XML схеми се случват в една колекция от XML документи и приложението за извлечане на информация често работи с повече от един източника. Този феномен е наречен хетерогенност на схемата (*schema heterogeneity*) или многообразие на схемата (*schema diversity*) и представлява още едно предизвикателство. Фигура 10.6 сравнява елементи, които имат различни имена: *creator* в *d2* и *author* в *d3*. В други случаи, структурната организация на схемите могат да бъдат различни: имената са преки наследници на възела *автор* в *q3*, но имат вложени възли *търво име* и *последно име* в *d3*. Ако ние използваме строго съвпадение по дървото, тогава *q3* няма да извлече *d2* и *d3*, въпреки че и двата документа са от значение. Някаква форма на приближено съвпадение на елемента име в комбинация с полу-автоматично съвпадение на различни структури на документа могат да помогнат тук. Човешко редактиране за съответствие на элемента в различни схеми ще се справи по-добре от автоматичните методи.

Хетерогенната схема е една от причините за несъответствията между заявка-документ като *q3/d2* и *q3/d3*. Друга причина е, че потребителите често не са запознати с имена на елементите и структурата на схемите на колекциите, които търсят. Това прави предизвикателство за дизайна на потребителския интерфейс на XML извличащата система. В идеалния случай, потребителския интерфейс трябва да изложи дървовидната структура на колекцията и да позволи на потребителите да уточнят елементите, които търсят. Ако вземем този подход, тогава дизайна на интерфейса за заявката в структурираното извлечени е много по-сложна от поленцето за търсене, в което пишем заявка от думи, при неструктуриното извлечане.

Можем също да поддържаме потребителят да интерпретира всички родител-наследник връзки в заявки с позволен брой възли. Наричаме тези заявки *разширенi заявки*. Дървото на фиг.10.3 и *q4* на фиг.10.6 са примери за разширени заявки. Ние показваме потомък взаимоотношенията с пунктирани стрелки. В *q4* пунктиряните стрелки свързват *book* със *Gates*. Като псевдо-XPath нотация приемаме *book//#"Gates"*: книга, която съдържа думата *Gates*, където пътят между книга и Гейтс може да бъде произволно дълъг. Псевдо-XPath нотацията за разширена заявка, която показва че Гейтс се среща в раздел книга е *book//section//#"Gates"*. Той е удобен за потребителите да могат да издават такива разширени заявки, без да се налага да се посочва

точната структурна конфигурация, в която терминът се случива – било защото те не знаят за точната конфигурация или защото те не знаят схемата на колекцията.

На фиг. 10.7, потребителят търси глава озаглавена FFT(q5). Предполагаме, че няма такава глава в колекцията, но че има сведения за книги за FFT (d4). Връщането на книга за FFT не е това, което потребителят е търсил, но е по-добре отколкото да върнем нищо. Разширениите заявки няма да помогнат тук. Разширената заявка qб също не връща нищо. Както ще дискутираме в Раздел 10.4, потребителят предпочита спокойно тълкуване на структурирани ограничения: Елементи, които не отговарят на структурираните ограничения перфектно трябва да се класират по-надолу, но те не бива да се пропускат от резултатите.



Фигура 10.7 Структурно несъответствие между две запитвания и документ.

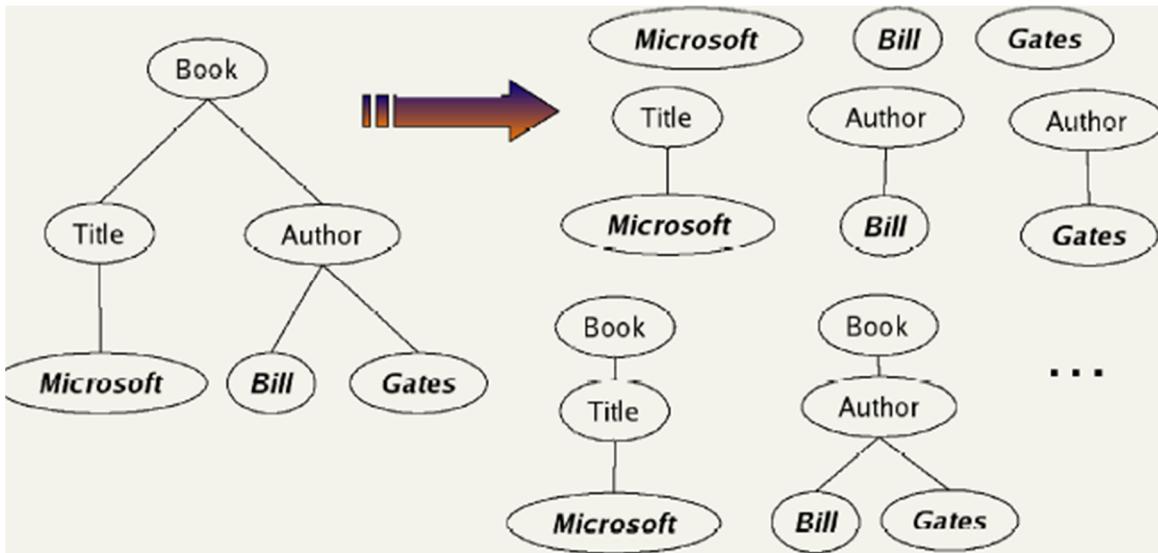
10.3 Векторен пространствен модел за XML извлечане на информация

В този раздел, ние представяме един прост векторно-пространствен модел за XML извлечане.

На фиг. 10.4 ние искаме книга, озаглавена Юлий Цезар, да отговаря на q1 и да не отговаря (или в по-малка степен) на q2. При неструктуриното извлечане, че има единствено измерение на векторното пространство за Цезар. В XML извлечанието, ние трябва да отделим заглавието Цезар от име на автор Цезар. Един начин е всяко измерение във векторното пространство да кодира дума с позицията и в XML дървото.

Фигура 10.8 илюстрира това представяне. Първо взимаме всеки текстов възел (който в напата настройка е винаги едно листо) и го разделяме на множество възли, по един за всяка дума. Така листо Бил Гейтс е разделено на две листа Бил и Гейтс. След това определяме измеренията да бъдат *лексикографски поддървета* (*lexicalized subtrees*) от документи – поддървета, които съдържат най-малко един термин. Подмножество на тези лексикографски поддървета е показано на фигурата, но има и други, например, поддърво отговарящо на целия документ без листото Гейтс. Сега можем да представим заявките и документите като вектори в това пространство от лексикографски поддървета и да изчислим съответствията между тях. Това означава, че можем да използваме векторно-пространствения формализъм от Глава 6 за XML извлечане.

Основната разлика е, че измеренията във векторното пространство при неструктурорираното извличане са термини, докато при XML извлечането са поддървета.



Фигура 10.8 Превръщане на XML документ (вляво) в множество от лексикографски поддървета (влясно).

Налице е компромис между размерността на пространството и точността на резултатите от заявката. Ако ние ограничим измеренията до термини, ще имаме стандартна система за извличане във векторно пространство, която ще извлича много документи, които не отговарят на структурата на заявката (например Гейтс в заглавието вместо автор елемента). Ако съзадем измерение за всяко поддърво в колекцията, размерността на пространството ще стане прекалено голяма. Компромис е да индексираме всички пътища, които завършват на един термин, с други думи, всяка XML контекст-термин двойка. Наричаме такава двойка *структурни терми* (*structural term*) и го означаваме с (c, t) : двойка от XML-контекст c и термин t . Документът на фиг. 10.8 има 9 структурирани термини. Седем са показани (например, "Bill" и $\text{Author} \# \text{Bill}$) и два не са показани: $/Book/\text{Author} \# \text{Bill}$ и $/Book/\text{Author} \# \text{Gates}$. Дървото с листа Бил и Гейтс е лексикографско поддърво, което няма структурирани термини. Използваме по-рано въведената псевдо-XPath нотация.

Ние ще тълкуваме всички заявки като разширени заявки-може да има произволен брой възпителни възли за всеки родител-наследник възел в заявката. Например, ще интерпретираме q_5 на фиг. 10.7 като q_6 .

Но ние предпочитаме документи, които отговарят на структурата на заявката, като вмъкват няколко допълнителни възела. Извличаните резултати са по-добри и това се разбира от теглото на всяко съвпадение. Проста мярка за близост между пътя на заявката c_q и пътя на документа c_d е следната функция за прилика на контекста CR:

$$CR(c_q, c_d) = \begin{cases} \frac{1+|c_q|}{1+|c_d|} & \text{if } c_q \text{ matches } c_d \\ 0 & \text{if } c_q \text{ does not match } c_d \end{cases} \quad (10.1)$$

където $|c_q|$ и $|c_d|$ са броя възли в пътя на заявката и пътя на документа, и c_q съвпада с c_d , ако можем да трансформираме c_q в c_d като вмъкнем допълнителни възли. Два примера от фиг. 10.6 $CR(c_{q4}, c_{d2}) = 3/4 = 0.75$ и $CR(c_{q4}, c_{d3}) = 3/5 = 0.6$, където c_{q4} , c_{d2} и c_{d3} са отговарящи

пътища от върха до листото през q4, d2 и d3, съответно. Стойността на $CR(c_q, c_d)$ е 1.0, ако q и d съвпадат.

Крайната оценка за даден документ се изчислява като вариант на косинусувата мярка (Уравнение (6.10), страница 121), която наричаме SimNoMerge:

$$(10.2) \quad \text{SIMNOMERGE}(q, d) = \sum_{c_k \in B} \sum_{c_l \in B} CR(c_k, c_l) \sum_{t \in V} \frac{\text{weight}(q, t, c_k) \text{weight}(d, t, c_l)}{\sqrt{\sum_{c \in B, t \in V} \text{weight}^2(d, t, c)}},$$

където V е речника на не-структуриранны термини; B е множеството от всички XML контексти; тежестите (q, t, c) и (d, t, c) са тегла на термина t в XML контекст по заявкa q и документ d , съответно. Изчисляваме теглата по един от методите от Глава 6, като $\text{idf}_t \cdot \text{wf}_{t,q}$. Обърнатата честота на срещане в документа idf_t зависи от кои елементи използваме за изчисляване на df_t (Раздел 10.2). Мярката за близост $\text{SIMNOMERGE}(q, d)$ не е истинска косинусова мярка и

стойностите и могат да бъдат по-големи от 1.0. Делим на $\sqrt{\sum_{c \in B, t \in V} \text{weight}^2(d, t, c)}$, за да нормализираме дължината на документа (Раздел 6.3.1, стр.121). Пропускаме да нормализираме заявката, за да опростим формулата. За дадена заявка, нормализирането е същото като за документ $\sqrt{\sum_{c \in B, t \in V} \text{weight}^2(q, t, c)}$

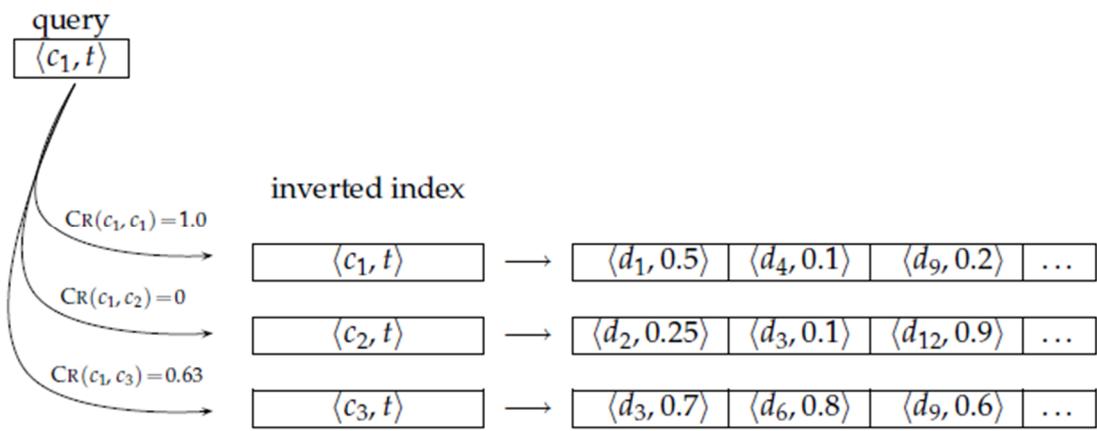
Алгоритъмът за изчисляване на SIMNOMERGE за всички документи в колекцията е показан на фиг. 10.9. Нормализираният масив на фиг. 10.9 съдържа формулата от уравнение (10.2) за всеки документ.

```

SCOREDOCUMENTSWITHSIMNOMERGE( $q, B, V, N, \text{normalizer}$ )
1 for  $n \leftarrow 1$  to  $N$ 
2 do  $score[n] \leftarrow 0$ 
3 for each  $\langle c_q, t \rangle \in q$ 
4 do  $w_q \leftarrow \text{WEIGHT}(q, t, c_q)$ 
5   for each  $c \in B$ 
6   do if  $CR(c_q, c) > 0$ 
7     then  $postings \leftarrow \text{GETPOSTINGS}(\langle c, t \rangle)$ 
8       for each  $posting \in postings$ 
9         do  $x \leftarrow CR(c_q, c) * w_q * \text{weight}(posting)$ 
10         $score[\text{docID}(posting)] += x$ 
11 for  $n \leftarrow 1$  to  $N$ 
12 do  $score[n] \leftarrow score[n] / \text{normalizer}[n]$ 
13 return  $score$ 

```

Фигура 10.9 Алгоритъм за даване на точки на документ SIMNOMERGE .



Фигура 10.10 Точки от заявка с един структурен термин в SIMNOMERGE.

Ние даваме пример за това как SIMNOMERGE изчислява заявка-документ прилики на фигура 10.10. (c_i, t) с един структуриран термин в заявката. Ние извличаме списък от публикации (c', t) с термин t . За първият резултат има $CR(c_1, c_1) = 1.0$, тъй като двата контекста са идентични. Следващият контекст няма прилика c_1 : $CR(c_1, c_2) = 0$ и се игнорира. Контекстът отговарящ на c_1 и c_3 е $0.63 > 0$ и ще бъде обработен. В този пример, документът с най-висок ранг е d_9 с близост от $1.0 \times 0.2 + 0.63 \times 0.6 = 0.578$. За опростяване теглото на заявката е прието да бъде 1.0.

Заявка-документ функцията от фиг. 10.9 е наречена SIMNOMERGE, заподо различните XML контекти са разделени за целите на претеглянето. Алтернативна мярка е SIMMERGE:

Събираме статистики за изчисляване на теглата от всички контексти, които имат прилика със c . Например, за изчисляване на документната честота на структуриран термин `atl#"recognition"`, ние също броим срещанията в контекста `fm/atl`, `article//atl` и т.н.

Променяме уравнението (10.2) чрез сливане на всички структурирани термини в документа, които имат прилика със структурирания термин от заявката. Например, контекстите `/play/act/scene/title` и `/play/title` в документа ще бъдат слити, когато има съвпадение на термина от заявката `/play/title#"Macbeth"`.

Контекстите имат не-нулева прилика в много случаи, когато CR от уравнение (10.1) връща 0.

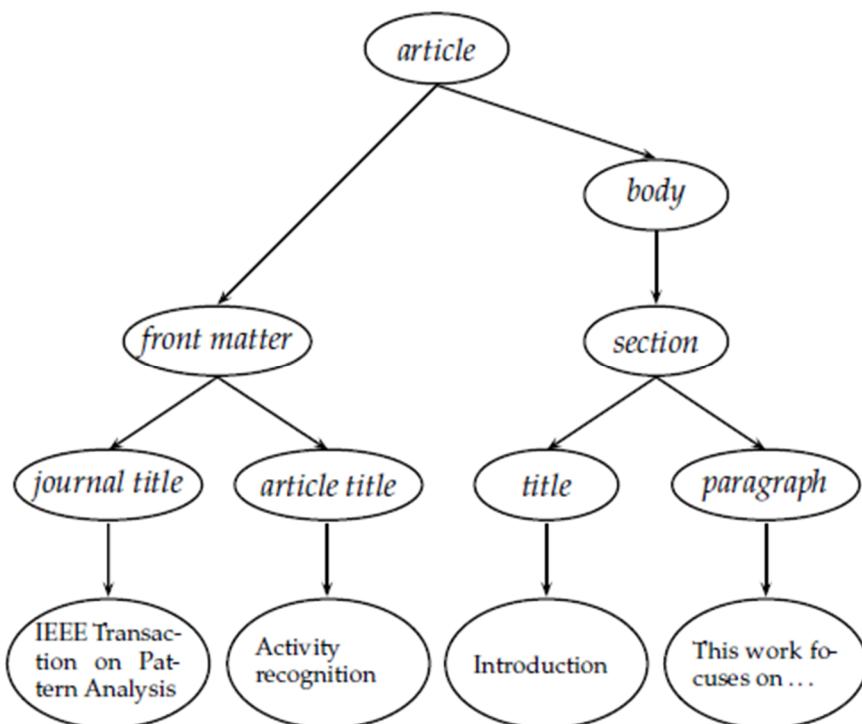
Облекчените условия за съвпадение на SIMMERGE повишава ефективността на XML извлечането.

10.4 Оценка на извлечането от XML

Основното място където може да намерите информация за XML извлечането е инициативата INEX (INitiative for the Evaluation of XML retrieval), която е произвела референтни колекции и комплекти от заявки.

12,107	number of documents
494 MB	size
1995–2002	time of publication of articles
1,532	average number of XML nodes per document
6.9	average depth of a node
30	number of CAS topics
30	number of CO topics

Фигура 10.2 INEX 2002 колекция статистики



Фигура 10.11 Опростена схема на документи в INEX колекция.

Колекцията INEX 2002 съдържа 12,000 статии от IEEE списания. В таблица 10.2 е предоставена колекция от статистики, а на фигура 10.11 е показана част от схемата на колекцията. Колекцията от списания IEEE е разширена през 2005. През 2006 INEX изполва Английската Уикипедия като тестова колекция. Значението на документите е оценено от хора, като е ползвана методологията описана в Раздел 8.1, като е променена за структурирани документи, което ще обсъдим на кратко.

Има два вида документи с която INEX може да работи СО “content-only” (само съдържание) или CAS “content and structure” (структурата и съдържание). При СО документите се правят заявки по ключови думи, както е при неструктурнираното извличане на информация. CAS документите имат структурни ограничения в допълнение към ключовите думи. Ние вече представихме такъв пример за CAS на фигура 10.3. Ключовите думи в този случай са лято и ваканция. Структурните ограничения са за броя на ключовите думи в раздел, който е част от статия имаша атрибут година със стойност 2001 или 2002. CAS има и двата критерия за структура и съдържание, което го прави по-сложен метод в сравнение с неструктурнираното извличане. INEX 2002 дефинира ортогонална релевантност между покритието на

компонентите и локалната релевантност. Измерението на компонентното покритие оценява възстановяването на структурата, колко правилно е да бъде по средата в дървото. Откриват се четири случая:

Пълно покритие (E). Търсената информация е основната тема на компонента и компонентът е значима единица информация.

Твърде малко (S). Търсената информация е основната тема на компонента, а компонентът не е значима единица информация.

Твърде голямо (L). Търсената информация е налична в компонента, но не е основната тема.

Липсващо покритие (N). Търсената информация не е тема на компонента.

Локалното измерване на знанието също има четири нива: от голямо значение, сравнително подходящо, маргинално значение и от никакво значение. За компонентите се съди по комбинацията от двете измерения, като тя се получава в код от букви и цифри. Сравнително подходящите компоненти са твърде малко в сравнение с тези от голямо значение, които имат точно покритие. На теория има 16 комбинации на покритие и значение, но много от тях не се срещат. На пример компоненти без значение не могат да имат точно покритие, така комбинация 3N е невъзможна.

Смислените комбинации са квантизиирани както следва:

$$Q(rel, cov) = \begin{cases} 1.00 & \text{if } (rel, cov) = 3E \\ 0.75 & \text{if } (rel, cov) \in \{2E, 3L\} \\ 0.50 & \text{if } (rel, cov) \in \{1E, 2L, 2S\} \\ 0.25 & \text{if } (rel, cov) \in \{1S, 1L\} \\ 0.00 & \text{if } (rel, cov) = 0N \end{cases}$$

Тази схема за оценка взема предвид факта, че двоичните смислени решения, които са стандартни в неструктуриното извлечане на информация не са подходящи за XML извлечане. 2S компонентата представя неизпълнена информация, която е трудна за разбиране без повече контекст, но отговаря частично на заявката. Квантилизиращата функция Q не работи с двоичен избор значимо/незначимо, което ни позволява да разбираме компонентата като частично значима.

algorithm	average precision
SIMNoMERGE	0.242
SIMMERGE	0.271

Фигура 10.3 INEX 2002 резултати от векторно пространствения модел от Раздел 10.3 за съдържание/структура (CAS) заявки и функцията $Q=$

Броят на значимите компоненти от извлеченото множество A от компоненти, може да бъде изчислен като:

$$\#(\text{relevant items retrieved}) = \sum_{c \in A} Q(rel(c), cov(c))$$

Като приближение, стандартните определения за точност взети от глава 8 и F могат да бъдат приложени към настоящото модифицирано определение на съответните елементи с някои неточности идващи от разликата на сумата със степен на уместност на двоичните оценки.

Един от недостатъците на този начин на измерване на значението е, че при покриването не се отчита. Ние обсъдихме идеята за пределното значение в контекста на неструктуррирано извлечане в раздел 8.5.1. Този проблем е сериозен в XML извлечането заради проблема с дълбокото влагане на елементи един в друг, който се получава като резултат от търсенето. Основният фокус на INEX е разработката на алгоритми и мерки за оценка, които да връщат пълни резултати оценени коректно.

Таблица 10.3 показва двата вектора на INEX 2002 от векторно-пространствената система обсъдена в раздел 10.3. По добра стратегия е да се ползва SimMerge, тъй като включва няколко структурни ограничения и най-вече разчита на съвпадение на ключовите думи. Медианата на средната точност на този метод е 0.147 (където медианата се влияе от средната прецизност на темите). Ефективността на XML извлечането е по-ниска от тази на неструктуррираното извлечане, тъй като XML извлечането е по-трудно. Вместо просто да намерим документ, ние трябва да намерим частта от документа отговоряща най-много на заявката. Да разгледаме система, която връща документ с фиксирано значение 0.6 и двоично значение 1 на върхът на върнатия списък. Тогава интерполираната точност е 0.00 на двоичната оценка, но може да бъде толкова ниска, колкото е фиксираното значение 0.6.

Таблица 10.3 ни дава типичната скорост на XML извлечането, но не прави разлика между структурирано и неструктуррирано извлечане. Таблица 10.4 показва директно ефекта от употребата на структурирано извлечане.

	content only	full structure	improvement
precision at 5	0.2000	0.3265	63.3%
precision at 10	0.1820	0.2531	39.1%
precision at 20	0.1700	0.1796	5.6%
precision at 30	0.1527	0.1531	0.3%

Фигура 10.4 Сравнение между търсене в съдържанието и пълно структурно търсене в INEX 2003/2004.

Резултатите са за езиково-базираната система в глава 12, която е оценена с CAS системите от INEX 2003 и 2004. Прецизността е като прецизността „ k “ дефинирана в глава 8. Дискретизиращата функция ползва за оценяване на картите голям брой подходящи елементи (закръгленето отговаря на 3Е елементите дефинирани за Q) като ги прави 1, а всички останали 0. Системите ползващи само съдържанието се третират като неструктурирани множества от думи. Структурираните елементи, които удоволстворяват структурните ограничения имат висок ранг спрямо тези, които не ги удоволстворяват. Например за заявка на фигура 10.3 елементите, които садържат фразата *summer holidays* в раздела си, ще получат висок ранг спрямо тези съдържащи в себе си “*abstract*”.

Таблицата показва как структурата помага да се увеличи прецизността към върхът на резултатния списък. Има голямо увеличение на прецизността при $k = 5$ и при $k = 10$. Почти няма подобреие при $k = 30$. Тези резултати показват ползата от ползването на структурирано извлечане. Структурираното извлечане налага допълнително ограничение структурен филтър и документите, които отговарят на филтъра е много вероятно да са търсените. Заради

филтърът някои търсени документи могат да отпаднат, но за прецизно ориентирани задачи структурното извличане дава по-добри резултати.

11 Вероятностно извлечане на информация

В по-предните теми забелязахме, че ако имаме няколко известни релевантни и нерелевантни документи, можем директно да дадем оценка за вероятността израз t да участва в релевантен документ $P(t|R=1)$, а също и, че това може да е основата на класификатор, който решава дали документите са релевантни или не. В този раздел ще направим по-систематично въведение в този подход в Извличането на информация (ИИ), който осигурява различна формална основа за модел на извлечане и дава като резултат различни техники за задаване тежести на термините.

Потребителите започват с *информационни нужди*, които превеждат до *представяне чрез фрази за търсене*. Също така има *документи*, които се превръщат в *представяне на документи* (последното се различава поне по това как текстът е разделен на части, но съдържа като цяло по-малко информация, например както когато е използван непозиционен индекс). Позовавайки се на тези две представяния, системата се опитва да определи колко добре документът съответства на информационните нужди. В булево- или векторно-пространствените модели за ИИ, свъпадението се прави чрез формално дефиниран, но семантично неточно пресмятане на индексираните термини. При дадена единствено фраза за търсене, система за ИИ има неопределено разбиране за информационните нужди. При дадени представяния на фразата и на документа, системата прави неопределена догадка дали съдържанието на документа е свързано с информационните нужди. Теорията на вероятностите осигурява принципна основа за разсъждение при неопределеност. Този раздел дава един отговор на това как да се използва тази основа, за да се оценява колко вероятно е даден документ да е свързан с информационните нужди.

Има повече от един възможен модел на извлечане, който има вероятностна основа. Тук ще изложим теорията на вероятностите и Принципа за вероятностна наредба (части 11.1-11.2), след това ще се съсредоточим върху Двоичния модел на независимост (част 11.3), който е оригиналния и все още най-значим вероятностен модел. Накрая ще представим свързани с тези, но разширени методи, които използват брой на термините, включително и емпирично успешната Okapi BM25 схема за определяне на теглата, както и модели с Бейсови мрежи (част 11.4). След това в раздел 12 ще представим алтернативен вероятностен езиков подход за моделиране при ИИ, който се развива със значителен успех през последните години.

11.1 Преговор на основната теория на вероятностите

С променлива A означаваме събитие (подмножество на пространството на изходите). Като еквивалентно представяне можем да използваме случайна променлива, която е функция от изходите към реални числа; подмножеството е областта, в която случайната променлива A има определена стойност. Често не знаем със сигурност дали някакво събитие е истина в света. Можем да изискаме вероятността на събитието да е $0 \leq P(A) \leq 1$. За две събития A и B събитието, при което се случват и A , и B , се описва със съвместната вероятност $P(A, B)$. Условната вероятност $P(A|B)$ изразява вероятността да се случи събитие A , при условие, че B се е случило. Основната връзка между съвместната и условната вероятност се дава със следното *верижно правило*:

$$(11.1) P(A, B) = P(A \cap B) = P(A | B)P(B) = P(B | A)P(A)$$

Без никакви допълнителни предположения, вероятността за съвместно събитие е равна на вероятността на едно от събитията, умножена по вероятността на другото събитие, при условие, че първото се е случило.

Ако означаваме с $P(\underline{A})$ допълнението на дадено събитие, по подобен начин намираме:

$$(11.2) P(\underline{A}, B) = P(A)P(\underline{A})$$

Теорията на вероятностите има и *правило за разделяне*, което казва, че ако събитие B може да се раздели на пълно множество от непресичащи се подсъбития, то вероятността за B е suma от вероятностите на подсъбитията. Частен случай на правилото ни дава:

$$(11.3) P(B) = P(A, B) + P(\underline{A}, B)$$

От горните можем да изведем правилото на Бейс за обръщане на условна вероятност:

$$(11.4) P(B) = \frac{P(A)P(A)}{P(B)} = \left[\frac{P(A)}{\sum_{X \in \{A, \underline{A}\}} P(X)P(X)} \right] P(A)$$

Това уравнение може да се разглежда и като начин за обновяване на вероятности. Започваме с начална оценка на вероятността на събитие A , когато нямаме никаква друга информация; това е *априорната вероятност* $P(A)$. Правилото на Бейс ни дава начин да изчислим *апостериорната вероятност* $P(A | B)$, след като сме отбелязали наличието на събитие B , базирано на вероятността B да се случи в двета случая – ако A е или не е изпълнено.

Накрая, често е удобно да говорим за *шанс* на събитие, който ни дава коефициент за това как вероятностите се изменят :

$$(11.5) O(A) = \frac{P(A)}{P(\underline{A})} = \frac{P(A)}{1 - P(A)}$$

11.2 Принцип за вероятностна наредба

11.2.1 Случаят с 1/0 загуба

Ще предполагаме постановка с наредено извличане, както в раздел 6.3, където имаме колекция от документи, потребителят поставя фраза за търсене и получава като резултат нареден списък с документи. Също така ще използваме двоично знание за релевантност, както в раздел 8. За фраза q и документ d от колекцията, нека $R_{d,q}$ е случаина променлива, която указва дали d е релевантен при дадена фраза q . Това означава, че приема стойност 1, ако документът е релевантен, и 0 в противен случай. Често ще пишем само R вместо $R_{d,q}$.

Използвайки вероятностен модел, очевидният ред, в който да представим документите на потребителя е подредени по оценката на тяхната вероятност за релевантност, като вземем предвид информационните нужди: $P(R = 1 | d, q)$. Това е основата на Принципа за вероятностна наредба (*Probability Ranking Principle* или PRP):

„Ако отговорът на система за извличане на всеки въпрос е наредба на документите от колекцията в намаляващ ред на вероятността за релевантност към потребител, подал въпроса, където вероятностите се оценяват възможно най-точно на база на каквито и данни да са

подадени на системата за тази цел, цялостната ефикасност на системата за този потребител ще бъде най-добрата постижима на база тези данни.”

В най-простиия случай на PRP няма цена за извлечане или други странични съображения, които да разграничават тежестта на действията или грешките. Губим точка за връщане на нерелевантен документ или пропуск да се върне релевантен такъв (такава двоична позиция, при която се оценява *точността*, се нарича *1/0 загуба*). Целта е да се върне възможно най-добрая резултат сред първите k документа, за всяка стойност на k , която потребителят решава да използва. Тогава PRP просто подрежда всички документи в намаляващ ред на $P(R = 1 | d, q)$. Ако множеството от извлечените резултати трябва да се върне вместо наредба, *Бейсовото оптимално правило за решение*, решението, което минимизира риска от загуба, е просто да се върнат документите, които е по-вероятно да бъдат релевантни, отколкото нерелевантни:

$$(11.6) \quad d \text{ е релевантен, ако } P(R = 1 | d, q) > P(R = 0 | d, q)$$

Теорема 11.1: PRP е оптимален, в смисъл, че минимизира очакваната загуба (известна още като Бейсов риск) при 1/0 загуба.

Доказателството на тази теорема изисква всички вероятности да се знаят с точност. На практика това никога не е така. Независимо от това обаче PRP дават много полезна основа за развиване на модели за ИИ.

11.2.2 PRP с цени за извлечане

Да предположим, че вместо това използваме модел с цени на извлечане. Нека C_1 е цената за пропускане на релевантен документ, а C_0 – цената за извлечане на нерелевантен документ. Тогава Принципът за вероятностна наредба казва, че ако за конкретен документ d и за всички d' , които още не са извлечени

$$(11.7) \quad C_0 \cdot P(R = 0 | d) - C_1 \cdot P(R = 1 | d) \leq C_0 \cdot P(R = 0 | d') - C_1 \cdot P(R = 1 | d')$$

то d е следващият документ, който ще бъде извлечен. Такъв модел ни дава форма структура, в която можем да моделираме разграничаващи цени на фалшиво положителните и фалшиво отрицателни грешки, дори и проблеми с поведението на системата, още на фазата на моделиране, вместо на тази на оценяване (както направихме в раздел 8.6).

11.3 Двоичен модел на независимост

Двоичният модел на независимост (*Binary Independence Model* или *BIM*), който ще представим в този раздел, е моделът, който традиционно се използва в PRP. Той използва някои прости предположения, които правят оценяването на вероятностната функция $P(R | d, q)$ практически ориентирано. Тук „двоичен“ означава: документите и търсенията са представени като двоични вектори на срещанията на термините. Тоест документ d се представя чрез вектор, където $x_t = 1$, ако терминът t се среща в документа d и $x_t = 0$, ако t не се среща. С това представяне много различни документи съответстват на един и същи вектор. Също така ако представяме q чрез вектор на срещанията \vec{q} (разликата между q и \vec{q} е по-маловажна, тий като често q е във формат на множество от думи). „Независимост“ означава, че термините са моделирани като независимо срещащи се в документите. Моделът не разпознава връзки между термините. Това предположение е далеч от точно, но често дава задоволителни резултати на практика; това е

„наивното“ предположение на Наивния Бейсов модел, разгледан по-подробно в раздел 13.4. Всъщност, BIM е точно същият като Бернули Наивен Бейсов модел на много променливи, представен в раздел 13.3. В известен смисъл това предположение е еквивалентно на предположението на векторния пространствен модел, където всеки термин е измерение, ортогонално на всички останали термини.

Първо ще представим модел, който предполага, че потребителят има нужда от информация на единствена стъпка. Както коментирахме в секция 9, виждайки множество от резултати, потребителят може да постави по-точно своята информационна нужда (да стесни кръга на търсене). Защастие, както споменахме там, тривиално е да се разшири BIM така, че да предоставя структура за обратна връзка на релевантността, която ще представим в модела в раздел 11.3.4.

За да направим стратегията за извличане на информация точна, нужно е да оценим как термините в документа допринасят за релевантността му, по-специално искаме да знаем как честотата на срещане на термини, честотата на документа, неговата дължина и други статистики, които можем да изчислим, влияят на решението за релевантност на документа и как те могат да се комбинират, за да се изчисли вероятността за релевантност. След това подреждаме документите в намаляващ ред на тази вероятност.

Тук предполагаме, че релевантността на всеки документ е независима от релевантността на останалите документи. Както отбелязахме в раздел 8.5.1, това не е точно: предположението е особено опасно на практика, ако допуска системата да връща еднакви или почти еднакви документи. При BIM моделираме вероятността $P(R|d, \vec{q})$ един документ да е релевантен чрез вероятност в смисъла на векторите на срещаните на термините $P(R|\vec{x}, \vec{q})$. Тогава, използвайки правилото на Бейс, имаме:

$$(11.8) P(\vec{x}, \vec{q}) = \frac{P(R=1, \vec{q})P(\vec{q})}{P(\vec{q})} P(\vec{x}, \vec{q}) = \frac{P(R=0, \vec{q})P(R=0|\vec{q})}{P(\vec{x}|\vec{q})}$$

В горните уравнения $P(R = 1, \vec{q})$ и $P(R = 0, \vec{q})$ са вероятностите че ако релевантен, съответно нерелевантен, документ е върнат, то представянето на този документ е \vec{x} . Трябва да мислим за тази величина като дефинирана с оглед на пространството от възможните документи в областта. Никога не могат да бъдат изчислени точните стойности на тези вероятности, затова трябва да се използват приближения: статистики за истинската колекция от документи се използват за оценяване на вероятностите. $P(\vec{q})$ и $P(\vec{q})$ означават априорната вероятност за получаване на релевантен или нерелевантен документ за търсене \vec{q} . Отново, ако знаем процента на релевантните документи в колекцията, то можем да използваме това число за оценка на $P(\vec{q})$ и $P(\vec{q})$. Тъй като документ е или релевантен, или нерелевантен за търсене, то трябва да е изпълнено:

$$(11.9) P(\vec{x}, \vec{q}) + P(\vec{x}, \vec{q}) = 1$$

11.3.1 Получаване на подреждаща функция за термините на търсенето

При дадена фраза за търсене q искаме да подредим върнатите документи по намаляващо $P(R = 1 | d, q)$. При BIM това се моделира с подреждане по $P(\vec{x}, \vec{q})$. Вместо да оценяваме вероятностите директно, понеже се интересуваме само от подредбата на документите, ще работим с други величини, които са по-лесни за изчисляване и които дават същата подредба.

По-конкретно ще подреждаме документите по техния шанс за релевантност (тъй като шансът за релевантност е монотонен с вероятността за релевантност). Това улеснява ситуацията, тъй като можем да изпуснем общия знаменател в 11.8 и получаваме:

$$(11.10) \quad O(\vec{x}, \vec{q}) = \frac{P(\vec{x}, \vec{q})}{P(\vec{x}, \vec{q})} = \frac{\frac{P(R=1, \vec{q}) P(\vec{q})}{P(\vec{q})}}{\frac{P(R=0, \vec{q}) P(R=0| \vec{q})}{P(\vec{x}| \vec{q})}} = \frac{P(\vec{q})}{P(\vec{q})} \cdot \frac{P(\vec{x}| R=1, \vec{q})}{P(\vec{x}| R=0, \vec{q})}$$

Левият множител в най-дясната част на уравнението е константа за всяка фраза за търсене. Тъй като само подреждаме документите, не е необходимо да я изчисляваме. Десният множител все пак се нуждае от изчисляване и това първоначално изглежда трудно: как може точно да се определи вероятността за съвпадение на целия вектор на срещанията? В този момент правим *Наивното Бейсово предположение за условна вероятност* – че присъствието или отсъствието на една дума в документа е независимо от това на всяка друга дума (при дадена фраза):

$$(11.11) \quad \frac{P(\vec{x}| R=1, \vec{q})}{P(\vec{x}| R=0, \vec{q})} = \prod_{t=1}^M \frac{P(x_t|R=1, \vec{q})}{P(x_t|R=0, \vec{q})}$$

Така:

$$(11.12) \quad O(\vec{x}, \vec{q}) = O(\vec{q}) \cdot \prod_{t=1}^M \frac{P(x_t|R=1, \vec{q})}{P(x_t|R=0, \vec{q})}$$

Тъй като всяко x_t е 0 или 1, може да разделим множителите и получаваме:

$$(11.13) \quad O(\vec{x}, \vec{q}) = O(\vec{q}) \cdot \prod_{t:x_t=1} \frac{P(x_t=1|R=1, \vec{q})}{P(x_t=1|R=0, \vec{q})} \cdot \prod_{t:x_t=0} \frac{P(x_t=0|R=1, \vec{q})}{P(x_t=0|R=0, \vec{q})}$$

Оттук нататък нека $p_t = P(x_t = 1|R = 1, \vec{q})$ е вероятността термин да се съдържа в документ, релевантен към фразата за търсене, а $u_t = P(x_t = 1|R = 0, \vec{q})$ – вероятността да се среща в ирелевантен документ. Тези величини могат да бъдат изобразени в следната таблица на вероятностите, в която елементите във всяка колона се сумират до 1 (11.14):

	документ	релевантен ($R = 1$)	ирелевантен ($R = 0$)
Терминът се среща	$x_t = 1$	p_t	u_t
Терминът липсва	$x_t = 0$	$1 - p_t$	$1 - u_t$

Да направим още едно опростявашо предположение, че за термини, които не се среща във фразата за търсене, е еднакво вероятно да се срещат в релевантни и ирелевантни документи, т.е. ако $q_t = 0$, то $p_t = u_t$ (това предположение може да се пропусне, ако правим обратна връзка на релевантността, както в раздел 11.3.4). Тогава за произведенията трябва да разглеждаме само термини, присъстващи във фразата, следователно:

$$(11.15) \quad O(\vec{x}, \vec{q}) = O(\vec{q}) \cdot \prod_{t:x_t=q_t=1} \frac{p_t}{u_t} \cdot \prod_{t:x_t=0, q_t=1} \frac{1-p_t}{1-u_t}$$

Лявото произведение е за термините от търсенето, които се срещат в документа, а дясното – за тези, които не се срещат.

Можем да преобразуваме израза като включим термините от търсенето, открити в документа, в дясното произведение, но същевременно разделяме дясното произведение с тях, за да не променяме стойността. Така получаваме:

$$(11.16) O(\vec{x}, \vec{q}) = O(\vec{q}) \cdot \prod_{t: x_t = q_t=1} \frac{p_t(1-u_t)}{u_t(1-p_t)} \cdot \prod_{t: q_t=1} \frac{1-p_t}{1-u_t}$$

Лявото произведение все още е над термини, срещащи се в документа, но дясното е над всички термини в търсенето. Това означава, че дясното е константа за всички възможни фрази за търсене, точно както и шансът $O(\vec{q})$. Тогава единствената величина, която трябва да изчислим, за да подредим документите по релевантност към търсенето, е лявото произведение. Можем да получим същата наредба и чрез логаритъм от тази величина, тъй като логаритъмът е монотонна функция. Така получаваме т. нар. *Ранг на извлечане* или *Retrieval Status Value* (RSV) за този модел:

$$(11.17) RSV_d = \log \prod_{t: x_t = q_t=1} \frac{p_t(1-u_t)}{u_t(1-p_t)} = \sum_{t: x_t = q_t=1} \log \frac{p_t(1-u_t)}{u_t(1-p_t)}$$

Така всичко се свежда до изчисляване на RSV. Дефинираме c_t :

$$(11.18) c_t = \log \frac{p_t(1-u_t)}{u_t(1-p_t)} = \log \frac{p_t}{1-p_t} + \log \frac{1-u_t}{u_t}$$

Коефициентите c_t означават логаритъм от пропорцията на шансовете на термините в тази фраза. Имаме шансовете на термин да присъства в документ, ако той е релевантен ($p_t/(1-p_t)$) и шансовете да присъства в документ, ако той е ирелевантен ($u_t/(1-u_t)$). Пропорцията на шансовете е отношението на два такива шанса, като накрая взимаме логаритъм от стойността. Полученото число е 0, ако термин има равни шансове да присъства в релевантни и ирелевантни документи. Коефициентите c_t служат като тегло на термините в модела и окончателно резултатът на документ относно дадена фраза за търсене е $RSV_d = \sum_{x_t=q_t=1} c_t$. При изчисление ще ги акумулираме за термините за търсене, срещащи се в документите, както правехме при векторно-пространствения модел в раздел 7.1. Сега остава да видим как ще изчисляваме стойностите на c_t за конкретна колекция и фраза за търсене.

11.3.2 Изчисляване на вероятности

За всеки термин t как ще изглеждат всички тези числа c_t за цялата колекция? Таблицата по-долу дава брой на документите в колекцията, където df_t е броят на документите, съдържащи t (11.19):

	документ	релевантен	ирелевантен	Общо
Терминът се среща	$x_t = 1$	s	$df_t - s$	df_t
Терминът липсва	$x_t = 0$	$S - s$	$(N - df_t) - (S - s)$	$N - df_t$
	Общо	S	$N - s$	N

Използвайки това, $p_t = s/S$ и $u_t = (N - df_t) - (S - s)/N$, също и

$$(11.20) c_t = K(N, df_t, S, s) = \log \frac{s/(S-s)}{(df_t-s)/((N-df_t)-(S-s))}$$

За да изключим възможността за получаване на нули (като например ако всеки или пък никой релевантен документ не съдържа този термин), стандартният подход е да добавим $\frac{1}{2}$ към всеки от 4-те множители в горното уравнение, след което да изменим крайните бройки (общите) със съответно (така най-долната дясна клетка ще съдържа $N + 2$). Тогава имаме:

$$(11.21) \hat{c}_t = K(N, df_t, S, s) = \log \frac{(s+\frac{1}{2})/(S-s+\frac{1}{2})}{(df_t-s+\frac{1}{2})/(N-df_t-S+s+\frac{1}{2})}$$

Добавянето на $\frac{1}{2}$ по този начин е приста форма на заглаждане. За опити с безусловни изходи (като отбелязване на присъствието или отсъствието на термин), един начин да изчислим вероятността на събитие според данните е като просто преоброям броя на срещанията на това събитие, разделен на броя на опитите. Това наричама *относителна честота* на събитие. Изчисляването на вероятност с относителна честота е *изчисляване на максималната вероятност* (*maximum likelihood estimate* или *MLE*), защото тази стойност прави наблюдаваните данни най-вероятни. Ако обаче използваме само *MLE*, то вероятностите на събитията, които виждаме, обикновено са твърде високи, докато други събития, които никога не са се случили. Когато поставяме на тях относителна честота 0, това е твърде ниска оценка, а също и счупва модела, тъй като всичко, умножено по 0, е отново 0. Едновременното намаляване на изчислените вероятности на срещаните събития и увеличаване на вероятностите за тези, които не се срещат, наричаме *заглаждане* (*smoothing*). Прост начин за това е да добавяме число a към всяка от бройките на срещанията. Тези *псевдобройки* съответстват на използването на равномерно разпределение над речника като Бейсова априорна вероятност, според уравнение 11.4. Първоначално предполагаме нормално разпределение на събитията, като големината на a бележи силата, с която предполагаме това разпределение, след което обновяваме стойностите на вероятностите на база на наблюдението. Тъй като вярата ни в нормалното разпределение е слаба, използваме $a = \frac{1}{2}$. Това е форма на *максимално апостериорно* оценяване, в което избираме най-вероятната стойност на вероятностите на база на априорните и наблюдаваните събития, следвайки уравнение 11.4

11.3.3 Вероятностни оценки в практиката

Приемайки, че релевантните документи са много малка част от колекцията, приемливо е да приближаваме статистики за ирелевантни документи чрез статистики за цялата колекция. Можем да приемем, че u_t (вероятността от появя на термина в ирелевантни документи за заявка) е df_i/N и

$$(11.22) \log \left[\frac{1-u_t}{u_t} \right] = \log \left[\frac{N-df_t}{df_t} \right] \approx \log \frac{N}{df_t}.$$

С други думи, можем да осигурим теоретична обосновка за най-често използваната форма на idf претегляне, която видяхме в Секция 6.2.1.

Приближаването в уравнение (11.22) не може лесно да бъде разширено до релевантните документи. Количество \hat{c}_t може да се оцени по различни начини:

1. Можем да използваме честотата на появяване на термина в известните и релевантни документи (ако има такива). Това е базисът на вероятностния подход за

постигане на релевантен резултат за теглото при обратна връзка, който е разгледан в следващата подсекция.

2. Croft and Harper (1979) предложили използването на константа в техния комбиниран модел на попаденията. Например, можем да предположим, че p_t е константа за всички термини x_t във фразата за търсене и нека $p_t = 0.5$. Това означава, че всеки термин има шансове да се съдържа в релевантен документ и така p_t и $(1 - p_t)$ факторите се компенсират в израза за RSV. Такава преценка е слаба, но не противоречи много на търсенето на термини, появяващи се в много, но не всички релевантни документи. Комбинирайки този метод с нашето предишно приближение за u_t , класирането на документа се определя просто чрез това, кои фрази се появяват в документите, подредени по тяхното idf претегляне. За кратки документи (заглавия или резюмета), в случаите, когато итеративното търсене е нежелателно, използването на това претегляне може да бъде задоволително. При много други обстоятелства обаче бихме искали да се справим по-добре.
3. Greiff (1998) твърди, че константната оценка на p_t в модела на Croft и Harper (1979) е теоретично проблематична и не е наблюдавана емпирично: както може да се очаква, p_t нараства с df_t . Опирајки се на своя анализ на данните, възможно предложение би било да се използва приближението $p_t = 1/3 + 2/3 dft/N$.

11.3.4 Вероятностни подходи за релевантността на обратната връзка

Можем да използваме (псевдо-)релевантна обратна връзка в итеративния процес на оценяване, за да получим по-точна оценка на p_t . Вероятностният подход за получане на релевантна обратна връзка се прилага по следния начин:

1. Даваме начална оценка на p_t и u_t . Това може да се направи като използваме оценките от предишната секция.
2. Използваме текущата оценка на p_t и u_t за да определим най-доброто предположение в набора от релевантни документи $R = \{d : Rd, q = 1\}$. Използваме този модел за да извлечем списък от кандидат-релевантни документи, които представяме на потребителя.
3. Комуникираме с потребителя, за да определим по-прецизно модела за R . Осьществяваме това, като научаваме от потребителя релевантната оценка за някои множества от документи V . Базирано на релевантната оценка, V са разделени на две подмножества $VR = \{d \in V, Rd, q = 1\} \subset R$ и $VNR = \{d \in V, Rd, q = 0\}$, което е разпадане на R .
4. Оценяваме отново p_t и u_t на базис на известните ни релевантни и ирелевантни документи. Ако множествата VR и VNR са достатъчно големи може да сме способни да оценим тези количества директно от документите като оценка на максималното правдоподобие:

$$(11.23) p_t = |VR_t| / |VR|$$

(където VR_t е множеството от документите в VR , които съдържат x_t). В практиката обикновено трябва да изгладим тези оценки. Това можем да постигнем като прибавим $1/2$ към $|VR_t|$ и към броя на релевантните документи, несъдържащи термина:

$$(11.24) p_t =$$

И така, множеството от документи, избрани от потребителя (V) обикновено е много малко и затова не може да се разчита на резултатната статистическа оценка (шумна е), дори и оценките да са изгладени. Често е по-добре да комбинираме новата информация с оригиналното предположение при Бейсовото актуализиране. В този случай имаме:

$$(11.25) p_t^{(k+1)} =$$

Тук $p_t^{(k)}$ е к-тата оценка за p_t в актуализирането на итеративния процес и е използвано Бейсова априорна вероятност следващата итерация с теглото на k . Свързването на това уравнение с уравнение (11.4) изисква малко повече теория на вероятностите, отколкото представихме тук. Но формата на резултатното уравнение е доста проста: вместо да разпределяме разномерно псевдо-броя, ние ще разпространим общо к псевдо-броя според предишната оценка, което представлява главното разпределение. При отсъствие на други доказателства (и предположението, че потребителя може би посочва грубо 5 релевантни или ирелевантни документи) стойността за $k = 5$ може да бъде приемана. Това означава, че най-силно претеглената оценка не се променя прекалено много от доказателства, при условие че се отнасят за малък брой документи.

Повтаря се процеса от стъпка 2, генерирайки поредица от приближения за R и съответно за p_t , докато потребителя не бъде доволен.

Лесно е и да се изведе версия за псевдо-релевантната обратна връзка от този алгоритъм, като положим $VR = V$. По -точно :

1. Избираме първоначални оценки за p_t и u_t , както по-горе.
2. Определяме предположение за размера на множеството на релевантните документи. Ако не сме сигурни, консервативното (прекалено малкото) предположение е най-добрания избор. Това определя използването на множество V с точно определена размерност от най-високо класирани документи.
3. Подобряваме нашите предположения за p_t и u_t . Избираме от методите на уравнения (11.23) и (11.25) за нова оценка на p_t , освен това сега ги базираме на множеството V вместо на VR . Ако към подмножеството V_t от документи V , съдържащи x_t , добавим $\frac{1}{2}$ изглаждане, получаваме:

$$(11.26) p_t =$$

И ако предположим, че документите, които не са извлечени не са релевантни, тогава можем да актуализираме ит оценка:

$$(11.27) u_t =$$

Повтаряне на стъпка 2 докато класирането на върнатите резултати не станат сходящи

Ведър получим ли истинска оценка за p_t , после стойността на c_t теглото, използвано в RSV, изглежда почти като tf-idf стойността. Например, използвайки уравнения (11.18), (11.22) и (11.26) имаме:

$$(11.28) c_t = \log\left[\frac{|V_t| + \frac{1}{2}}{|V| - |V_t| + 1} \cdot \frac{N}{df_t}\right]$$

Но нещата не са съвсем същите: $p_t / (1 - p_t)$ измерва частта (оценената) на релевантните документи, в които терминът t участва, не неговата честота:

$$(11.29) c_t = \log + \log \frac{N}{df_t}$$

Вижда се, че събираме двата логаритмично масцирани компоненти, не ги умножаваме.

11.4 Оценяване и някои разширения

11.4.1 Оценяване на вероятностен модел

Вероятностните методи са едни от най-старите официални модели в ИИ. Чак през 1970г. те са счетени като възможност за поставяне на едно по твърдо начало на ИИ. Традиционно вероятностното ИИ има чисти идеи, но методите му никога не са спечелвали като представяне. Получаване на разумни приближения на необходимите вероятности за вероятностен ИИ модел е възможно, но изиска някои главни съображения. В BIM те са:

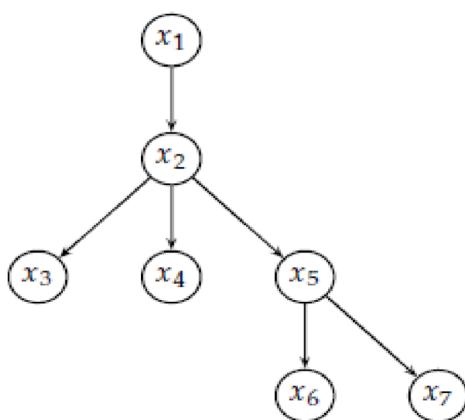
- Булево представяне на документи/фрази/съответствия
- Независимост на термините
- Термините извън фразата не влияят на резултата
- Стойностите на съответствията в документите са независими

Може би съображенията за моделирането правят трудно постигането на добро представяне. Главният проблем изглежда е, че във вероятностния модел или изиска частична състоятелна информация, или само да се даде възможност за извлечение на очевидно ниски модели на претегляне на термините.

Нещата започват да се променят през 1990г., когато BM25 схемата за теглата, има много добро представяне и започва да се приема като претегляща схема на термини от много групи. Разликата между „векторното пространство“ и „вероятностните“ ИИ системи не е толкова голяма: във всеки случай построяваме схема за ИИ. За вероятностните ИИ системи просто накрая добавяме запитвания по малко по-различна формула, мотивирана от теорията на вероятностите.

11.4.2 Дърворидна структура на зависимост между термините

Някои от съображенията на BIM могат да бъдат премахнати.



Фигура 11.1 Дървовидна зависимост между термините

11.4.3 Okapi BM25: недвоичен модел

BM отначало бил проектиран за записи в къс каталог и резюмета, работел добре в този контекст, но за модерното търсене в цял текст изглеждало, че моделът трябва да обърне внимание на честотата на срещане на термините и дължината на документа. BM25 схемата на теглата, често наричана *Okapi weighting*, след системата, в която първо е била създадена, била развита до начин за построяване на вероятностен модел, чувствителен към тези количества, които не представят прекалено много допълнителни параметри. Няма да развиваме пълната теория, която стои зад модела, а само ще представим серия от форми, които построяват сега използваната стандартна форма за оценяванена документи. Най-лесният резултат за документ d е просто idf претегляне представено от фразите за търсене, както в уравнение (11.22) :

$$(11.30) RSV_d = \sum_{t \in q} \log \frac{N}{df_t}$$

Понякога алтернативна версия на idf е използвана. Ако започнем с формулата в уравнение (11.21), но при липса на информация за състоятелна обратна връзка смятаме, че $S = s = 0$:

$$(11.31) RSV_d = \sum_{t \in q} \log \frac{N - df_t + \frac{1}{2}}{df_t + \frac{1}{2}}$$

Можем да подобрим уравнение (11.30) като разложим на множители почестотата на срещане на всеки от термините и дължината на документа:

$$(11.32) RSV_d = \sum_{t \in q} \log \left[\frac{N}{df_t} \right] \cdot \frac{(k_1 + 1)tf_{td}}{k_1((1-b) + b \times (\frac{L_d}{L_{ave}})) + tf_{td}}$$

Тук tf_{td} е честотата на срещане на термина t в документа d , L_d и L_{ave} са дължината на документа d и средната дължина на документи в цялата колекция. Променливата k_1 е положителен параметър който калибрира честотата на машабиране на термините.

$k_1 = 0$ съответства на двоичния модел (без честота на повтаряне на термините), а висока стойност за k_1 съответства на използването честотата на повтаряне на термините. b е друг позитивен параметър ($0 \leq b \leq 1$), който определя машабирането чрез дължината на документа: $b = 1$ съответства на пълно машабиране на теглото на термина чрез дължината на документа, докато $b = 0$ означава, че няма нормализиране чрез дължината.

Ако фразата е дълга тогава може също да използваме подобни тегла за фразите на търсене. Това е добре ако фразите са цели параграфи.

$$(11.33) RSV_d = \sum_{t \in q} \log \left[\frac{N}{df_t} \right] \cdot \frac{(k_1 + 1)tf_{td}}{k_1((1-b) + b \times (\frac{L_d}{L_{ave}})) + tf_{td}} \cdot \frac{(k_3 + 1)tf_{tq}}{k_3 + tf_{tq}}$$

Като tf_{tq} е честота на повтаряне на термина t във фразата q , k_3 е друг позитивен параметър който този път калибрира честотата на машабирането на термините на фраза. В представеното уравнение няма нормализиране на дължината фразите. То не е необходимо, защото извличането е било направено с фиксиране само на една фраза.

Ако имаме релевантни налични решения, тогава можем да използваме пълната форма на уравнение (11.21) на мястото на приближенето $\log(N/df)$ представено в (11.22):

$$(11.34) RSV_d = \sum_{t \in q} \log \left[\frac{\frac{|VR_t| + \frac{1}{2}}{|VN_R_t| + \frac{1}{2}}}{\frac{df_t - |VR_t| + \frac{1}{2}}{N - df_t - |VR| + |VR_t| + \frac{1}{2}}} \right] \times \frac{(k_1 + 1)tf_{td}}{k_1((1 - b) + b(\frac{L_d}{L_{ave}})) + tf_{td}} \times \frac{(k_3 + 1)tf_{tq}}{k_3 + tf_{tq}}$$

Тук VR_t , NVR_t и VR са използвани както в секция 11.3.4. Първата част на израза отразява значението на обратната връзка (или просто idf претеглянето ако няма никаква релевантна налична информация), втората част представя честотата на срещане на термина в документа и дължината на мащабиране на документа и третата отчита честотата на срещане на термина във фразата за търсене.

Формулата BM25 за претегляне на термините е била изполвана успешно върху множество колекции и задачи за търсене.

11.4.4 Бейсови подходи на мрежата за ИИ

Turtle и Croft (1989;1991) представили необходимостта от Бейсовите мрежи (Jensen and Jensen 2001) за ИИ като форма на графичен вероятностен модел. Няма да описваме Бейсовите мрежи в детайли, защото това ще отнеме прекалено много място. Като цяло тези мрежи използват насочени графи за да покажат вероятносни зависимости между променливите, както на фигура 11.1, и са довели до развитието на "умни" алгоритми за влияние, които да дадат възможност за научаване и правене на изводи с произволни знания в рамките на произволно насочени ациклични графи. Turtle и Croft използват изтънчени мрежи за да моделират по-добре комплексните зависимости между документ и нужната информация на потребителя.

Моделът се разделя на две части: мрежа за събиране на документи и мрежа за фрази. Първата е голяма, но може да бъде предварително пресметната: свързва документи, термове, концепции. Концепциите са като синонимен речник-разширяване на условията, посочени в документа. Мрежата за фрази за търсене е сравнително малка, но винаги когато постъпва нова фраза трябва да се прави и нова мрежа. Мрежата за фрази свързва фразите за търсене и подизразите с нужната информация на потребителя.

Резултатът е гъвкава вероятностна мрежа, която може да генерира различни по-лесни булеви и вероятностни модели. Наистина това е първият случай на статистически класиран модел за извлечане, който естествено поддържа структурирани фрази като оператори. Системата допуска ефективно високо-мащабно извлечане. Тя е била в основата на системата за извлечане на текст InQuery, създадена в университета в Масачузетц. Системата е представена на високо ниво на TREC evaluation и е продадена с търговска цел.

12 Езикови модели за извлечение на информация

Опитвайки се да извлечем информация от документи, добра идея е да конструираме заявки към тях, състоящи се от думи, които по дадени критерии са от значение за документа. Езиковите модели за извлечение на информация изпълняват тази идея - един документ отговаря на дадена заявка, ако е достатъчно вероятно моделът на документа да генерира заявката, т.е. ако документът съдържа думите от заявката достатъчно често.

За целта е нужно документите да бъдат подредени по следния начин – от всеки документ d се построява вероятностен езиков модел M_d , на базата на който документите се сортират по вероятността моделът да генерира заявката q ($P(q|M_d)$).

Крайни автомати и езикови модели

Един езиков модел на документ генерира заявки, по начин подобен на този, по който крайният автомат генерира език.

Да разгледаме крайния автомат:



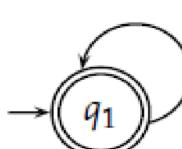
Стрелката показва началното състояние, а двойния кръг възможното крайно състояние. Автомата може да разпознава или генерира езика, съдържащ

I wish, I wish I wish, I wish I wish I wish,....

Когато всеки възел има вероятностно разпределение на възможните генерирации на термове, говорим за езиков модел. Тоест, езиковият модел е функция, която дава вероятностна стойност на терм. За езиковия модел M върху дадена азбука Σ имаме формулата:

$$\sum_{s \in \Sigma^*} P(s) = 1$$

Да разгледаме един прост пример на езиков модел:



the	0.2
a	0.1
frog	0.01
toad	0.01
said	0.03
likes	0.02
that	0.04
...	...

$$P(\text{STOP}|q_1) = 0.2$$

Той се състои само от един възел, с вероятности на генериране на една дума, дадени отстрани. След генериране на някоя дума, той може да продължи да генерира следваща или да завърши.

Можем да пресметнем вероятността на всяка последователност от думи. Например:

$$\begin{aligned} P(\text{frog said that toad likes frog}) &= (0.01 \times 0.03 \times 0.04 \times 0.01 \times 0.02 \times 0.01) \\ &\quad \times (0.8 \times 0.8 \times 0.8 \times 0.8 \times 0.8 \times 0.8 \times 0.2) \\ &\approx 0.000000000001573 \end{aligned}$$

На първия ред са вероятностните стойности на всяка следваща дума, а на втория вероятностите след нейното генериране автомата да продължи (0.8) или да спре (0.2).

Да разгледаме два различни езикови модела M_1 и M_2 :

Model M_1	Model M_2
the 0.2	the 0.15
a 0.1	a 0.12
frog 0.01	frog 0.0002
toad 0.01	toad 0.0001
said 0.03	said 0.03
likes 0.02	likes 0.04
that 0.04	that 0.04
dog 0.005	dog 0.01
cat 0.003	cat 0.015
monkey 0.001	monkey 0.002
...	...

Всеки от тях дава различна вероятностна стойност за всяка от думите. Можем да пресметнем вероятностите $P(s|M_1)$ и $P(s|M_2)$, където s е същата последователност от думи – “frog said that toad likes frog”

s	frog	said	that	toad	likes	that	dog
M_1	0.01	0.03	0.04	0.01	0.02	0.04	0.005
M_2	0.0002	0.03	0.04	0.0001	0.04	0.04	0.01

$$P(s|M_1) = 0.00000000000048$$

$$P(s|M_2) = 0.0000000000000384$$

12.1 Видове езикови модели

При определяне на вероятностите на поредица от термове използвахме *унифрамен езиков модел* (*unigram language model*). Той се характеризира с това, че всеки терм се определя независимо:

$$P_{\text{uni}}(t_1 t_2 t_3 t_4) = P(t_1) P(t_2) P(t_3) P(t_4)$$

Има и по-сложни модели, например *биграмния* (*bigram language model*), при който всеки следващ терм зависи от предишния:

$$P_{\text{bi}}(t_1 t_2 t_3 t_4) = P(t_1) P(t_2 | t_1) P(t_3 | t_2) P(t_4 | t_3)$$

По-сложни модели се използват за реализиране на гласово разпознаване (speech recognition), граматически проверки (spelling correction), машинен превод (machine translation) и други. В извличането на информация най-често използвания вид езикови модели е униграмният.

12.2 Полиномно разпределение

При униграмния модел, подредбата на думите не е от значение. Затова такива модели често са наричани модели „bag of words“. Понеже вероятността на дадена последователност от думи е еднаква при разместване на думите можем да говорим за полиномно разпределение (обобщен случай на биномно разпределение).

Вероятностната функция на полиномно разпределение има следната стойност:

$$\frac{n!}{x_1! \cdots x_k!} p_1^{x_1} \cdots p_k^{x_k}$$

Имаме n опита, като всеки опит завършва с един от k възможни изхода с вероятности p_1, p_2, \dots, p_k .

Нека $L_d = \sum_{1 \leq i \leq M} \text{tf}_{t_i, d}$ е дължината на документ d , M е размерът на „речника“. Тогава получаваме формулата:

$$P(d) = \frac{L_d!}{\text{tf}_{t_1, d}! \text{tf}_{t_2, d}! \cdots \text{tf}_{t_M, d}!} P(t_1)^{\text{tf}_{t_1, d}} P(t_2)^{\text{tf}_{t_2, d}} \cdots P(t_M)^{\text{tf}_{t_M, d}}$$

Основния проблем при избор на езиков модел е това, че не можем точно да определим това, от което се нуждаем. Обикновено разполагаме с определена извадка, която представлява частен случай на модела. Например при гласовото разпознаване (speech recognition) имаме даден кратък текст, но трябва да очакваме, че потребителя ще използва и други думи в други изречения, които ще бъдат непознати за модела.

12.3 Използване на модела за вероятност на заявките

Основният метод за използване на езиковите модели в извличането на информация е моделът за вероятност на заявките. От всеки документ d образуваме езиков модел M_d . Целта е да подредим документите по вероятността им да са възможно най-близки със заявката q , т.e. $P(d|q)$. От формулата на Бейс:

$$P(d|q) = P(q|d)P(d)/P(q)$$

$P(q)$ е еднаква за всички документи. Същото можем да кажем и за $P(d)$ и да ги игнорираме във формулата. Тогава резултатът ще бъде $P(q|d)$ – вероятността на заявката q , при условие езиковия модел на документа d .

Най-използвания метод за подреждане на документите е полиномния униграмен езиков метод. Имаме:

$$P(q|M_d) = K_q \prod_{t \in V} P(t|M_d)^{\text{tf}_{t,d}}$$

Тук $K_q = L_d! / (\text{tf}_{t_1,d}! \text{tf}_{t_2,d}! \cdots \text{tf}_{t_M,d}!)$, което е константа за дадена заявка и затова ще бъде игнориран.

За извлечане чрез езиков модел, генерирането на заявките се разглежда като случаен процес. Използва се следния подход:

- Определя се езиков модел за всеки документ
- Пресмята се $P(q|M_d)$ – вероятността за генериране на заявката според всеки от тези модели
- Документите се подреждат според стойностите на тези вероятности

При този основен модел се счита, че потребителя има някаква идея за съдържанието на документа, който търси и неговата заявка ще е базирана върху думите, съдържащи се в него. По този начин по-лесно се отличават най-вероятните документи от цяла група, между които е и търсеният от потребителя.

Изчисляване на вероятността за генериране на заявка

Вероятността да генерираме заявка при даден езиков модел (M_d) M_d на документ d , изчислявайки чрез максималната вероятност и предполагайки униграмен модел е:

$$\hat{P}(q|M_d) = \prod_{t \in q} \hat{P}_{\text{mle}}(t|M_d) = \prod_{t \in q} \frac{\text{tf}_{t,d}}{L_d}$$

където M_d е езиковият модел на документа d ,

$\text{tf}_{t,d}$ – честота на срещане на терма t в документа d (term frequency),

и L_d – броя на думите в документа d (tokens).

Т.е. просто броим колко често се среща дадена дума в документа и делим на общия брой думи в документа d .

Класическият проблем при използването на езиков модел е изчисляването на приближенето на вероятността, когато термовете се срещат много рядко в документите. В частност, някои думи от заявката може да не се срещат в документа, но да имат информационна стойност. Ако се опитаме да изчислим вероятността за терм, който липсва в документа d , то ще получим 0 и вероятността на цялата заявка ще бъде 0.

Нулевите вероятности са сериозен проблем при употребата на езикови модели, например при предсказване на следващата дума в разпознаване на реч, защото много думи ще се срещат рядко в тренировъчните данни.

Освен проблемът с нулевите вероятности, изчислявайки по този начин, думите, които се срещат в документа също получават лошо приближение. В частност, думите които се срещат точно веднъж получават по-високо приближение, тъй като тяхното присъствие понякога е случайно. Този проблем може да се реши чрез „изглаждане“ (smoothing).

Има много различни варианти за „изглаждане“ на вероятностни разпределения. В глава 11.3.2 вече разглеждахме добавяне на число (1 , $\frac{1}{2}$ или малко α) към наблюдаваните честоти на думите и нормализиране към новия общ брой. Ще разгледаме някои други методи за „изглаждане“, които комбинират наблюдаваните честоти на думите с по-общо вероятностно разпределение. Най-общо, дума, която не се среща в документа, трябва да е възможно да се срещне в заявкa, но вероятността и трябва да е близо, но не повече от вероятността да се срещне в цялата колекция от документи. Т.е. ако $tf_{t,d} = 0$, то:

$$\hat{P}(t|M_d) \leq cf_t / T$$

където cf_t е броят на срещанията на терма в колекцията, а T е броят на думите в цялата колекция.

В практиката често се използва комбинация от документно-зависимо мултиномино разпределение и мултиномино разпределение изчислено на базата на цялата колекция:

$$\hat{P}(t|d) = \lambda \hat{P}_{\text{mle}}(t|M_d) + (1 - \lambda) \hat{P}_{\text{mle}}(t|M_c)$$

където $0 < \lambda < 1$ и M_c е езиков модел построен от цялата колекция от документи.

По този начин се включва вероятността за срещане на терма в рамките на документа, както и общата честота на срещане на терма в цялата колекция от документи. Този модел се нарича линейно интерполиран езиков модел (Линейна интерполяция). Правилното определяне на λ е важно за доброто представяне на този модел.

Алтернативен метод за „изглаждане“ е да използваме езиков модел построен от цялата колекция от документи като априорно разпределение в Бейсов процес на обновяване (Бейсово „изглаждане“). Тогава получаваме:

$$\hat{P}(t|d) = \frac{tf_{t,d} + \alpha \hat{P}(t|M_c)}{L_d + \alpha}$$

И двата метода за „изглаждане“ се справят добре със задачи за извлечение на информация.

Степента на „изглаждане“ се контролира от параметрите α / λ – при малка стойност на λ или голяма на α имаме повече „изглаждане“. Променяйки тези параметри можем да оптимизираме представянето на модела. Когато имаме кратки заявки, по-подходящо е да „изглаждаме“ по-малко, и обратно – при големи заявки е полезно да използваме повече „изглаждане“.

Най-общо, класирането на резултати за заявка чрез базов езиков модел може да се разгледа като:

$$P(d|q) \propto P(d) \prod_{t \in q} ((1 - \lambda)P(t|M_c) + \lambda P(t|M_d))$$

Този израз дава вероятността, документът, който потребителят наистина търси да е d.

Експериментите на Ponte и Croft

Ponte и Croft (1998) публикуват първите експерименти за подхода на езиковия модел към извлечането на информация. Моделът, който те разглеждат е подобен на горе-описаният, но вместо комбинация на две мултиномни разпределения, те използват модел на Бернули с няколко променливи.

Използването на мултиномни разпределения е стандартно в повечето последващи публикации за езиковия модел и експерименталните данни дават основание да се твърди, че са по-успешни.

Ponte и Croft твърдят, че теглата на термовете, получени от езиковия модел са по-ефективни от тези получени чрез традиционния tf-idf метод (term frequency-inverse document frequency). На фигура 1 е показано малко подмножество от техните резултати, в което се сравнява представянето спрямо TREC теми 202-250 от TREC дискове 2 и 3. Заявките за с дължина на изречение и са от естествения език. Резултатите на езиковия модел са значително по-добри от

Precision			
Rec.	tf-idf	LM	%chg
0.0	0.7439	0.7590	+2.0
0.1	0.4521	0.4910	+8.6
0.2	0.3514	0.4045	+15.1 *
0.3	0.2761	0.3342	+21.0 *
0.4	0.2093	0.2572	+22.9 *
0.5	0.1558	0.2061	+32.3 *
0.6	0.1024	0.1405	+37.1 *
0.7	0.0451	0.0760	+68.7 *
0.8	0.0160	0.0432	+169.6 *
0.9	0.0033	0.0063	+89.3
1.0	0.0028	0.0050	+76.9
Ave	0.1868	0.2233	+19.55 *

тези на tf-idf.

Фигура 1. Резултати от сравнението между tf-idf и езиков модел направено от Ponte и Croft (1998). Табличата дава оценка за средна точност на оценяване маркирана с * по критерия на Уилококсън (Wilcoxon). Езиковият модел винаги се справя по-добре, но драстичните подобрения се наблюдават, когато имаме по-голяма степен на точност на връщане.

Езиковият модел в сравнение с други подходи за извлечане на информация

Езиковият модел предлага нов начин за разглеждане на проблема за извлечане на текст и е много полезен при обработка на език и реч. Най-големият проблем при него е намирането на приближение на документния модел, т.е. ефективното му „изглеждане“.

Езиковият модел предполага, че документите и заявките за информация са обекти от един и същи тип. В резултат, моделът е концептуално прост, прецизен и лек за изчисление. Той

прилича на XML извличането на данни по това, че и там заявките и документите са от един и същи тип.

От друга страна, като всички модели за извличане на информация, и той има негативни страни. Предположението, че документите и формулировката на информационната нужда (заявката) са еднотипни е нереалистично. Текущите подходи използвани езикови модели използват много прости модели на езика, обикновено унитрамни. Без изрична дефиниция за релевантност е трудно да се инкорпорира тази концепция в модела. Също така е трудно да се използва модел по-сложен от унитрамния, за да се включат понятия като фраза, пасаж или булеви оператори за извличане. Има разработки на езиковите модели, които се опитват да адресират тези проблеми.

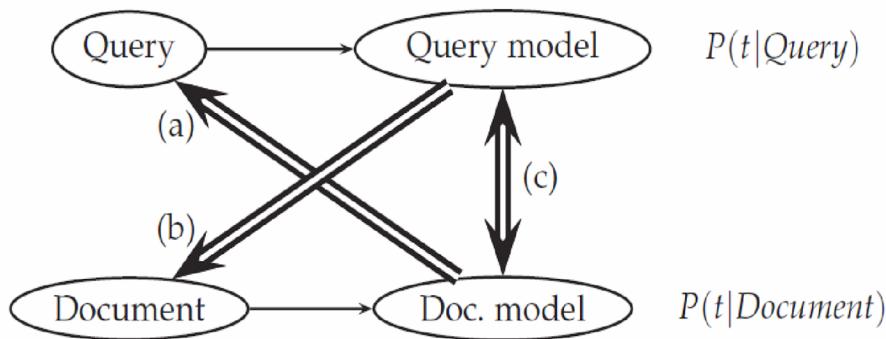
Езиковият модел е тясно свързан с tf-idf моделите. Честотата на термовете се представя директно в tf-idf моделите и много разработки са утвърдили важността на нормализацията на дължината на документа. Ефектът от комбинацията между вероятност за генериране на документ с вероятност за генериране на колекция е подобна на idf: термовете, които се срещат рядко в колекцията, но са често срещани в някои документи, ще имат по-голямо влияние върху оценката на документите.

От гледна точка на производителност, най-скорошните разработки показват, че езиковият модел е много ефективен и надминава tf-idf и BM25 (Best Match 25). Въпреки това, все още няма достатъчно доказателства, че той надминава по производителност добре-тренирано традиционно векторно пространство за извличане. (vector space).

Разширения на базата на подхода на езиковия модел

Има и други начини да се разглежда използването на езикови модели в извличането на информация. Вместо да разглеждаме вероятността на документов езиков модел M_d да генерира заявка, може да разглеждаме вероятността на езиков модел на заявка M_q да генерира документ. Основната причина да предпочитаме първия подход пред втория, е, че в заявката има много по-малко текст, чрез който да построим езиков модел. От друга страна, лесно е да се види как да включим обратна връзка за релевантност в такъв модел – може да разширим заявката с термове взети от релевантни документи и да обновим езиковия модел M_q . Чрез подходящи решения, този подход ще доведе до Бинарен независим модел (Binary Independence Model, глава 11).

Моделът на релевантност от Lavrenko и Croft (2001) е пример за модел изграден от вероятност на документа, който включва псевдо-обратна връзка за релевантност и постига много добри резултати.



Фигура 2. Трите подхода при разработката на езиков модел: (1) Вероятност на заявките, (2) Вероятност на документа, (3) Сравнение на моделите

Вместо директно да генерираме модел в една от двете посоки, можем да направим езиков модел от документта и заявката и след това да разглеждаме разликата между тези два езикови модела. Lafferty и Zhai (2001) представят тези три подхода при разглеждане на проблема (показани на фигура 2) и разработват общ подход за минимизиране на риска (general risk minimization approach) при търсене на документ. Например, един начин да се моделира риска е да върнем документ d като релевантен за заявка q разглеждайки отклонението (*divergence*) на Kullback-Leibler между респективните им езикови модели:

$$R(d; q) = KL(M_d \| M_q) = \sum_{t \in V} P(t|M_q) \log \frac{P(t|M_q)}{P(t|M_d)}$$

KL отклонението е асиметрична мярка за отклонение, произлизаша от теорията на информацията, която измерва колко лошо вероятностното разпределение M_q моделира M_d . (Cover and Thomas 1991, Manning and Schütze 1999). Lafferty и Zhai (2001) публикуват резултати, които дават основание да се смята, че подходът чрез сравнение на моделите се справя по-добре като от езиковия модел на заявките, така и от езиковия модел на документите. Основен недостатък на KL отклонението като функция за класификация е, че резултатите не са сравними при различни заявки.

Kraaij и Spitters (2003) предлагат алтернативно решение, което моделира близостта като нормализирано логаритмувано вероятностно отношение (или като разлика между взаимните ентропии).

Базовите езикови модели не разглеждат алтернативно изразяване (сионими) или отклонение в употребата на езика между заявките и документите. Berger и Lafferty (1999) предлагат модели за превод, за да преодолеят разликата в езика между заявка и документ. Моделът за превод дава възможност да се генерират думи за заявка, които не са в документа, чрез превод към алтернативни термове със сходно значение. Това дава основа за извличане на информация между различни езици.

Предполагаме, че моделът за превод може да се представи, чрез условно вероятностно разпределение $T(\cdot | \cdot)$ между термове от речника. Моделът за генериране на заявка за превод изглежда така:

$$P(q|M_d) = \prod_{t \in q} \sum_{v \in V} P(v|M_d) T(t|v)$$

Термът $P(v|M_d)$ е основният езиков модел на документа, а термът $T(t|v)$ прави превода.

Този метод е по-тежък изчислително и изисква построяването на модел за превод. Моделът за превод обикновено използва отделни ресурси като синонимен речник или двуезичен речник, но може да се построи използвайки колекцията документи.

Разширяването на подходите използвани езикови модели е зона на активна разработка. Като цяло, моделите за превод, моделите с обратна връзка за релевантност и подходът със сравнение на моделите са показвали, че подобряват производителността пред базовия модел на вероятност на заявките.

Публикации по темата:

- | | |
|--|---------------------------|
| Manning and Schütze (1999, Chapter 6). | Zhai and Lafferty 2002. |
| Jurafsky and Martin (2008, Chapter 4). | Kraaij et al. 2002. |
| Ponte and Croft 1998. | Liu and Croft 2004. |
| Hiemstra 1998. | Tao et al. 2006. |
| Berger and Lafferty 1999. | Hiemstra and Kraaij 2005. |
| Miller et al. 1999. | Gao et al. 2004. |
| Croft and Lafferty 2003. | Cao et al. 2005. |
| Hiemstra and Kraaij 2005. | Spärck Jones 2004. |
| Zhai and Lafferty 2001b. | Lafferty and Zhai 2003. |
| Zaragoza et al. 2003. | |

THE LEMUR TOOLKIT (<http://www.lemurproject.org/>)

13 Класификация на текст и Наивен Бейсов подход

До тук основно се разглежда моментално (*ad hoc*) извличане на информация, когато клиентите имат нужда от информация на момента и те я търсят, чрез задаване на един или повече въпроси на търсачката. Обаче, много потребители имат нарастващи информационни нужди. Например, може да желаят да следят разработките на *многоядриeni компютърни чипове*. Един вариант да се направи това е да се напише като въпрос: многояден AND компютър AND чип и да се търси всеки ден по индекса на новинарските статии. В тази и следващите две глави ще бъде разгледан въпросът: Как да се автоматизира тази повтаряща се задача? За тази цел много системи поддържат *продължителни въпроси*. Продължителните въпроси са подобни на всички други въпроси, разликата е, че те периодично се изпълняват върху корпус/колекция, в който се прибавят нови документи с течение на времето.

Ако продължителният въпрос е само многояден AND компютър AND чип, вероятно ще бъдат пропуснати много статии по същата тема, които съдържат други термини, напр. *многоядриeni процесори*. За постигане на добри резултати продължителните въпроси трябва да се прецезират с течение на времето и да стават все по-сложни. В горния пример, може да се приложи Буlevия метод на търсене със следния въпрос (многояден OR много-ядрен) AND (чип OR процесор OR микропроцесор).

За да се схване същността на проблема, който създават продължителните въпроси, ще бъдат въведени основните понятия за проблема класификация. Нека са дадени набор от класове, търси се класа, към който ще принадлежи даден обект. В примера с продължителния въпрос, той служи за групиране на новите статии в два класа: документи за *многоядриeni компютърни чипове* и документи, които не са за *многоядриени компютърни чипове*. Това е т. нар. двукасов класификация. Класификацията използвана от продължителните въпроси се нарича още рутинна или филтрираща и ще се дискутира в Секция 15.3.1.

Не е необходимо класът да е тясно фокусиран към продължителния въпрос *многоядриени компютърни чипове*. Често, класът е с по-общо съдържание, като "Китай" или "кафе". Такива по-общи класове, обикновено, се причисляват към теми и следователно класификационната задача се нарича *классификация на текст, категоризация на текст, тематична класификация* или *тематично търсене*. На Фигура. 13.1 е показан пример за Китай. Продължителните въпроси и темите се различават по степента си на специфичност, но методите за избиране на маршрута, филтриране и класификация са едни и същи. Следователно маршрутизирането и филтрирането ще бъдат включени в рубриката класификация на текст в тази и следващите глави.

Идеята за класифициране е основна и има множество приложения в и извън извличането на информация (IR). Например, при компютърната визуализация може да се използва класификатор за разделяне на изображенията по класове като – пейзаж, портрет и други. Тук ще бъдат разгледани примери от IR:

- Необходими са няколко предварителни стъпки при индексирането, което се дискутираше в глава 2: откриване на декодера на документа (ASCII, Unicode UTF-8, други); сегментация на думите (али празното място/ западната между две букви е

разделител на две думи или техническа грешка?); истинското рамкиране (truecasing) и идентифициране на езика на документа.

- Автоматичното разпознаване на спам страниците (които да не се включват в метода за индексиране на търсачката).
- Автоматичното определяне на документи с сексуално съдържание (което се включва в резултатите от търсениято само ако потребителят е изключил опция SafeSearch).
- Емоционално определяне или автоматично класифициране мнения за филм или продукт като положителен или отрицателен. Прилага се например, ако потребителят търси отрицателни мнения преди да си купи камера за да е сигурен, че няма да има нежелани усложнения или качествени проблеми.
- Персонално сортиране на електронна поща (имейли). Потребителят може да има папки като “обяди”, “електронни сметки”, имейли от “семейство и приятели” и други подобни и желае класификатор, който да класифицира всеки входящ имайл и автоматично да го подрежда в подходящата папка. По-лесно е да се намират съобщенията в подредени папки отколкото в много голяма кутия за входяща поща (Inbox). Най-често среяното подобно приложение е папката за спам, в която се поставят автоматично всички съобщенията, за които се предполага, че са спам.
- Специфично-тематично или вертикално търсене. Вертикалното търсене се използва при рестриктивни търсения на специфични теми. Например въпрос за компютърни науки при метода на вертикалното търсене за тема Китай ще върне списък от факултети по компютърни науки в Китай с по-висока прецизност и и след това ще направи основно търсене по въпроса. Този резултат ще се получи, заподобно методът на вертикалното търсене не включва страници, които съдържат различните значения на термина Китай (напр. относно порцелана), но включва подобни по съдържание документи, дори те да не споменават точно термина Китай.
- Накрая, подреждащата функция в IR може също да се базира на класификатор на документи както е обяснено в Секция 15.4.

Гореизброеният списък показва важното значение на класификацията в IR. Повечето системи за извличане на информация съдържат много компоненти, които използват някаква форма за класифициране. В този учебник се използва класификация на текста, напр.

Компютърът не е най-важния инструмент за класифициране. Много класификационни задачи са решени мануално/ръчно. Книгите в библиотеката, предназначени за Библиотеката на Конгреса са категоризирани от библиотекар. Но ръчното класифициране е по-скъпо. Примерът за търсениято на *многоядрени компютърни чипове* показва един алтернативен подход: класификация, чрез използване на продължителните въпроси – които могат да се разглеждат като правила – най-често писани на ръка. Подобно на примера (*многояден OR много-ядрен*) AND (*чип OR процесор OR микропроцесор*), правилата понякога са еквивалентни на Буlevи представления.

Правилото определя основната комбинация от думи, които описват класа. Ръчно кодираните правила притежават добри скалиращи свойства, но създаването им и управлението им продължително време изисква много труд. Специалист с технически умения (главен експерт, който е добър в писане на регулярни изрази) може да създаде множество от правила, което

конкурира или надминава точността на автоматично генерираните класификатори, които ще бъдат дискутирани накратко, обаче, е трудно да се намери човек с такива умения.

Освен мануалната класификация и ръчно изработените правила има и трети подход за класификация, наречен машинно самообучаваща се, базирана на текст класификация. Върху този подход е фокусът на следващите няколко глави. В машинното самообучение наборът от правила или по-общо критерият за класификация на текст автоматично се обучава от трениращи данни. Този подход се нарича още статистическа текстова класификация, ако обучаващият метод е статистически. При статистическата класификация на текст се изисква определен брой примерни документи (или обучаващи документи) за всеки клас. Необходимостта от мануална класификация не е елиминирана, защото трениращите документи се подготвят от съответен специалист, който ги описва / поставя им етикети така, че всеки документ да попадне в съответния за него клас. Но поставянето на етикети е по-лесна задача от писането на правила. Не са необходими специални умения, за да се разгледа даден документ и да се реши дали той се отнася за Китай или не. Понякога такова поставяне на етикети е вече безусловна част от съществуващия работен процес. Например, може да се преглеждат новите статии върнати от продължителния въпрос всяка сутрин и да се дава подходяща информация като обратна връзка (вж. Глава 9) чрез преместване на документите, които съответстват на търсенето в специална папка “*многоядриeni процесори*”.

Тази глава се започва с въведение в проблемите на текстовата класификация, включващи формална дефиниция; след това ще се разгледа Наивния Бейсов подход (опростен алгоритъм на Бейс - NayveBayes) - НБ, прост и ефективен класификационен метод. Всички алгоритми за класификация, които се разглеждат тук представят документите в многомерното пространство. За подобряване ефикасността на тези алгоритми, е желателно да се редуцира размерността на пространството, за което се най-често се прилага в текстовите класификатори техника, позната като селекция на характеристиките/ feature selection.

13.1 Проблемът класификация на текст

В текстовата класификация е дадено, описането $d \in \square$, където \square е документното пространство, и фиксиран набор от класове $\square = \{c_1, c_2, \dots, c_j\}$. Класовете се наричат още категории или етикети. Обикновено, документното пространство \square е многомерно пространство и класовете са определени ръчно според нуждите на търсенето, както в примера за Китай и документите, в които се разглежда многоядриeni компютърни чипове. Задава се обучаващ набор \square от етикетирани документи $\langle d, c \rangle$, където $\langle d, c \rangle \in \square \times \square$. Например:

$\langle d, c \rangle = \text{Пекин се присъединява към Световна Търговска Организация, Китай}$

За документ от едно изречение “Пекин се присъединява към Световна Търговска Организация(Beijing joins theWTO), Китай” и класа/етикета “Китай”.

Чрез прилагане на обучаващ метод или обучаващ алгоритъм се обучава класификатора или класифициращата функция γ , която свързва документите с класовете:

(13.1) $\gamma: \square \rightarrow \square$

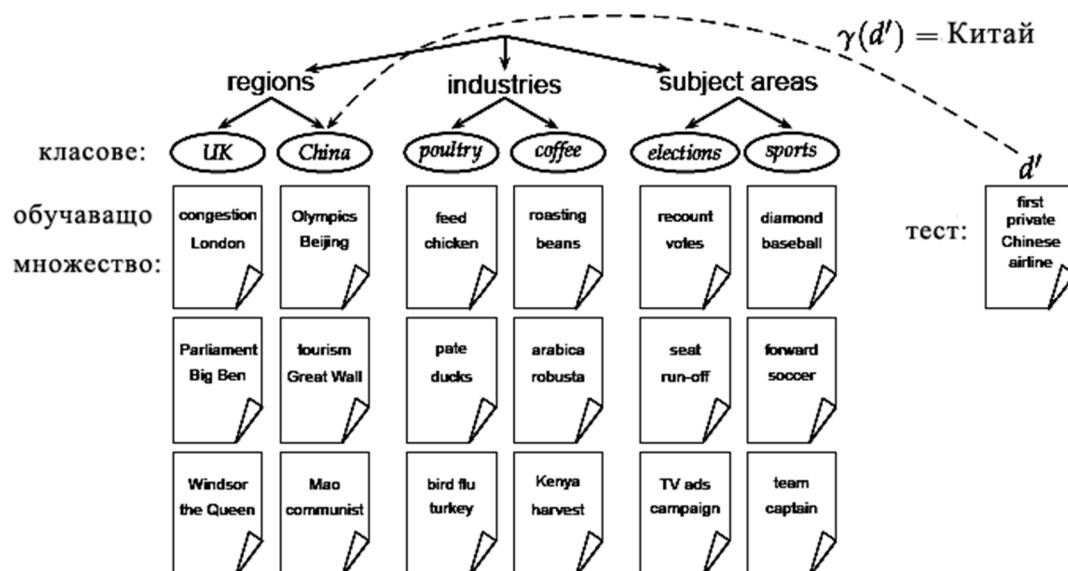
Такъв тип обучение се нарича обучение с учител (supervised learning), учителят (човекът, който дефинира класовете и етикетите на обучаващите документи) управлява обучаващия процес.

Методът на обучение с учител се отбелязва с Γ и се записва като $\Gamma(\square) = \gamma$. Обучаващия метод Γ взема обучаващия набор \square като входен и връща обучената класификационна функция γ .

Повечето имена на обучаващите методи Γ се използват и за класификаторите γ . Когато се казва, че “Наивния Бейсов (НБ) обучаващ метод Γ е силен”, се има предвид, че той може да се прилага за много различни обучаващи проблеми и е малко вероятно да създаде класификатори, които катастрофално да се провалят. Но когато се казва, че “НБ дава около 20 % грешка”, се описва експеримент, в който съответният НБ класификатор γ (който се е получил в резултат на НБ обучаващия метод) дава 20% грешка при прилагането му.

Фигура 13.1 показва примера за текстова класификация от the Reuters – RCV1 корпус, представен в Секция 4.2. Представени са шест класа (САЩ, Китай, ..., спортове), всеки с по три обучаващи документа. Дадени са по няколко мнемонични думи за съдържанието на всеки документ. Обучаващият набор осигурява някои типични примери за всеки клас, така че да може да се обучи класифициращата функция γ .

Веднъж обучена γ , може да се прилага върху тестовото множество (или тестовите данни), чийто клас е непознат, напримерновия документ “Първата частна китайска авиолиния”.



Фигура. 13.1 Класове, обучаваща и тестова група в текстов класификатор

На фиг. 13.1 класифициращата функция поставя новия документ в класа $\gamma(d) = \text{“Китай”}$, което е и коректното назначение.

Класовете в текстовата класификация често имат интересна структура, например йерархична (Фигура 13.1). Дадени са по два примера за всяка от регионалните категории, индустриалните категории и субективните категории. Йерархията може съществено да помогне решаването на класификационен проблем (вж. Секция 15.3.2). Дотогава, ще се приема, в главите за текстови класификатори, че класовете формират групи, в които няма субгрупови взаимодействия.

Изображението дефинирано чрез 13.1 показва, че всеки документ е член точно на един клас. Моделът от Фигура 13.1 не е най-подходящия модел за йерархия. Всъщност, документа за Олимпиадата през 2008 би трябвало да е член едновременно на два класа: “Китай” и

“спортове”. Този класификационен проблем ще бъде дискутиран по-нататък в Секция 14.5. За момента ще се разглеждат само случаите, в които всеки документ принадлежи само /на един клас.

Целта на текстовата класификация е голяма точност върху тестови данни или нови данни – например, новите статии всяка сутрин за примера за *многоядения чип*. Лесно е да се постигне висока точност върху тестовото множество (достатъчно е да се запомнят етикетите). Но високата точност върху обучаващото множество не гарантира добре работещ класификатор. Когато се използва обучаваща група, се приема, че обучаващия и тестовия набор са подобни или имат подобно разпределение. Прецизната дефиниция на това понятие е представена в Секция 14.6.

13.2 Текстов класификатор по Наивно-Бейсов подход

Първият обучаващ метод с учител, който ще бъде въведен е полиномен Наивен Бейс, той е вероятностен обучаващ метод. Вероятността документ **d** да е в клас **c** се изчислява като:

$$(13.2) P(c|d) \propto P(c) \prod P(t_k|c)$$

$$1 \leq k \leq n_d$$

където $P(t_k|c)$ е условната вероятност на терминът t_k , наблюдаван в документ от клас **c**. $P(t_k|c)$ може да се интерпретира като мярка, за това до колко t_k доказва, че **c** е във верния клас . $P(c)$ е вероятността документът да се намира в клас **c**. Ако термините в документа не осигуряват ясно указание за един клас спрямо друг, се избира класа с по-висока вероятност t_1, t_2, \dots, t_{n_d} са думи в **d**, които са част от речника, използван при класификацията и n_d е броят на тези думи в **d**. Например, $\langle t_1, t_2, \dots, t_{n_d} \rangle$ за документа от едно изречение ВејпандТайрејjointheWTO могат да бъдат Вејп, Тайре, join WTO), с $n_d = 4$, ако термините “and” и “the” се третират като стоп-думи.

Целта при класифицирането на текст е да се намери най-подходящият клас за документа. Най-подходящият клас в НБ е най-вероятният или максимално апостериорния (индуктивния) МАР клас **c_{map}**:

$$(13.3) c_{\text{map}} = \operatorname{argmax}(c|d) = \operatorname{argmax} \propto \prod P(t_k|c).$$

$$c \in C \quad 1 \leq k \leq n_d$$

Написано е вместо **P**, защото истинските стойности на параметрите **P(c)** и **P(t_k|c)** не са известни, но тези параметри са оценени чрез обучаващото множество, както ще се разбере след малко.

В уравнение (13.3) много условни вероятности се умножават по една за всяка позиция $1 \leq k \leq n_d$. Поради което може да се получи резултат с плаваща точка, препълнен отдолу. Следователно е по-добре да се изпълнят изчисленията чрез прибавяне на логаритми от вероятностите отколкото последните да се умножават. Класът с най-висок резултат от логаритъма на вероятността е най-вероятен; $\log(xy) = \log(x) + \log(y)$ и логаритмичната функция е монотонна. Максимумът, който се търси в повечето превръщания в НБ е:

$$(13.4) c_{\text{map}} = \operatorname{argmax} [\log \propto + \square \log (t_k|c)]$$

$$c \square C \quad 1 \leq k \leq n_d$$

Уравнение (13.4) има праста интерпретация. Всеки условен параметър $\log(t_k | c)$ е тегло, което посочва колко добър е индикатора t_k за c . По подобен начин \log дава теглото, което определя относителната честота на c . Класовете с по-голяма честота е по-вероятно да са коректните класове от тези с по-малка честота. Сумата от логаритмите и теглата на термините е мярка за попадането на документа в подходящия клас и уравнение (13.4) избира класа, за който има най-голяма стойност.

Първоначално се работи с интуитивната интерпретация на полиномния НБ модел и се отлага формалното получаване за Секция 13.4.

Как се оценяват параметрите (c) и $(t_k | c)$? Първоначално се прави опит да се оцени максималната вероятност (MLE, Секция 11.3.2), която е просто относителната честота и съответства на най-правдоподобната стойност за всеки параметър от обучаващите данни. За тази оценка е:

$$(13.5) \quad \mathbb{C} = N_c / N$$

където N_c е номерът на документа от клас c , а N е общия брой документи.

Условната вероятност $(t | c)$ се оценява като относителна честота на термина t в документите, принадлежащи към клас c :

$$(13.6) \quad P(t | c) = T_{ct} / \sum_{t' \in V} T_{ct'}$$

където T_{ct} е броят на намерените термини t в обучаващите документи от клас c , включително и многократното намиране на термина в документа. Тук се прави допускане за позиционна независимост, което ще бъде дискутирано детайлно в следващата секция: T_{ct} е брояч за намиране във всички позиции k в документите от обучаващото множество. Следователно, не може да се изчисляват различни оценки за различните позиции, т.е. ако например думата се среща два пъти в документа на позиции k_1 и k_2 , $(k_1 | c) = (k_2 | c)$.

Проблемът при MLE оценяването е, че тя е нула за комбинацията термин - клас, която не се среща в обучаващите данни. Ако терминът "Световна Търговска Организация" (WTO) се намира само в документи "Китай" в обучаващия набор, MLE оценката му за другите класове, напр. Великобритания, ще бъде нула:

$$(WTO | Великобритания) = 0$$

В този случай, документът от едно изречение: "Британия е член на Световната Търговска Организация (WTO)." ще има условна вероятност равна на нула за Великобритания, защото условните вероятности за всички термини в уравнение (13.2) се умножават.

Обучаващ Многочленен НБ (\square, \square)

1. $V \square$ ИзвадиРечник(\square)
2. $N \square$ ПребройДокументи (\square)
3. за всяко $c \square \square$
4. $\text{do} N_c \square$ БройДокументиВКласа (\square, c)
5. prior $[c] \square N_c / N$
6. $text_c \square$ КонкатенацияНаВсичкиТекстовеВКласа (\square, c)
7. за всяко $t \square \forall$

8. **do** T_{ct} **бройХарактеристикиНаТермини** ($text_c, t$)
9. за всяко $t \in V$
10. **do** $cond\ prob [t] [c] = (T_{ct} + 1) / \sum_v (T_{cv} + 1)$
11. **return** $V, prior, cond\ prob$

Приложен/ПолиномиаленНБ ($\square, V, prior, condprob, d$)

1. $W \leftarrow \text{ИзвадиХарактеристикитеОт Документите} (V, d)$
2. за всяко $c \in C$
3. **do** $score [c] = \log prior [c]$
4. за всяко $t \in W$
5. $score [c] += \log cond\ prob [t] [c]$
6. **return** $\arg \max_c score [c]$

Фигура. 13.2 Найвен Бейсов Алгоритъм (полиномиален модел): Обучение и тестване

Ясно е, че моделът би трябвало да определи с висока вероятност документа към класа “Великобритания” заради наличието на термина “Британия”. Проблемът с нулевата вероятност за WTO не може да се заобиколи без значение колко много данни има за класа “Великобритания” от другите статии. Оценката е нула поради разпръснатостта: обучаващите данни никога не са толкова големи, че да могат да обхванат адекватно честотата на рядко срещаните случаи, напр. честотата на срещане на WTO в документите “Великобритания”.

За елиминиране на нулевите стойности се прибавя единица или се прилага Лапласовото изглаждане, което просто прибавя единица към всяко броене:

$$(13.7) \quad (t | c) = (T_{ct} + 1) / (\sum_v T_{cv} + B),$$

където $B = |V|$ е броят на термините в речника. Изглаждането чрез прибавяне на единица може да се приеме като постоянен помощник (всеки термин се среща еднократно във всеки клас), така че обновяването е данна за обучаващата група. Обърнете внимание, че приорната вероятност за наличието на термина е противоположна на приорната вероятност за класа, която се оценява чрез уравнение (13.5) на документално ниво.

Таблица 13.1 Данни за параметрична оценка на примери

$\frac{1}{\text{на}} \text{Думи в документа}$	B	c	$=$
документа	Китай?		
Обучаваща група	Китайски Пекин Китайски		Да
	Китайски Китайски Шанхай		Да
	Китайски Макао		Да
	Токио Япония Китайски		Не
Тестова група	Китайски Китайски Китайски Токио	?	
	Япония		

Таблица 13.2 Обучаващи и тестови времена за НБ

Метод	Времева сложност
Обучаващ	$\square(\square L_{ave} + \square V)$
Тест	$\square(L_a + \square M_a) = \square(\square M_a)$

Така всички елементи, които са необходими за обучение са налични и може да се приложи НБ класификатора. Пълният алгоритъм е описан на Фигура 13.2.

Пример 13.1: За примера в Таблица 13.1, полиномните параметри, за които трябва да се класифицират тестовите документи са приорните вероятности $C = \frac{3}{4}$ и $O = \frac{1}{4}$ и следните условни вероятности:

$$\begin{aligned}(C|c) &= (5+1)/(8+6) = 6/14 = 3/7 \\ (Tokyo|c) &= (Japan|c) = (0+1)/(8+6) = 1/14 \\ (Chinese|) &= (1+1)/(3+6) = 2/9 \\ (Tokyo|) &= (Japan|) = (1+1)/(3+6) = 2/9\end{aligned}$$

Знаменателите са $(8+6)$ и $(3+6)$ защото дълчините на **text_c** и **text** са 8 и 3, съответно, а константата **B** в уравнение (13.7) е 6 тъй като речникът се състои от шест термина.

$$\begin{aligned}(c|d_5) &\square \frac{3}{4} \cdot (3/7)^3 \cdot 1/14 \cdot 1/14 \square 0.0003 \\ (|d_5) &\square \frac{1}{4} \cdot (2/9)^3 \cdot 2/9 \cdot 2/9 \square 0.0001\end{aligned}$$

По този начин класификаторът определя тестовия документ към **C** = "Китай". Причината за това класификационно решение е, че трите стойности на положителния индикатор "Китайски" в d_5 натежават спрямо двата негативни индикатора за "Япония" и "Токио".

Какво представлява времевата сложност в НБ? Сложността при изчисление на параметрите е $\square(|\square||V|)$, тъй като параметричната група съдържа $|\square||V|$ условни и $|\square|$ приорни вероятности. Неизбежната предварителна обработка за изчисляване на параметрите (извлечане от речника, броене на термините и др.) може да се направи с едно обхождане (onpass) на обучаващите данни. Времевата сложност на този компонент, следователно, е $\square(|\square|L_{ave})$, където $|\square|$ е броят на документите, а L_{ave} е средната аритметична стойност от дълчините на документите.

Тук $\square(|\square|L_{ave})$ се използва за означение на $\square(T)$, където **T** е дължината на обучаващия корпус. Това е нестандартно, $\square(.)$ не е дефинирано за средната аритметична стойност.

Времевата сложност на ПриложенПолиномиаленНБ, фиг. 13.2 е $\square(|\square|L_a)L_a$ и M_a са броят на характеристиките и типовете, съответно, в тествания документ. ПриложенПриложенНБ може да се модифицира така, че да стане $\square(L_a + |\square| M_a)$ (Упражнение 13.8). Приема се, че дължината на тестваниите документи е ограничена, $\square(L_a + |\square| M_a) = \square(|\square| M_a)$, защото $L_a < b |\square| M_a$ за фиксирана константа **b**.

В таблица 13.2 са обобщени времевите сложности. Основно, има $|\square||V| < |\square|L_{ave}$, така че и двете обучаваща и тествания сложност са линейни за времето, необходимо за сканиране на данните. Поради необходимостта за поне еднократно търсене в данните, може да се каже, че

НБ има оптимална времева сложност. Неговата ефикасност е една от причините за популярността на НБ като метод за текстова класификация.

13.2.1 Връзка с полиномния универсално граматиченезиков модел

Полиномния НБ модел е формално идентичен с полиномния универсално граматиченезиков модел (Секция 12.2.1). Уравнение (13.2) е частен случай на Уравнение (12.12), дадено по-долу при $\square = 1$:

$$P(d|q) \square P(d) \prod P(t|M_d)$$

$$t \square q$$

Документът **d** в класификацията на текст (Уравнение (13.2)) играе ролята на въпроса в езиковото моделиране (Уравнение (13.8)) и класовете **c** в текстовата класификация са в ролята на документите **db** езиковото моделиране. Уравнение (13.8) се прилага за подреждане на документите съгласно вероятността на тяхното съответствие с въпроса **q**. В НБ класификацията, обикновено, интерес представляват само класовете с висок ранг (top-ranked).

Прилагат се MLE оценките от Секция (12.2.2) и се среща проблема с нулевите оценки, дължащи се на редките данни; но вместо изглаждане чрез прибавяне на единица (add-onessmoothing), тук се използва съчетание на две разпределения за решаването на проблема. Изглаждането чрез прибавяне на единица е подобно на изглаждането чрез прибавяне на -1/2/+1/2 в Секция 11.3.4.

Упражнение 13.1

Защо се очаква $| \square | |V| < | \square | L_{ave}$ в таблица 13.2 да е в сила за повечето текстови корпуси?

Обучаващ Бернули НБ (\square, \square)

1. $V \square$ Екстрагирай Речника (\square)
2. $N \square$ Преброй Документите (\square)
3. за всяко $c \square \square$
4. **do** $N_c \square$ Преброй Документите В Класа (\square, c)
5. *prior* [c] N_c/N
6. за всяко $t \square V$
7. **do** $N_{ct} \square$ Преброй Термините В Документа От Класа (\square, c, t)
8. *cond prob* [t][c] $(N_{ct} + 1)/(N_c + 2)$
9. **return** $V, prior, cond prob$

Приложен Бернули НБ ($\square, V, prior, cond prob, d$)

1. $Vd \square$ Екстрагирай Термините От Документа (V, d)
2. за всяко $c \square \square$
3. **do** $score [c] \square log prior[c]$
4. за всяко $t \square V$
5. **do if** $t \square Vd$
6. **then** $score [c] += log cond prob[t][c]$
7. **else** $score [c] += log(1 - cond prob[t][c])$
8. **return** $\arg \max_c score[c]$

Фигура 13.3 НБ алгоритъм (модел на Бернули): обучение и тестване.

Изглаждането чрез прибавяне на единица в Ред 8 (горе) е аналогично на Уравнение (13.7) при $B = 2$.

13.3 Модел на Бернули

Съществуват два различни варианта, по които може да се организира НБ класификатора. Моделът, представен в предната секция е полиномиален модел. Той генерира по един термин от речника за всяка позиция в документа и допуска генеративния модел, който ще бъде дискутиран по-детайлно в секция 13.4.

Алтернатива на полиномиален модел е многовариантният модел на Бернули или моделът на Бернули. Той е еквивалентен на бинарния независим модел от Секция 11.3, който генерира индикатор за всеки термин от речника, като с 1 се означава наличието на термина в документа, а с 0 – липсата му. На фигура 13.3 са представени обучаващия и тестващия модел на Бернули. Моделът на Бернули е със същата времева сложност, както и многочленния модел.

Различните модели предполагат генериране на различни оценъчни стратегии и на различни класификационни правила. Моделът на Бернули оценява $(t|c)$ като фракция от документи от клас **c**, които съдържат термина **t** (фигура 13.3, ОбучаващБернулиНБ, ред 8). За разлика от него, полиномиалният модел оценява $(t|c)$ като фракция на характеристиките или фракция на позициите в документите от клас **c**, които съдържат термина **t** (Уравнение 13.7)). При класифициране на тест-документ моделът на Бернули прилага бинарната проверка на информациите, игнорирайки броят на случаите, докато полиномиалният модел следи многоократните случаи. В резултат на това, моделът на Бернули, обикновено, прави много грешки при класифициране на дълги документи. Напр. въведената книга може да е определена за класа “Китай”, защото веднъж е бил използван терминът Китай.

Моделите, също така, се различават по действието си спрямо неоткрити в класификацията термини. Последните няма да окажат влияние върху класификационното решение в полиномния модел; но в модела на Бернули вероятността за неоткриване е множител, в който се изчислява $P(c|d)$ (Фигура 13.3 ПриложенБернулиНБВ, ред 7). Така е, защото само моделът на Бернули моделира ясно отсъствието на термините.

Упражнение 13.2: Да се приложи моделът на Бернули върху примера в Таблица 13.1. Оценките за приорната вероятност да бъдат същите, както преди: $C = \frac{3}{4}$ и $O = \frac{1}{4}$. Условните вероятности са:

$$(\text{Китайски}|c) = (3+1)/(3+2) = 4/5$$

$$(\text{Япония}|c) = (\text{Токио}|c) = (0+1)/(3+2) = 1/5$$

$$(\text{Пекин}|c) = (\text{Макао}|c) = (\text{Шанхай}|c) = (1+1)/(3+2) = 2/5$$

$$(\text{Китайски}|) = (1+1)/(1+2) = 2/3$$

$$(\text{Япония}|) = (\text{Токио}|) = (1+1)/(1+2) = 2/3$$

$$(\text{Пекин}|) = (\text{Макао}|) = (\text{Шанхай}|) = (0+1)/(1+2) = 1/3$$

Делителите са $(3+2)$ и $(1+2)$, защото има 3 документа в **c** и един документ в **c**, а константата **B** в Уравнение (13.7) е 2 – има две варианта при разглеждане на всеки термин: “присъства – не присъства”.

Резултатите за тестовите документи за двета класа са:

$$(c|d_5) \square \odot . (\text{Китайски}|c) . (\text{Япония}|c) . (\text{Токио}|c) . (1 - (\text{Пекин}|c)) . (1 - (\text{Шанхай}|c)) . (1 - (\text{Макао}|c)) = \frac{3}{4} . \frac{4}{5} . \frac{1}{5} . \frac{1}{5} . (1 - \frac{2}{5}) . (1 - \frac{2}{5}) . (1 - \frac{2}{5}) \square 0.005$$

И аналогично:

$$13.3.1.1 P(|D_5) \square \frac{1}{4} . \frac{2}{3} . \frac{2}{3} . \frac{2}{3} . (1 - \frac{1}{3}) . (1 - \frac{1}{3}) . (1 - \frac{1}{3}) \square 0.022$$

Така класификаторът отбелязва тестовия документ за \neq не-“Китай”. Когато се гледа само бинарното разпределение и не се отчита честотата на появя на термините, “Япония” и “Токио” са индикатори за $(2/3 > 1/5)$ и условните вероятности за “Китайски” за c и не се различават достатъчно, за да повлияят класификационното решение.

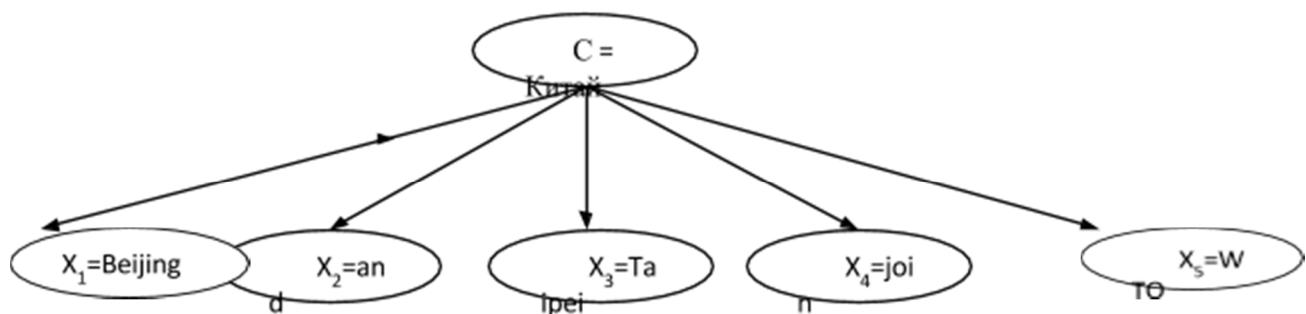
13.4 Свойства на НБ

За по-добро разбиране на двета модела и на допусканията, които те правят, нека се върнем назад и проверим как сме обяснявали техните класификационните правила в Глави 11 и 12. Документът се поставя в класа, за който е получена максимална апостериорна вероятност (Секция 13.3.2), която се изчислява по следния начин:

$$\begin{aligned} c_{\text{map}} &= \arg \max_{c \square \square} P(c|d) \\ (13.9) \quad &= \arg \max_{c \square \square} P(d|c) P \odot / P(d) \end{aligned}$$

$$(13.10) = \arg \max_{c \square \square} P(d|c) P \odot,$$

където Бейсовото правило (Уравнение 11.4) се поставя в (13.9) и знаменателят се отрязва в последната стъпка, затова $P(d)$ е един и същ за всички класове и не влияе върху argmax.



Фигура 13.4 Полиномиален модел на НБ

Уравнение (13.10) може да се разглежда като описание на генеративния процес при Бейсовата текстова класификация. За да се генерира документ, първо се избира клас c с вероятност $P(c)$ (Фигура 13.4 и 13.5). Двета модела се различават по формализацията на втората стъпка, генерирането на документ от даден клас в съответствие с условното разпределение $P(d|c)$:

$$(13.11) \quad \text{Полином } P(d|c) = P(t_1, \dots, t_k, \dots, t_{nd} | c)$$

$$(13.12) \quad \text{Бернули } P(d | c) = P(\square e_1, \dots, e_i, \dots, e_M | c),$$

където $\square t_1, \dots, t_{nd} \square$ е секвенция на термините, които се наблюдават в d (минус термините, които са изключени от речника) и $\square e_1, \dots, e_M \square$ е бинарен вектор с размерност M , който определя за всеки термин, дали се наблюдава в d или не.

Сега би трябвало да е по-ясно защо се въвежда пространство с документи \square в Уравнение (13.1) при дефиниране на класификационния проблем. Критичната стъпка при решаване на текстовия класификационен проблем е изборът на представителна извадка от документи. $\square t_1, \dots, t_{nd} \square$ и $\square e_1, \dots, e_M \square$ са две различни представителни извадки от документи. В първия случай \square е множество от всички секвенции на термините. Във втория случай \square е $\{0, 1\}^M$.

Уравнение (13.11) и (13.12) не могат да се използват директно за класификация на текст. За модела на Бернули, ще се оценяват $2^M |\square|$ различни параметъра, по един за всяка възможна комбинация от M -стойности e_i и клас. Броят на параметрите в случая с полинома е от същия порядък като големината му. При такива големи размери надеждното оценяване на тези параметри е невъзможно.

За да се намали броят на параметрите, се прави Наивно Бейсово допускане за условна независимост. Приема се, че стойностите на характеристиките са независими за всеки друг даден клас:

$$(13.13) \quad \text{Многочлен } P(d | c) = P(\square t_1, \dots, t_{nd} | c) = \prod_{k=1}^{n_d} P(X_k = t_k | c)$$

$$1 \leq k \leq n_d$$

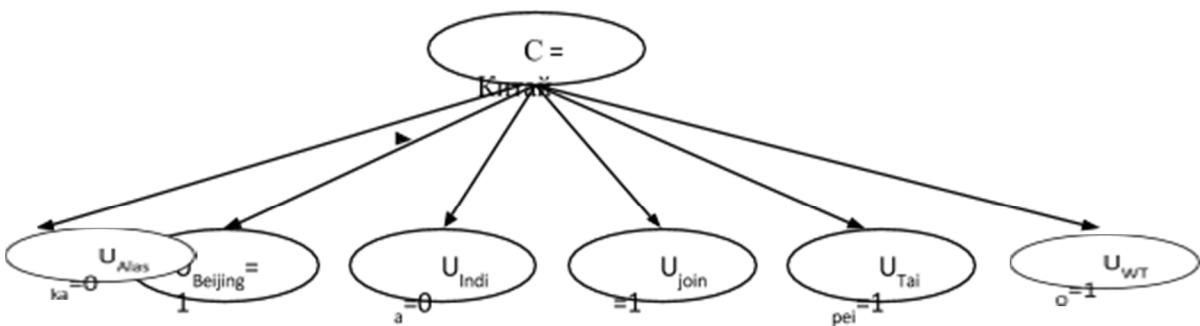
$$(13.14) \quad \text{Бернули } P(d | c) = P(\square e_1, \dots, e_M | c) = \prod_{i=1}^M P(U_i = e_i | c)$$

$$1 \leq i \leq M$$

Въвеждат се две случаини променливи, за да се изградят два различни генеративни модела. X_k е случаина променлива за позиция k в документа и приема стойностите на термините от речника. $P(X_k = t | c)$ е вероятността в документа от клас c терминът t да се намира на позиция k . U_i е случаина променлива за термина i от речника и приема стойност 0 (липсва) или 1 (присъства). $P(U_i = 1 | c)$ е вероятността в документ от клас c терминът t_i да се намира на всяка позиция повече от един път.

Допускането за условна независимост е илюстрирано на Фигури 13.4 и 13.5. Класът "Китай" генерира стойности за всяка от петте характеристики на термините (полином) или шест бинарни характеристики (Бернули) с централна вероятност, независими от стойностите на другите атрибути. Фактът, че документа от клас "Китай" съдържа термина "Тайпе" не дава вероятността дали същия документ съдържа и "Пекин".

В реалността, допускането за условна независимост не се поддържа за текстови данни. Термините са условно зависими един от друг. Но както ще се дискутира след малко, НБ моделите дават добри резултати при прилагане на допускане за условна независимост.



Фигура 13.5 НБ модел на Бернули

Въпреки допускането за условна независимост, все още има прекалено много параметри за полиномния модел, ако се приеме, че всяка позиция k в документа има различно вероятностно разпределение. Позицията на термина в документа сама по себе си не носи информация за класа. Въпреки разликата в словосъчетанията “Китай съди Франция” и “Франция съди Китай”, наличието на “Китай” в позиция 1 в първия документ спрямо позиция 3 във втория не се използва в НБ класифицирането защото всеки термин се разглежда отделно.

Ако се приеме, че термините имат различно разпределение за всяка позиция k , би трябвало да се оценява различно множество от параметри за всяко k . Вероятността “зърно” (bean) да е първи термин в документ “Кафе” (coffee) може да се различава от вероятността да е втори термин и т.н. Това отново предизвиква проблеми при оценяване на разпръснати данни.

Поради тази причина се прави второ допускане за независимост за полиномния модел - позиционна независимост. Условните вероятности за термина са еднакви, независимо от позицията му в документа:

$$P(X_{k1} = t | c) = P(X_{k2} = t | c)$$

за всички позиции k_1, k_2 , термини t и класове c . Следователно, има единствено разпределение на термините, което е валидно за всички позиции k_i и X може да се използва за обозначаването му. Позиционната независимост е еквивалентна на модела “торба с думи”, който е въведен в контекста на специалното извлечане в Глава 6.

С допусканятия за условна и позиционна независимост е необходимо само да се оценят $\square(M | \square)$ параметри $P(t_k | C)$ (полиномен модел) или $P(e_i | c)$ (модел на Бернули), по една за всяка комбинация термин-клас, а не броят, който е експонента на M , големина на речника. Допусканятия за независимост намаляват броя на оценяваните параметри неколократно.

Таблица 13.3 Полиномен модел спрямо модел на Бернули

	Полиномен модел	Модел на Бернули
Модел на събитията	Генериране на характеристики	Генериране на документи
Случайни величини	$X =$ тако t се намира в дадената позиция	$U_t = 1$ ако t е наличен в документа
Извадка от документи	$d = \square t_1, \dots, t_k, \dots, t_{nd} \square, t_k \square V$	$d = \square e_1, \dots, e_i, \dots, e_M \square, e_i \square \{0, 1\}$

Оценка на параметри	$(X = t c)$	$(U_i = e c)$
Правило за решение: максимизиране	$\odot \prod_{1 \leq k \leq n_d} P(X_k = t_k c)$	$\odot \prod_{i=1}^n v(U_i = e_i c)$
Многократни наблюдения	Пресмята се	Игнорира се
Дължина на документите	Може да работи с по-дълги документи	Работи най-добре с къси документи
Номер на характеристиката	Може да работи с много	Най-добре работи с няколко
Оценка за термина “the”	$(X = \text{the} c) \approx 0.05$	$(U_{\text{the}} = 1 c) \approx 1.0$

В обобщение, генерира се документ в полиномния модел (Фигура 13.4) и се поставя в клас $C = c$ с $P(c)$, където C е случаена променлива, приемаща стойности от Ω . На следваща стъпка се генерира термин t_k в позиция k с $P(X_k = t_k | c)$ за всяка n_d позиция в документа. Цялото X_k има еднакво разпределение на термините за даден c . В примера на Фигура 13.4 е представено генериране на $t_1, t_2, t_3, t_4, t_5 = \text{Beijing and Taipei join the WTO}$ съответстващо на документа от едно изречение: “Beijing and Taipei join the WTO”.

За пълното уточняване на модела, генериращ документи, трябва също така, да се дефинира разпределение $P(n_d | c)$ по дълчината на документите. Без това полиномният модел е модел, генериращ характеристики, а не модел генериращ документи.

При генерирането на документ чрез модела на Бернули (Фигура 13.5), първо се избира клас $C = c$ с $P(c)$, след това се генерира бинарен индикатор e_i за всеки термин t_i от речника ($1 \leq i \leq M$). В примера на Фигура 13.5 е показана генерацията $e_1, e_2, e_3, e_4, e_5, e_6 = 0, 1, 0, 1, 1, 1$ съответстваща на документа от едно изречение “Beijing and Taipei join the WTO”, където думите “and” и “the” са приети за стоп-думи.

В таблица 13.3 се сравняват двата модела, като са включени оценявящите уравнения и правилата за вземане на решения.

Наивния Бейс, наречен така заради допусканията за независимост, е наистина много опростен за модел от естествен език. Според допускането за условна независимост характеристиките са независими от класа. Това почти никога не е вярно за термините в документите. В много случаи обратното е вярно. Двойките “хонг” и “конг” или “лондон” и “англия” от Фигура 13.7 са примери за зависими термини. Освен това, полиномния модел допуска и позиционна независимост. Моделът на Бернули игнорира позициите в документите, той се интересува само от присъствие или отсъствие. Моделът “горба с думи” отхвърля всичката информация за подреждането на думите в изреченията от естествения език. Как може НБ да е добър текстов класификатор, след като неговия модел за естествен език е толкова опростен?

Таблица 13.4 Върната оценка означава точно предвиждане, но точното предвиждане не включва вярна оценка.

	c_1	c_2	Избор на клас
Истинска вероятност $P(c d)$	0.6	0.4	c_1
$\odot \prod_{1 \leq k \leq n_d} P(t_k c)$ (Уравнение(13.13))	0.00099	0.00001	

НБ оценка($c d$)	0.99	0.01	c_1
--------------------	------	------	-------

Отговорът е – въпреки, че вероятностните оценки на НБ са от ниско качество, неговите класификационни решения са неочеквано добри. Да разгледаме документа d с вероятности $P(c_1|d) = 0.6$ и $P(c_2|d) = 0.4$, както е показано на Таблица 13.4. Да приемем, че d съдържа много термини, които са позитивни индикатори за c_1 и много термини, които са негативни индикатори за c_2 . Следователно, когато се прилага полиномния модел в Уравнение (13.13), $(c_1) \prod_{1 \leq k \leq nd} (t_k | c_1)$ ще е много по-голяма от $(c_2) \prod_{1 \leq k \leq nd} (t_k | c_2)$ (0.00099 срещу 0.00001 в таблицата). След деленето на 0.001 се получават добре формирани вероятности за $P(c|d)$, които завършват с една оценка, близка до 1.0 и една близка до 0.0. Това е общо: Печелившият клас при НБ класификацията, обикновено, има много по-голяма вероятност от останалите класове и оценките на вероятностите значимо се различават. Но класификационното решение се базира на класа, получил най-висок резултат. Без значение, колко точно са направени оценките. Въпреки лошите оценки, НБ оценява по-висока вероятност за c_1 и следователно поставя d във верния клас в Таблица 13.4. Вярното оценяване включва точно предсказване, но точното предсказване не включва вярната оценка. НБ класификаторите оценяват лошо, но, често, класифицират добре.

Дори това да не е метода с най-висока точност за текст, НБ има много качества, които го превръщат в сериозен съперник за текстов класификатор. Той изпъква, ако има много еднакво важни характеристики, които съвместно повлияват класификационното решение. Той, също така е устойчив на “шум” (разглежда се в следващата секция) и постепенна промяна в понятията (conceptdrift) – постепенна промяна с времето на някое понятие, като “Президент на САЩ” от Бил Клинтън към Джордж Буш (вж. Секция 13.7). Класификатори като kNN (Секция 14.3) трябва внимателно да се настройват към характеристиките свойства през определен период от време. Малка промяна в документа във времето ги поврежда.

Таблица 13.5 Множество от документ, при които допускането за независимост създава проблем

- (1) Той се премести от Лондон, Онтарио, в Лондон, Англия.
- (2) Той се премести от Лондон, Англия, в Лондон, Онтарио.
- (3) Той се премести от Англия в Лондон, Онтарио.

Моделът на Бернули е особено устойчив на постепенната промяна в понятията. На Фигура 13.8 се вижда, че той има прилично представяне, когато използва няколко от дузината термини. Най-важните индикатори за класа, най-малко се променят. Следователно, модел, който се базира само на тези характеристики е по-подходящ за поддържане на основното ниво на точност.

Главната сила на НБ модела е в неговата ефективност: Обучение и класификация могат да бъдат извършени само с едно обхождане на данните. Той комбинира ефективността с добрата точност, поради което често се използва за основа при разработване на текстова класификация. Често той е избиран като метод, ако (i) получаването на няколко допълнителни проценти точност не си заслужават проблемите при приложение на текстовата класификация, (ii) на разположение е огромно количество обучаващи данни и обучението с много данни е по-изгодно отколкото да се използва по-добър класификатор с по-малко обучаващо множество, (iii) ако устойчивостта спрямо *conceptdrift* е важна.

В този учебник НБ се разглежда като класификатор за текст. Допускането за независимост не е в сила за текст. Обаче, може да се покаже, че НБ е оптимален класификатор (в смисъл, че дава минимална грешка за нови данни) за данни, където допускането за независимост е в сила.

13. Класификация на Текстове и Наивен Бейс

13.4.1 Разновидност на мултиномиалния модел

Една алтернативна формализация на мултиномиалния модел представя всеки документ \mathbf{d} като M -измерен вектор $\langle tf_{t_1,d}, \dots, tf_{t_M,d} \rangle$, където $tf_{t_i,d}$ е честотата, с която t_i се появява в документа \mathbf{d} . Пресмятаме $P(\mathbf{d} | c)$ по следния начин :

$$(13.15) P(d | c) = P(\langle tf_{t_1,d}, \dots, tf_{t_M,d} \rangle | c) \sim \prod_{1 \leq i \leq M} P(X = t_i | c)^{tf_{t_i,d}}$$

Това уравнение е еквивалентно на *последователния модел* от уравнение (13.2) като вземем $P(X = t_i | c)^{tf_{t_i,d}} = 1$ за термините, които не се появяват в $d(tf_{t_i,d} = 0)$, а тези, които се появяват $tf_{t_i,d} \geq 1$ пъти, ще донесът $tf_{t_i,d}$ множителя и в двете уравнения.

13.5 Подбор на Характеристиките

Подборът на Характеристиките е процес, при който се избира подмножество от термини, появяващи се в тренировъчното множество и използването му за класификация на текста. Този процес има две основни цели. Първата, да направи тренирането и употребата на класификатор по-ефективна, намалявайки големината на използвания речник. Това е от особено значение за класификатори, които, за разлика от Наивния Бейс, са скъпи за трениране. Втората цел е да повиши класификационната точност като елиминира шума или по-точно *шумните характеристики* в текста. Шумни са онези характеристики, които прибавени към представянето на документ, увеличават класификационната грешка върху нови данни. Например да вземем рядко срещан термин като **арахноцентричен**, който не носи никаква информация за клас *Китай*, но всичките му срещания в тренировъчното множество са в документ от този клас. Тогава обучителния метод може да създаде класификатор, който погрешно слага всички документи, в които се среща този термин, в клас *Китай*. Неточност от този вид се нарича *overfitting*.

Ще разгледаме алгоритъма на простиия подбор на характеристиките. За даден клас c пресмятаме $A(t,c)$ – мярка за полезност, за всеки термин от речника и избираме k от тях, които са с най-високи стойности на $A(t,c)$. Всички останали термини се изключват и не се използват за класификация. Ще въведем три различни мярки за полезност : обща информация, $A(t,c) = I(U;C_c)$; χ^2 -тест, $A(t,c) = X^2(t,c)$; честота, $A(t,c) = N(t,c)$.

В тази секция ще представим подбор на характеристиките за двукласова класификация като *Китай* и *не-Китай*.

13.5.1 Обща информация (ОИ)

ОИ измерва колко информация, присъствието/отсъствието на даден термин, дава за правилната му класификация към клас c . Формално:

$$(13.16) I(U;C) =$$

$$\sum_{e_t \in \{0,1\}} \sum_{e_c \in \{0,1\}} P(U = e_t, C = e_c) \log_2 [(P(U = e_t, C = e_c)) / (P(U = e_t)P(C = e_c))],$$

където \mathbf{U} е случайна променлива, която приема стойности $e_t = 1$ (документа съдържа термин t) и $e_t = 0$ (документа не съдържа t) и \mathbf{C} е случайна променлива, приемаща аналогични стойности e_c в зависимост дали документът е от клас c или не. Ако оценим вероятностите с принципа на максимално правдоподобие, то уравнения (13.16) и (13.17) са еквивалентни:

$$(13.17) I(U;C) = (N_{11}/N) \log_2 [N N_{11} / (N_1 N_{.1})] + (N_{01}/N) \log_2 [N N_{01} / (N_0 N_{.1})] + \\ (N_{10}/N) \log_2 [N N_{10} / (N_1 N_{.0})] + (N_{00}/N) \log_2 [N N_{00} / (N_0 N_{.0})],$$

където \mathbf{N} -те са броя на документите, за които съответните стойности на e_t и e_c са записани с долни индекси. Например, N_{10} е броят на документите, които съдържат $t(e_t = 1)$ и не са в $c(e_c = 0)$. $N_{.1} = N_{10} + N_{11}$ е броят на документите, които съдържат $t(e_t = 1)$ и броим документите независимо от това към кой клас принадлежат. Очевидно $\mathbf{N} = N_{00} + N_{01} + N_{10} + N_{11}$ е броя на всички документи.

ОИ измерва колко информация – в смисъла на теория на информацията – даден термин носи за класа. Ако разпределението на термина е сырото в класа каквото е и в колекцията като цяло, то $I(U;C) = 0$. ОИ достига своя максимум ако терминът е перфектен индикатор за принадлежност към класа, т.e. терминът се съдържа в документа \mathbf{D} документа е в дадения клас. Както може да се очаква запазването на носещите информация термини и премахването на тези, които не носят такава, води до намаляване на шума и подобряване точността на класификатора.

13.5.2 χ^2 Подбор на Характеристиките

Друг популяррен подбор на характеристиките е χ^2 . В статистиката, χ^2 -тестът се прилага за тестване независимост на две събития, където две събития A и B по дефиниция са *независими* ако $P(AB)=P(A)P(B)$ или еквивалентното – $P(A|B) = P(A)$ и $P(B|A) = P(B)$. При този подбор на характеристиките свете събития, които разглеждаме са : появяване на термин и появяване на клас. В този случай класираме термините като пресметнет:

$$(13.18) X^2(D,t,c) = \sum e_t \epsilon \{0,1\} \sum e_c \epsilon \{0,1\} [(N e_t e_c - E e_t e_c)^2 / E e_t e_c],$$

където e_t и e_c са дефинирани както във уравнение (13.16). \mathbf{N} е *наблюдаваната* честота в \mathbf{D} , а пък E е *очекваната* такава. Например, E_{11} е очакваната честота на t и c да се появят заедно в документ, като допускаме, че терминът и класът са независими.

χ^2 е мярка за това колко се различават очакваната бройка E и наблюдаваната такава \mathbf{N} . Голяма стойност на χ^2 говори за вярност на хипотезата за независимост, която казва, че очакването и наблюденето са близки. Ако двете събития са зависими, то появата на термина в документ би означавало по-вероятна (или по-малко вероятна) появява на класа. Това означава, че този термин би трябвало да се вземе в предвид като характеристика за дадения клас. Това въщност е смисълът на χ^2 подбора на характеристиките.

Друг начин за пресмятане на X^2 е:

$$(13.19) X^2(D,t,c) = \\ [(N_{11} + N_{10} + N_{01} + N_{00}) x (N_{11} N_{00} N_{10} N_{01})^2] / [(N_{11} + N_{01}) x (N_{11} + N_{10}) x (N_{10} + N_{00}) x (N_{01} + N_{00})]$$

Това уравнение е еквивалентно на (13.18).

Оценка на χ^2 като метод за подбор на характеристиките

От статистическа гледна точка този метод е проблематичен. За тест с една степен на свобода трябва да се използва т.н. корекция на Yates, която намалява на статистическата важност. Също така, когато даден статистически тест се използва многократно, то вероятността за грешка расте. В текстовата класификация, обаче, рядко има голямо значение дали няколко термина са добавени или премахнати към характеристиките. Важно е само доколко тези термини се отнасят до класовете и документите. Т.к. χ^2 подбора на характеристиките калсира само характеристики по тяхната полезност или независимост, то няма нужда да се тревожим, че не се придържа строго към статистическата теория.

13.5.3 Честотно-ориентиран Подбор на Характеристиките

Третият вид метод, избира термините, които най-често се срещат в дадения клас. Честотата може да бъде дефинирана както като честота на документите (брой документи в класа c , които съдържат термина t) така и като честота на колекция (брой символи на t , които се срещат в документи от c). Документната честота е по-подходяща за модела на Бернули, а пък колекционната такава- за мултиномиалния метод.

Честотно-ориентириания подбор на характеристиките избира няколко често срещани термина, които не носят конкретна информация за класа. Когато хиляди такива характеристики са събрани, този метод се справя добре. В случаите, когато не се търси оптimalна точност, този метод е добра алтернатива на по-комплицираните методи.

13.5.4 Подбор на Характеристиките за мулти-классификатори

За операционна система с голям брой класификатори е желателно да се избере само едно множество от характеристики вместо да има отделно такова за всеки класификатор. Един начин да се направи това е да се пресметне χ^2 статистиката за $n \times 2$ таблица, където колоните са появяванията и непоявяванията на термина а всеки ред отговаря на един от класовете. След това можем да изберем k термина с най-висока χ^2 статистика като вече направихме.

В повечето случаи, обаче, тези статистики се пресмятат първо по отделно за всеки клас като се използва двукласификаторната задача за c и \hat{c} и след това се комбинират. Един начин за комбиниране е да пресметне един показател на качеството за всяка характеристика, например като се осреднят стойностите $A(t,c)$ за характеристиката t , и след това да се изберат k характеристики с най-висок показател на качеството. Друг често използван метод за комбиниране е да се изберат най-добрите k/n характеристики, за всеки от n класификатори и да се комбинират тези n множества в едно голямо характеристично множество.

Точността на класификацията намалява когато изберем k общи характеристики за система от n класификатори за разлика от n различни множества с големина k .

13.5.5 Сравнение на Методите за Подбор на Характеристиките

ОИ и χ^2 са доста различни методи. Независимостта на термина t и класа c може понякога да бъде отхвърлена с голяма степен на сигурност дори и ако t носи малко информация за принадлежността на документ към c . Това е вярно за доста малко термини. Поради критерия за значимост, χ^2 избира по-рядко срещани термини от ОИ. Също така критериите на ОИ не винаги избират термините, които максимизират класификационната прецизност.

Въпреки разликите между двата метода, класификационната точност на характеристичните множества, избрани от двата метода, не се различават систематично. В повечето задачи за текстова класификация, има малко силни индикатори и много слаби. Стига всички силни индикатори и голяма част от слабите да са избрани, точността би трябвало да е добра. И двата метода правят това.

И трите метода, които разглеждахме, са *алини* методи. Те могат да изберат характеристики, които не носят допълнителна информация в сравнение с вече избрани характеристики. Въпреки, че това може да повлияе на точността, неалчните методи се използват рядко поради тяхната изчислителна сложност.

13.6 Оценка на текстовата класификация

Ще използваме *ефективност* като общ термин за мярка която оценява качеството на класификационните решения, включвайки прецизност, отзоваване, F_1 , и точност. В тази книга терминът *представяне* се отнася до изчислителната ефективност на класифицирането и системите за ИИ.

Когато обработваме дадена колекция с няколко двукласови класификатора, често искаме да изчислим една обобщена мярка, която комбинира мярките за отделните класификатори. Има два метода за тази цел. Първият – *макроосредняване* – изчислява средното за всички класове. Вторият – *микроосредняване* – взима решение за всеки документ поотделно като обхожда всеки клас и след това изчислява ефективна мярка въз основа на извлечената информация.

Разликата между двата метода може да бъде голяма. Макроосредняването дава еднакви тегла на всеки клас, докато микроосредняването дава равни тегла на всяко класификационно решение взето за-документ. Микроосредняването дава реални резултати за ефективност върху големи класове в тестовата колекция. За малки класове трябва да се използва макроосредняването.

Резултатите от направени изследвания показват, че средната ефективност на НБ е неконкурентноспособна с класификатори като SVMs когато се тренират и тестват върху *независими и идентични по разпределение* данни. Тези разлики, обаче, могат да останат невидими или дори да се обърнат, при реална работа, когато тренировъчния пример се взима от подмножество от данните, върху които класификаторът ще бъде използван. Класирането на класификатори зависи от класовете, колекцията от документи и експерименталната ситуация.

Когато оценяваме различни методи, много важно е да поддържаме стриктно разделение между тестовото и тренировъчното множество. Лесно бихме могли да вземем правилно класификационно решение върху тестовото множество като използваме информация събрана от това множество, като например факта, че даден термин е добра характеристика за тестовото множество, въпреки, че не е такава в тренировъчното. Като правило, точността върху нови данни, ще бъде по-малка отколкото тази върху тестовото множество.

Може да се използва *работно множество*, за тестване в процеса на установяване и развиване на вашия метод. Такова множество може да бъде използвано при намиране на добра стойност за параметрите на метода, който развивате, като например броя на характеристиките, които да вземете в предвид. След установяване стойностите на параметрите може да се премине към пускане на метода върху тестовото множество и побликуване на резултатите.

14 Класификация с векторно пространство

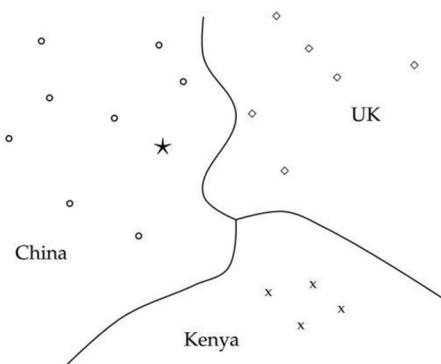
При наивния Бейсов метод документът се представя като двоичен терм или като бинарен вектор (e_1, \dots, e_n). В тази глава ще приемем различно представяне на текстовете – моделът на векторно пространство, представен в глава 6. Според този модел всеки документ е представя като вектор с една реална компонента, обикновено tf-idf тегло, за всеки терм. По този начин пространството от документи X , което представлява дефиниционното множество на класификационната функция γ е $R|V|$. В тази глава се представят няколко метода за класификация, които оперират върху вектори с реални стойности.

Основната хипотеза, при използването на моделът на класификация с векторнопространство, е хипотезата за съседство.

14.1 Хипотеза за съседство:

Документите в един и същи клас образуват област на съседство и областите на различните класове не се застъпват.

За пример да разгледаме една задача за класификация, при която класовете се разграничават от шаблони от думи. Нека разделяме документите в класове според държавите, за които се отнасят. Документите в клас Китай ще съдържат много думи като китайски, Пекин и Мао, а тези от клас Обединеното кралство - Лондон, Великобритания и кралица. Документите от двата класа формират две различни области на съседство. Темата на тази глава е как да начертаем граници, които ги разделят, и да класифицираме нови документи.



► Figure 14.1 Vector space classification into three classes.

Фиг.14.1

Дали дадено множество от документи сформира област на съседство много зависи от начина, по който сме избрали да представим самите документи: типа на претегляне, списъка от стоп думи и т.н. Да вземем за пример класовете документи, написани от група хора и написани от едно лице. Честата поява на личното местоимение в първо лице единствено число – аз – свидетелства за документ от класа документи, написани от едно лице. Но тази информация ще бъде изтрита, ако използваме списък от стоп думи. Ако представянето на отделните документи

е неподходящо, хипотезата за съседство няма да е в сила и класификацията чрез векторно пространство става невъзможна.

Препоръчително е също да използваме претеглено и нормализирано според дължината tf-idf представяне. Например терм с 5 срещания в документа трябва да има по-голямо тегло, но 5 пъти по-голямо тегло ще акцентира твърде много на този терм.

В тази глава ще представим два метода за класифициране чрез векторно пространство: на Rocchio и kNN. При класификацията на Rocchio векторното пространство се разделя на области(или прототипи), центрирани около центрове на тежестта, изчислени за всеки клас. Класификацията на Rocchio е простира и резултатна, но неточна, когато класовете не са много близки до сфери с еднакви радиуси.

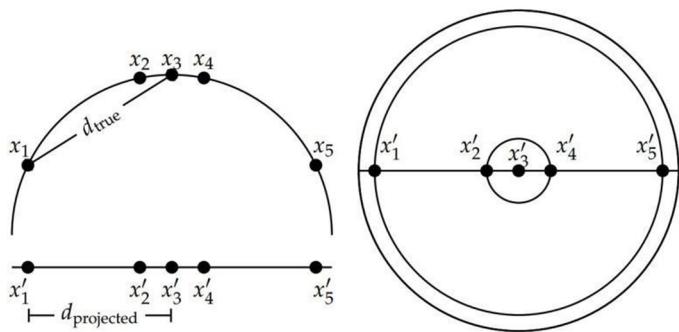
При kNN документът се слага в този клас, от който има най-много на брой съседи от k-те си най-близки съседи. При kNN не е изрично нужно обучение, може направо да се използва необработваното обучаващо множество от документи. При достатъчно голямо обучаващо множество kNN може да се използва за несферични и други сложни класове по-успешно от метода на Rocchio.

Много алгоритми за класификация на текстове могат да се разглеждат като линейни. Заради размяната отклонение-разлика нелинейните модели не винаги са по-добри от линейните. Нелинейните модели трябва да се справят с повече параметри при ограничени обучаващи данни, поради това е по-вероятно те да допуснат грешка за малки и нехомогенни множества от данни.

Когато се използват класификатори за два класа за проблеми с повече от два класа, има две основни задачи: един-от - документът се поставя в точно един от няколко взаимно изключващи се класове, и някои-от - документът може да се постави в няколко класа. С класификатори за два класа се решават някои-от проблеми, а с комбинация от няколко такива класификатора - един-от проблеми.

14.2 Представяния на документи и мерки за свързаност във векторни пространства

Както вече отбелаяхме ще представяме документите като вектори в $R^{|V|}$. За да илюстрираме свойствата на векторите, представящи документи, ще изобразяваме тези вектори като точки в равнина. Всъщност векторите, представящи документи, са единични вектори с нормализирана дължина, които представляват точки на повърхнината на хиперсфера. Представянето на векторите като точки от равнина се получава като се проектират съответните точки от сферата(двумерен пример на фиг. 14.2). Разстоянията на повърхнината на сферата и в проективната равнина са приблизително еднакви, когато се ограничим върху малка област от повърхнината и изберем подходяща проекция.



Фиг. 14.2

Решенията при класифицирането чрез използване на векторно пространство се взимат въз основа на разстоянието. В тази глава като мярка за разстояние ще използваме Евклидовото разстояние.

Много важна роля в класифицирането чрез векторно пространство играят центровете на тежестта. Те не са с нормализирана дължина. За вектори, които не са нормализирани, скаларното произведение, тъговото и Евклидовото разстояние имат различно поведение като цяло. Затова когато изчисляваме сходството между документ и център на тежестта, ще се концентрираме главно върху малки локални области. Колкото по-малка е областта, толкова по-близко е поведението на тези мерки.

14.3 Класифициране по Rocchio

На фиг.14.1 са показани класовете Китай, САЩ и Кения в двумерното пространство. Документите са представени като кръгове, ромбове и букви X. Границите на фигурата, които наричаме граници на решението, са избрани така, че да разделят трите класа, но от друга страна са произволни. За да класифицираме нов документ, изобразен като звезда, определяме областта, в която се намира, и приписваме на документа класа на тази област - в нашия пример Китай. Задачата при класифициране чрез векторно пространство е да създадем алгоритъма за възможно най-точно определяне на границите.

Класификацията на Rocchio е една от най-добрите при определянето на границите на класовете. При нея за това определяне се използват центрове на тежестта. Центърът на тежестта на клас с се изчислява като средно аритметично на векторите по формулата

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

където D_c е множеството документи от клас с: $D_c = \{d : \langle d, c \rangle \in D\}$. $\vec{v}(d)$ е нормализираният вектор d .

Границата между два класа при класификацията на Rocchio е множеството от точки, които са на равно разстояние от двата центъра на тежестта. Това са $|a_1| = |a_2|$, $|b_1| = |b_2|$, и $|c_1| = |c_2|$ на фигурата. Това множество от точки винаги е права. Обобщението на права в M-мерно

пространство е хиперравнина, която дефинираме като множеството от точки \vec{x} удоволствоящи равенството:

$$\vec{w}^T \vec{x} = b,$$

където \vec{w} е M-мерният нормален вектор от хиперравнината, а b е константа.

Правилото за класифициране е: точка се поставя в клас в зависимост от това в коя област попада. Казано по друг начин: определяме центъра на тежестта $\vec{\mu}_C$, до който точката е най-близо, и добавяме съответния документ към този клас с. В нашия пример добавяме звездата към областта Китай, защото се намира най-близо до центъра на тежестта на тази област.

Псевдокод на алгоритъма:

Обучаване при ROCCHIO(C,D)

```

1 for each  $c_j \in C$ 
2 do  $D_j \leftarrow \{d : \langle d, c_j \rangle \in D\}$ 
3  $\vec{\mu}_j \leftarrow \frac{1}{|D_j|} \sum_{d \in D_j} \vec{v}(d)$ 
4 return  $\{\vec{\mu}_1, \dots, \vec{\mu}_J\}$ 
```

Прилагане на ROCCHIO($\{\vec{\mu}_1, \dots, \vec{\mu}_J\}$, d)

```
1 return  $\arg \min_j |\vec{\mu}_j - \vec{v}(d)|$ 
```

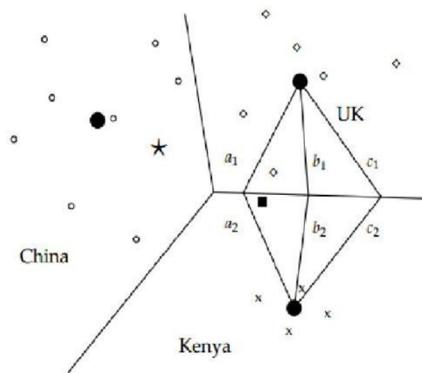
Използваната мярка е Евклидово разстояние. Нейна алтернатива е тъгловото разстояние:

Присъедини d към клас $c = \arg \max_c \cos(\vec{\mu}(c'), \vec{v}(d))$

Както вече споменахме двете мерки понякога водят до различни решения за класифициране.

Класификацията на Rocchio е форма на алгоритъма Rocchio relevance feedback.

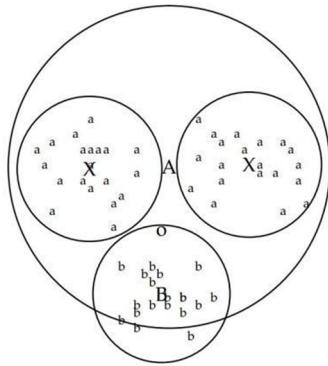
За да се определи правилно съседството, класовете при класификацията на Rocchio трябва да образуват области, доближаващи се до сфери с еднакви радиуси. На фигура 14.3 тъмният квадрат точно под границата на Обединеното Кралство и Кения по-добре пасва на класа Обединено кралство, защото неговите елементи са разположени по-нарядко от тези на Кения. Но Rocchio го поставя в Кения, защото игнорира детайлите по разпределението на точките в клас, а използва само разстоянието до центъра на тежестта.



► Figure 14.3 Rocchio classification.

Фиг. 14.3

Условието за сферичност не е спазено и на фигура 14.5. Не можем да представим клас „a“ добре с един прототип, защото се състои от две групи. Алгоритъмът на Rocchio често класифицира грешно такъв тип мултимодални (състоящи се от много сфери) класове. Пример за мултимодалност са текстове за държавата Бирма, чието име е сменено на Мианмар през 1989. Двете групи с текстове, преди и след смяната на името, трябва да са близки една до друга в пространството.



Фиг. 14.5

Класифицирането на два класа е друг случай, при който класовете рядко могат да се представят като сфери с еднакви радиуси. Повечето алгоритми за разделяне на два класа разграничават клас като Китай, заемащ малка област от пространството, и неговото голямо допълнение. Ако приемем, че радиусите са приблизително равни, много текстове, които не се отнасят за Китай, ще попаднат в неговия клас. Затова, повечето класификатори за два класа използват модифицирано правило за вземане на решения:

Прибави d към клас с тогава и само тогава, когато $|\vec{\mu}(c) - \vec{v}(d)| < |\vec{\mu}(c) - \vec{v}(d)| - b$,

където b е положителна константа. Често допълнението не се използва въобще и затова правилото се опростява до:

$|\vec{\mu}(c) - \vec{v}(d)| < b'$,

където b' е положителна константа.

В таблицата е попълнена времевата сложност на класифицирането на Rocchio:

Режим	Времева сложност
обучаване	$\Theta(D L_{ave} + C V)$
тестване	$\Theta(L_a + C M_a) = \Theta(C M_a)$

Където L_{ave} е средния брой елементи (tokens) на документ; L_a и M_a – броя елементи (tokens), съответно типове съответно в тествания документ. Изчисляването на Евклидовото разстояние между центъра на тежестта на класа и документ е $\Theta(|C|M_a)$. Добавянето на всички документи към съответните им центрове на тежестта е $\Theta(|D|L_{ave})$. Разделянето на всяка сума от вектори на размера на класа ѝ за пресмятането на центъра на тежестта е $\Theta(|V|)$.

14.4 k -ти най-близък съсед (kNN)

За разлика от алгоритъма на Rocchio, при класифицирането по k-ти най-близък съсед, границата на решението се избира локално. При 1NN слагаме всеки документ в класа на най-близкия му съсед. При kNN добавяме документа към класа, в който той има най-голям брой от k-те си най-близки съседи. Логиката на kNN класификацията е, че основавайки се на хипотезата за съседство, очакваме проверяваният документ d да има същия етикет като обучителните документи, разположени в локалната област около d.

Границите на решение при 1NN са слепени сегменти от Вороноеva мозайка. Вороноеvата мозайка на множество от обекти разделя пространството на Воронееви клетки, където клетката на всеки обект се състои от всички точки, които са по-близо до обекта отколкото до другите обекти. В нашия случай обектите са документи. Вороноеvата решетка разделя равнината на $|D|$ затворени многоъгълници, всеки съдържащ съответния си документ.

При всяко друго естествено $k \geq 2$ пространството отново се раделя на затворени многоъгълници, такива че за всеки от тях областта на k-те най-близки съседи е една.

Класификацията 1NN не е много точна, защото решението при класифициране на документ зависи само от един обучаващ документ, който може да се окаже грешно разпределен или нетипичен. kNN за $k > 1$ е по-точен.

Псевдокод на алгоритъма kNN:

TRAIN-KNN(C,D)

```
1 D' ← PREPROCESS(D)
2 k ← SELECT-K(C,D')
3 return D', k
APPLY-KNN(C,D', k, d)
1 Sk ← COMPUTENEARESTNEIGHBORS(D', k, d)
2 foreach cj ∈ C
3 do pj ← |Sk ∩ cj|/k
4 return arg maxj pj
```

Има вероятностна версия на kNN класификация алгоритъм. Може да оценяваме вероятността даден документ да е член на клас с като пропорция от k-те най-близки съседи в с. Например за $k=3$, вероятностната оценка за звездата да е от клас кръг е $P^{\wedge}(\text{circle class} \mid \text{star}) = 1/3$, да е от клас X е $P^{\wedge}(\text{X class} \mid \text{star}) = 2/3$ и – от клас ромб $P^{\wedge}(\text{diamond class} \mid \text{star}) = 0$.

Параметърът k често се избира въз основа на опита или познанията за проблема. Желателно е k да е нечетно число. Често се избира $k = 3$ и $k = 5$, но и по-големи стойности между 50 и 100 също се използват. Друг начин е да се избере k, което дава най-добри резултати при класифициране на част от обучаващото множество.

Можем също да изберем k-те най-близки съседи според тяхното ъглово разстояние. При този начин резултатът на всеки клас се изчислява по формулата:

$$score(c, d) = \sum_{d' \in Sk(d)} Ic(d') \cos(\sim\vec{v}(d'), \vec{v}(d)),$$

където $Sk(d)$ е множеството от d' – най-близки съседи, $Ic(d') = 1$, ако d' е от клас c и 0 в противен случай. Присъединяваме документа към класа с най-висок резултат.

14.4.1 Времева сложност и оптимизация на kNN

В таблицата е дадена времевата сложност на алгоритъма:

С предварителна обработка на обучаващото множество:

обучение	$\Theta(D La)$
проверка	$\Theta(La + D MaveMa) = \Theta(D MaveMa)$

Без предварителна обработка на обучаващото множество:

обучение	$\Theta(1)$
проверка	$\Theta(La + D MaveMa) = \Theta(D MaveMa)$

Mave е средният размер на речниковия състав на документите в колекцията.

Обучението при kNN се състои от определяне на параметъра k и предварителна обработка на документите. Можем предварително да изберем k и да не обработваме документите. В такъв случай алгоритъмът не изисква никакво обучение. Но на практика в повечето случаи обучението е необходимо.

Времето за проверка зависи линейно от мощността на обучаващото множество, защото при нея изчисляваме разстоянието от всеки обучаващ документ до проверявания документ. Времето за проверка не зависи от броя на класовете J . Затова е подходящо kNN да се използва за проблеми с голямо J .

kNN се нарича още учене по памет или учене от пример, защото не се прави никаква оценка на параметри като при Rocchio (центрове на тежестта), а просто се запомнят всички примери от обучаващото множество и документът се сравнява с тях.

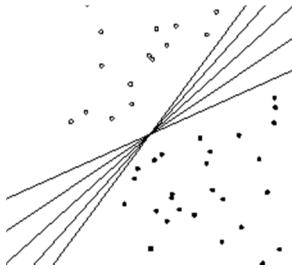
Има бързи kNN алгоритми за малки размерности на M . Също така има бързи приближаващи алгоритми за големи M , но те често допускат много грешки.

Асимптотично нивото на допускане на грешки на kNN е по-малко от $2x$, където x е оптималното ниво на допускане на грешки. Това е така заради влиянието на шума. Шумът влияе на два компонента на kNN: проверявания документ и най-близкия обучаващ документ.

14.5 Линейни и нелинейни класификатори

Линеен класификатор е класификатор за два класа, който определя принадлежността към даден клас чрез сравняване на линейна комбинация на характеристиките спрямо даден праг.

В двумерното пространство, линейният класификатор представлява линия.



Фиг. 14.8 Съществуват безброй много хиперравнини, разделящи два линейноразделими класа.

На фиг. 14.8 са показани пет примера. Тези линии могат да се изразят чрез формулата $w_1x_1 + w_2x_2 = b$. Правилото за класификация на линеен класификатор разпределя даден документ в клас c , ако $w_1x_1 + w_2x_2 > b$ и в клас \underline{c} , ако $w_1x_1 + w_2x_2 \leq b$. $(x_1, x_2)^T$ е двумерното векторно представяне на документа, а $(w_1, w_2)^T$ е векторът, който определя (заедно с b) границата на решение.

Този двумерен класификатор може да се обобщи и за многомерно пространство като се определи хиперравнина, такава че $\vec{W}^T \vec{x} = b$. В този случай критерият за класификация е следният:

- Ако $\vec{W}^T \vec{x} > b$, документът се отнася към класа c .
- Ако $\vec{W}^T \vec{x} \leq b$, документът се отнася към класа \underline{c} .

Хиперравнина, която използваме като линеен класификатор, наричаме хиперравнина на решение (a decision hyperplane).

Примери за линейни класификатори са Rocchio и наивният Бейсов класификатор (Naive Bayes). За да покажем, че Rocchio е линеен класификатор, ще разгледаме вектора \vec{x} . Той принадлежи на границата на решение, ако е на равни разстояния от центровете на тежестта на двата класа:

$$|\vec{\mu}(c_1) - \vec{x}| = |\vec{\mu}(c_2) - \vec{x}|$$

Това уравнение съответства на линеен класификатор с нормален вектор

$$\vec{w} = \vec{\mu}(c_1) - \vec{\mu}(c_2) \text{ и } b = 0.5 * (|\vec{\mu}(c_1)|^2 - |\vec{\mu}(c_2)|^2).$$

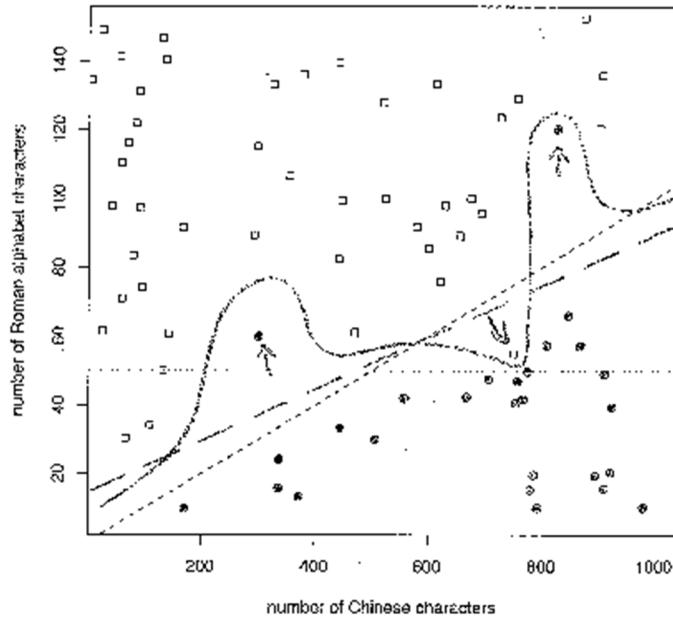
За да покажем, че наивният Бейсов класификатор също е линеен, избираме този клас c , за който $\hat{P}(c|d)$ има най-голяма стойност. $\hat{P}(c|d) \propto \hat{P} \odot \prod_{1 \leq k \leq n_d} \hat{P}(t_k|c)$, а n_d е броят обекти в документа, които са част от лексиката. Да означим допълнението на c с c^c . Тогава за логаритъма получаваме:

$$\log \frac{\hat{P}(c|d)}{\hat{P}(c^c|d)} = \log \frac{\hat{P}(c)}{\hat{P}(c^c)} + \sum_{1 \leq k \leq n_d} \log \frac{\hat{P}(t_k|c)}{\hat{P}(t_k|c^c)}.$$

Избираме клас c в случаите, в които пропорцията благоприятни към неблагоприятни случаи е по-голяма от 1 или, еквивалентно, когато логаритъма е по-голям от 0. От формулата $\vec{W}^T \vec{x} = b$ може да се получи формулата:

$$\log \frac{\hat{P}(c|d)}{\hat{P}(c^c|d)} = \log \frac{\hat{P}(c)}{\hat{P}(c^c)} + \sum_{1 \leq k \leq n_d} \log \frac{\hat{P}(t_k|c)}{\hat{P}(t_k|c^c)},$$

ако $w_i = \log [\hat{P}(t_i|c)/\hat{P}(t_i|c^c)]$, x_i е броят срещания на t_i в d и $b = -\log [\hat{P}(c)/\hat{P}(c^c)]$. Индексът i , $1 \leq i \leq M$, се отнася за термове от лексиката (а не за позиции в d , както k) и \vec{x} и \vec{w} са M -мерни вектори. Доказвахме, че наивният Бейсов класификатор е линеен класификатор.



Фиг. 14.10 Линеен проблем с шум. В този хипотетичен сценарий с класификация на учеб страници, страниците, в които има само китайски символи са обозначени със запълнени кръгове, а страниците, в които има и китайски и латински символи – с квадрати. Двата класа са разделени с линейна граница на класове (линията с по-късите чертички), с изключение на три документа с шум (обозначени със стрелки).

Фигура 14.10 е графичен пример за линеен проблем. Разпределенията на двата класа, $P(d|c)$ и $P(d|\underline{c})$, са разделени посредством линия. Тази разделятелна линия наричаме граница на клас. Тя е „истинската“ граница на двата класа и я различаваме от границата на решение, която обучаващият метод изчислява, за да състави приближение на границата на класа.

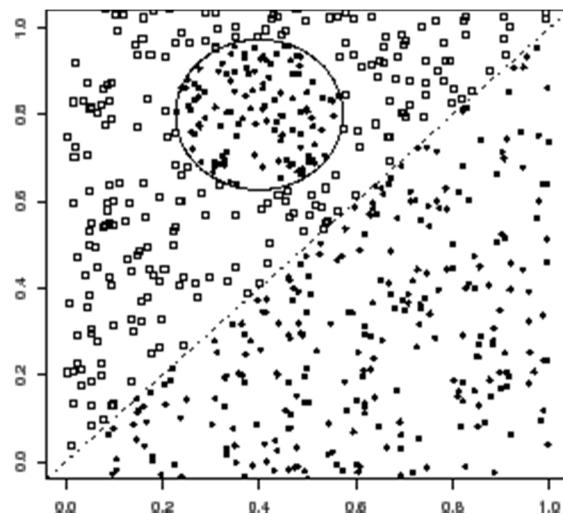
На фиг. 14.10 има документи с шум, които не се вписват добре в цялостното разпределение на класовете. Шумът е подвеждаща характеристика, която, когато е включена в представянето на документа, увеличава грешката при класификация. Аналогично, документ, съдържащ шум, е документ, който, когато е включен в обучаващо множество, подвежда обучаващия метод и увеличава грешката при класификация. Интуитивно, представителното пространство е разделено на зони с предимно хомогенни класови назначения. Документ, който не следва доминантния за своята зона клас, се нарича документ с шум.

Документите с шум са една от причините за това обучението на линеен класификатор да е толкова трудно. Ако обръщаме твърде много внимание на документите с шум, когато избираме хиперравнината на решение на класификатора, тогава класификаторът ще бъде неточен върху нови данни. Обикновено е трудно да се определи кои документи съдържат шум.

Ако съществува хиперравнина, която перфектно разделя два класа, то те се наричат линейноразделими. Ако е изпълнена линейната разделимост, тогава има безброй много линейни разделители, както е показано на фиг. 14.8, където възможните разделятелни хиперравнини са безкраен брой.

Фиг. 14.8 също така представя и друго предизвикателство при обучението на линеен класификатор. В случай, че се сблъскаме с линейна разделимост, се нуждаем от критерий, по който да изберем хиперравнина на решение измежду всички хиперравнини на решение, които перфектно разделят обучаваците данни. В общия случай някои от тези хиперравнини ще разделят добре нови данни, а други – не.

Пример за нелинеен класификатор е kNN. Границите на решение на kNN локално са линейни сегменти, но глобално имат сложна форма, която не е еквивалентна на линия в двумерното пространство или на хиперравнина в многомерното пространство.



Фиг. 14.11 Нелинеен проблем

Фиг. 14.11 е друг пример за нелинеен проблем: няма добър линеен разделител между разпределенията $P(d|c)$ и $P(d|\underline{c})$ заради зоната, заградена в кръг. Линейните класификатори класифицират погрешно тази зона. Нелинеен класификатор като kNN би бил много по-точен за този тип проблеми стига обучаващото множество да е достатъчно голямо.

Ако даден проблем е нелинеен и неговите граници на клас не могат да бъдат добре приближени до линейни хиперравнини, тогава нелинейните класификатори са често по-точни от линейните. Ако даден проблем е линеен, най-добре е да се използва по-прост линеен класификатор.

14.6 Класификация в повече от два класа

Линейните класификатори за два класа могат да бъдат разширени за $J > 2$ класа. Методът, който се изполва за целта, зависи от това дали класовете са взаимно изключващи се или не.

Класификацията на класове, които не са взаимно изключващи се, наричаме „някои-от” (any-of), „многоетиетна” (multilabel) или класификация „с много стойности” (multivalue). В този случай, даден документ може да принадлежи на няколко класа едновременно, да принадлежи на един клас, или да не принадлежи на никой от класовете. Подвеждащо е обаче да казваме, че класовете са независими един от друг, тъй като рядко те са статистически независими. За да осъществим класификация с повече от два класа, обучаваме J различни класификатора γ_i за „някои-от” класификация. Всеки от тези класификатори връща или c_i , или $\underline{c_j} : \gamma_i(d) \in \{c_i, \underline{c_j}\}$.

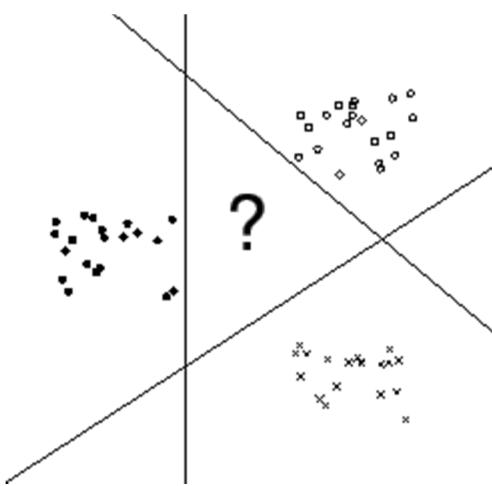
Задачите за „някои-от” класификация могат да се решат с линейни класификатори по следния начин:

1. За всеки клас се построява такъв класификатор, че обучаващото множество се състои от множеството от документи в класа (позитивни етикети) и допълнението на това множество (негативни етикети).
2. Върху тестовия документ се прилага отделно всеки един от класификаторите. Решението на един класификатор не оказва влияние върху решението на останалите класификатори.

Вторият тип класификация с повече от два класа е „един-от” класификацията. При нея класовете са взаимно изключващи се. „Един-от” класификацията се нарича още „мултиномиална” (multinomial), „разделена на части” (polytymous), „многокласова” (multiclass) или „едноетиленна” (single-label). При „един-от” класификацията съществува една класифицираща функция γ и тя се изобразява в множеството C , т.е. $\gamma(d) \in \{c_1, c_2, \dots, c_j\}$. kNN е нелинеен „един-от” класификатор.

Реалните „един-от” проблеми не са така разпространени в класификацията на текстове както „някои-от” проблемите. С класове като „Великобритания”, „Китай”, „домашни птици” и „кафе” даден документ може да съответства на много теми едновременно, като например: „Министър-председателят на Великобритания посети Китай, за да говори за търговията на кафе и домашни птици.”

За класификационния проблем, свързан с идентифицирането на езика на даден документ, „един-от” предположението е добро приближение, тъй като повечето текст е написан само на един език. В такива случаи използването на „един-от” ограничения може да увеличи ефективността на класификатора, защото така се елиминират грешките, причинени от това, че „някои-от” класификаторите причисляват даден документ или към повече от един клас, или не го причисляват към никой клас.



Фиг. 14.12 Ъ хиперравнини не разделят пространството на J взаимно независими области

Ъ хиперравнини не разделят $R^{|V|}$ на J взаимно независими области както е показано на фиг. 14.12. Затова, когато използваме линейни класификатори за два класа за „един-от” класификация, ни трябва комбиниран метод. Най-простият начин е да подредим класовете по даден признак и после да изберем този, който е първи. Геометрично подреждането може да бъде направено на базата на разстоянието до J линейни разделителя. Колкото по-близо са

документите до разделителя на даден клас, толкова по-голяма е вероятността да бъдат погрешно класифицирани. За мярка при подреждането на класове може да използваме „доверие“ (confidence), например вероятност за принадлежност към даден клас. Този алгоритъм за „един-от“ класификация с линейни класификатори можем да опишем по следния начин:

1. За всеки клас построяваме класификатор, чието обучаващо множество се състои от множеството от документи в класа (позитивни етикети) и допълнението на това множество (негативни етикети).
2. Прилагаме всеки класификатор поотделно върху тестовия документ.
3. Отнасяме документа към класа с:
 - максимален резултат
 - максимално „доверие“
 - или най-голяма вероятност.

Важен инструмент за анализиране представянето на класификатор за $J > 2$ класа е матрицата на объркване. За всяка двойка класове $\langle c_1, c_2 \rangle$ матрицата на объркване показва колко на брой документи от c_1 са били погрешно отнесени към c_2 . Матрицата на объркване може да покаже възможности за подобряване точността на системата.

14.7 Размяната „отклонение – разлика“ (The bias – variance tradeoff)

Нелинейните класификатори са по-мощи от линейните. За някои проблеми съществува нелинеен класификатор с нулева грешка при класификация, но не и линеен такъв. Значи ли това, че трябва винаги да използваме нелинейни класификатори за оптимална ефективност?

За да отговорим на този въпрос, ще разгледаме размяната „отклонение – разлика“ – една от най-важните концепции в машинното обучение. Размяната помага да се обясни защо няма универсален оптимален метод за обучение. Поради това избирането на подходящ метод за обучение е неизбежна част от решаването на проблемите, свързани с текстова класификация.

В тази секция използваме линейните и нелинейните класификатори като прототипни примери съответно за „не толкова мощно“ и „мощно“ обучение. Правим това опростяване поради много причини. Първо, много нелинейни модели включват линейни модели като специален случай. Например, един нелинеен модел за обучение като kNN (k най-близки съседи) в някои случаи ще породи линеен класификатор. Второ, съществуват нелинейни модели, които са по-прости от линейни модели. Трето, сложността на обучението не е свойство на класификатора, защото съществуват много аспекти на обучението (избиране на характеристики, съвместно филтриране, регуляризация, ограничения), които правят даден метод на обучение по-слаб или по-мощен без да променят типа на класификатора. В тази секция обаче тези усложнения няма да се взимат под внимание.

За оценка ще използваме не броя на правилно класифицираните тестови документи, а присъщата несигурност на етикерането. В много проблеми, свързани с текстовата класификация, дадено представяне на документ може да се появи от документи, принадлежащи на различни класове. Това се получава, защото документи от различни класове може да са свързани с едно и също представяне на документ. Например, документите „Китай съди Франция“ и „Франция съди Китай“ са свързани към едно и също представяне $d' = \{Китай,$

Франция, съди}. Но само вторият документ е релевантен спрямо класа $c' =$ „обвинения, повдигнати от Франция”.

За да опростим пресмятанията, няма да отчитаме броя грешки върху тестовото множество при оценяване на даден класификатор. Вместо това ще обърнем внимание дали и колко добре класификаторът оценява условната вероятност $P(c|d)$ даденият документ да принадлежи на определен клас. В горния пример $P(c'|d) = 0,5$.

Нашата цел е да намерим такъв класификатор γ , за който $\gamma(d)$ е възможно най-близо до вероятността $P(c|d)$, където d е документ. Това измерваме чрез средноквадратичната грешка (MSE): $MSE(\gamma) = E_d[\gamma(d) - P(c|d)]^2$, където E_d е очакването по отношение на $P(d)$.

Казваме, че класификаторът γ е оптимален за разпределението $P(<d,c>)$, ако той минимира $MSE(\gamma)$.

Желателно е класификаторите да минимишрат MSE. Нужен ни е и критерий за обучаващите методи. Обучаващ метод Γ е функция, на която се подава етикирано обучаващо множество D и която връща класификатор γ .

Нашата цел е да намерим такъв метод Γ , който, използвайки обучаващи множества, учи класификатори γ с минимална MSE. Можем да формализираме това, като минимишране на обучаващата грешка: грешка-при-обучение(Γ) = $E_D[MSE(\Gamma(D))]$, където E_D е математическото очакване при етикирани обучаващи множества. За опростяване на нещата може да считаме, че обучаващите множества имат фиксиран размер. Тогава разпределението $P(<d,c>)$ определя разпределение $P(D)$ над обучаващите множества.

Можем да използваме обучаващата грешка, като критерий при избора на обучаващ метод за текстова класификация. Методът Γ е оптимален обучаващ метод за разпределението $P(D)$, ако минимишира обучаващата грешка.

За по-прегледно ще пишем Γ_D вместо $\Gamma(D)$.

$$\text{грешка-при-обучение}(\Gamma) = E_D[MSE(\Gamma_D)] = E_D E_d[\Gamma_D(d) - P(c|d)]^2 = E_d[\text{отклонение}(\Gamma, d) + \text{разлика}(\Gamma, d)]$$

$$\text{отклонение}(\Gamma, d) = [P(c|d) - E_D \Gamma_D(d)]^2$$

$$\text{разлика}(\Gamma, d) = E_D[\Gamma_D(d) - E_D \Gamma_D(d)]^2$$

Нека $\alpha = P(c|d)$ и $x = \Gamma_D(d)$. Тогава:

$$E[x - \alpha]^2 = Ex^2 - 2Ex\alpha + \alpha^2 = (Ex)^2 - 2Ex\alpha + \alpha^2 + Ex^2 - 2(Ex)^2 + (Ex)^2 =$$

$$= [Ex - \alpha]^2 + Ex^2 - E2x(Ex) + E(Ex)^2 = [Ex - \alpha]^2 + E[x - Ex]^2$$

$$E_D E_d[\Gamma_D(d) - P(c|d)]^2 = E_d E_D[\Gamma_D(d) - P(c|d)]^2 = E_d[[E_D \Gamma_D(d) - P(c|d)]^2 + E_D[\Gamma_D(d) - E_D \Gamma_D(d)]^2]$$

D и d са взаимно независими. За случаен документ d и случаен обучаващо множество D , D не съдържа етикиран екземпляр на d .

Отклонение е повдигнатата на квадрат разлика между $P(c|d)$ и $\Gamma_D(d)$, където $P(c|d)$ е условната вероятност d да е в класа c , а $\Gamma_D(d)$ е предвиждането на обучаващия класификатор. Ако обучаващият метод прави класификатори, които постоянно грешат, отклонението е голямо. Отклонението е малко, ако:

- класификаторите винаги класифицират правилно;

- различни обучаващи множества водят до грешки при различни документи;
- различни обучаващи множества водят до позитивни и негативни грешки при едни и същи документи, но средноаритметичното им клони към 0.

Ако е изпълнено едно от тези три условия, то тогава $E_D\Gamma_D(d)$, очакването върху всички обучаващи множества, клони към $P(c|d)$.

Линейни методи като Rocchio и Naive Bayes имат голямо отклонение при нелинейни проблеми, защото те могат да моделират само един тип граница на клас – линейна хиперравнина. Ако генериращият модел, $P(<d,c>)$, има сложна нелинейна граница на клас, отклонението ще е голямо, защото много на брой точки ще бъдат грешно класифицирани.

Можем да мислим за отклонението като за резултат от нашето знание за дадена област (или съответно за липсата на такова), което вграждаме в класификатора. Ако знаем, че истинската граница между двата класа е линейна, тогава линеен метод, създаващ линейни класификатори, има по-голям шанс за успех от нелинеен метод. Но ако истинската граница не е линейна и ние неправилно отклоним класификатора да бъде линеен, тогава точността при класификацията ще бъде много малка.

Нелинейните методи като kNN имат малко отклонение. В зависимост от разпределението на документи в обучаващото множество, научените граници на решение могат да варират значително. Поради това съществува възможност всеки документ да бъде класифициран правилно за някакво обучаващо множество. Затова средното предвиждане $E_D\Gamma_D(d)$ е по-близо до $P(c|d)$ и отклонението е по-малко отколкото при линеен обучаващ метод.

Разлика наричаме промяната в предвиждането на обучените класификатори: средната разлика, повдигната на квадрат, между $\Gamma_D(d)$ и $E_D\Gamma_D(d)$. Разликата е голяма, ако различни обучаващи множества D пораждат силно различаващи се класификатори Γ_D . Разликата е малка, ако обучаващото множество влияе минимално върху решението на класификатора Γ_D , независимо дали тези решения са правилни или не. Разликата измерва колко променливи са решението, а не дали те са верни или не.

Линейните обучаващи методи имат малка разлика, тъй като повечето случайно получени обучаващи множества генерираат подобни хиперравнини на решение. Линии на решение, генериирани от линейните обучаващи методи на фиг. 14.10 и фиг. 14.11 ще се отклоняват леко от основните граници на класовете в зависимост от обучаващото множество, но относянето към даден клас за повечето документи няма да бъде засегнато. Заградената в кръг област на фиг. 14.11 ще бъде отново неправилно класифицирана.

Нелинейните методи като kNN имат по-голяма разлика. kNN може да моделира много сложни граници между два класа. Затова е чувствителен към документи с шум като тези на фиг. 14.10. Съответно разликата при kNN е голяма. Понякога тестовите документи се класифицират неправилно – в случай, че се окажат близо до документ с шум в обучаващото множество – а понякога се класифицират правилно, ако близо до тях в обучаващото множество няма документи с шум. Резултатът силно зависи от обучаващото множество.

Обучаващите методи, при които разликата е голяма, са склонни да изменят обучаващото множество. Целта при класификацията е да се нагласят обучаващите данни до степен, при която обхващаме реалните свойства на разпределението $P(<d,c>)$. При изменянето

обучаващите методи се учат и от шум. Изменението увеличава MSE и често е проблем за обучаващите методи с голяма разлика.

За разликата можем да мислим още като за „сложност на модела“ или, еквивалентно, „ капацитет на паметта“ на обучаващия метод, т.е. колко детайлна характеристизация на обучаващото множество може да запомни и после да приложи върху нови данни. Този капацитет съответства на броя независими параметри, които могат да паснат на обучаващото множество. Всяка kNN област S_k взема независимо решение за класификация. По този начин капацитетът на kNN е ограничен само от размера на обучаващото множество. kNN може да запомня произволно големи обучаващи множества. За разлика от kNN, при Rocchio броят параметри е фиксиран – J параметъра за всяко измерение, един за всеки център на тежестта – и независим от размера на обучаващото множество. Rocchio не може да запомня в подробности детайлите на разпределението на документите в обучаващото множество.

Целта ни при подбора на обучаващ метод, е да минимизираме грешката при обучение. Грешката при обучение има два компонента – отклонение и разлика, които в общия случай не могат да бъдат минимизирани едновременно. В повечето случаи сравняването на два обучаващи метода Γ_1 и Γ_2 се свежда до това единият метод да има по-голямо отклонение и по-малка разлика, а другият – по-малко отклонение и по-голяма разлика. Тогава избирането на един обучаващ метод не се състои просто в това да избереш този метод, който генерира добри класификатори върху обучаващи множества (малка разлика), или този, който може да научи класификационни проблеми с много трудни граници на решение (малко отклонение). Вместо това трябва да претеглим съответните качества на отклонението и разликата в напечето приложение и да изберем според тях. Това наричаме „размяна отклонение-разлика“.

Фиг. 14.10 изобразява пример за размяна отклонение-разлика. Някои китайски текстове съдържат английски думи, написани с латински букви (напр. CPU, ONLINE, GPS). Да разгледаме задачата да се отделят уеб страниците, съдържащи само китайски символи, от тези, които съдържат и латински символи. Дадена търсачка може да предложи на китайски потребители да филтрира смесените страници. За тази класификация използваме две характеристики: броя латински символи и броя китайски символи на уеб страницата. Разпределението $P(< d, c >)$ генерира повечето смесени документи над линията от къси тирета, а китайските – под нея, но има няколко документа с шум.

На фиг. 14.10 виждаме три класификатора:

- **Класifikator с една характеристика** – изобразен е с хоризонталната линия от точки. Този класifikator използва само една характеристика – броя латински букви. Допускаме, че сме използвали обучаващ метод, който минимизира броя неправилни разпределения в обучаващото множество. Тогава положението на хоризонталната граница на решение не е много засегнато от разлики в обучаващото множество (т.е. от документи с шум). Следователно, обучаващ метод, генериращ такъв тип класификатори има малка разлика (variance). Но неговото отклонение е голямо, тъй като той ще класифицира неправилно квадрати в долния ляв ъгъл и ще изобразява със запълнени кръгове документи, съдържащи повече от 50 латински букви.
- **Линеен класifikator** – изобразен е с линията от дълги тирета. Обучението на линеен класifikator има по-малко отклонение, тъй като само документите с шум

и вероятно няколко документа близо до границата между двата класа са неправилно класифицирани. Разликата е по-голяма отколкото при класификаторите с една характеристика, но все още е малка. Линията от дълги тирета се отклонява леко от истинската граница между двата класа. Така ще се държат почти всички линейни граници на решения, научени от обучаващи множества. По този начин много малко документи (документи, които са близо до границата на класовете) ще бъдат променливо класифицирани.

- **Класifikator „Нагласи-обучаващото-множество-перфектно”** – изобразен е с непрекъсната линия. Тук обучаващият метод конструира граница на решение, която перфектно разделя класовете в обучаващото множество. Този метод има най-малко отклонение, защото няма документ, който да се класифицира постоянно погрешно – класификаторите понякога дори разпределят правилно документи с шум в тестовото множество. Отклонението обаче е голямо при този метод. Тъй като документите с шум могат да преместят границата на решение произволно, тестовите документи, които са близо до документи с шум в обучаващото множество, ще бъдат погрешно класифицирани.

Може би е изненадващо, че толкова много от най-известните алгоритми за текстова класификация са линейни. Някои от тези методи са измежду най-ефективните известни ни методи. Типичните класове при текстовата класификация са сложни и изглежда неправдоподобно да бъдат добре моделирани линейно. Тази интуитивност е подвеждаща при многомерните пространства, с които обикновено се сблъскваме в текстовите приложения. С увеличаване броя на измеренията вероятността за линейна отделимост се увеличава значително. Дори по-мощни нелинейни обучаващи методи могат да моделират граници на решение, които са по-сложни от хиперправнина, но са и също така по-чувствителни към шум в обучаващите данни.

15 Метод на опорните вектори и машинно самообучение върху документи

Подобряването ефективността на класифицирането е много развиващата се област в машинното самообучение през последните две десетилетия, и резултата от тази работа доведе до нова генерация класификатори, като метод на опорните вектори, известен като SVM (support vector machine), оптимизирани дървета за избор, регуляризирана логистична регресия, невронни мрежи и рандомизирани гори. Много от тези методи, включително и SVM, основната тема в тази глава, са приложени успешно към проблемите на извличане на информация, в частност към текстовата класификация. SVM е вид класификатор на голямото отстояние (large-margin classifier): той основан на метода на машинното самообучение, където целта е във векторното пространство да се намери граница между два класа, които са максимално далече от всяка точка в обучаващото се множество от данни.

15.1 SVM: Случай на линейното разделяне

За два класа, които са отделни множества данни, съществуват голям брой линейни разделители. Интуитивно мислената линия по средата между данните от двата класа изглежда по-добре, отколкото тази, която минава много близо до всеки от елементите на двата класа. Някои от методите като алгоритъмът „персептрон“, откриват всеки възможен разделител, докато други, като наивният Бейсов алгоритъм, търсят най-добрая разделител според определени условия. SVM в частност определя условията, така че да се търси такава повърхност, която да е максимално далече от всяка точка. Това разстояние от повърхността до най-близката точка определя разстоянието от класификатора.

При този метод функцията, която изчислява повърхнината е напълно определена бикновено от малки подмножества от данни, които определят позицията на разделителя. Тези точки се наричат опорни вектори.

Даването на максимална стойност на разстоянието изглежда удачно, защото точките близо до повърхнината определят много неточно класификационно решение. Класификатор с голямо отстояние прави не достатъчно точно решение. Това ни дава класификационно разумно разстояние: малка грешка в измерването или малка промяна в данните, няма да доведе до грешна класификация. Сравнено с хиперравнинното решение, ако искаме да използваме широк разделител между класовете, имаме много малко възможности как да го разположим. Като резултат от това, намаляваме използваната памет, и следователно очакваме способността за точното изследване на данните да се увеличи.

Нека да формализираме SVM с малко алгебра. Хиперравнината може да бъде дефинирана като пресечница на терм b и на нормален вектор W , които е перпендикулярен на хиперравнината. Този вектор в тематиката на машинното самообучение е известен като “вектор на теглата”. За да изберем сред всички хиперравнини, които са перпендикулярни на нормалния вектор, използваме терма b . Понеже хиперравнината е перпендикулярна на нормалния вектор всички точки x от повърхнината удовлетворяват равенството $wx = -b$. Сега нека да вземем множество от точки $D = \{x_i, y_i\}$, където всяко число е двойка от точка x и клас y съответен на

него. За SVM двата класа данни винаги се означават с $+1$ и -1 , а пресичащият терм винаги е означаван с b . Така получаване линейният класификатор $f(x) = \text{sign}(w^t x + b)$. Стойност -1 определя единия клас, $+1$ другия.

Можем да бъдем сигурни в класификацията си за точка, ако тя е далеч от границата. За дадено обучаващо се множество данни и хиперповърхнина, дефинираме функционално разстояние от i -то то x_i , което се отнася към хиперравнината (w, b) като количество $y_i(w^t x_i + b)$. Функционалното разстояние на данните с отношение към повърхнината е два пъти функционалното разстояние на точките от множеството с минимално функционално разстояние. Но съществува проблем с използването на дефиницията: стойността не е константа, защото можем винаги да направим функционалното разстояние голямо колкото искаме, като просто увеличим w и b . Това предполага, че трябва да използваме никаква константа за максималния размер на вектора w . За да разберем как точно да я определим, нека да използваме геометрия. Разстояние от точка x до границата ще го означим с r . Знаем че най-малкото разстояние между точка и повърхнина е перпендикуляр към равнината, т.e успореден на вектора w . Единичният вектор е $\frac{w}{|w|}$. Нека да означим най-близката точка на повърхнината до x с x' . Тогава $x' = x - yr \frac{w}{|w|}$, където умножението с y просто променя знака за двета случая, x да бъде от двете страни на повърхнината. Още повече x' лежи на границата и удовлетворява уравнението $w^t x' + b = 0$. Следователно $w^t(x - yr \frac{w}{|w|}) + b = 0$. Ако изразим r от него получаваме

$r = y(\frac{w^t x + b}{|w|})$. Точките най-близо до хиперравнината са support vectors. Геометричното разстояние на класификатора е максималната ширина на линията, която може да бъде описана така че да раздели support vectors на двата класа. Можем да използваме единични вектори, като изискваме $|w| = 1$. Това ще направи геометричното разстояние да бъде еднакво с функционалното разстояние.

Можем да увеличаваме функционалното разстояние, за решаване на сложни SVM. Нека да изберем функционалното разстояние на всички точки да бъде поне 1 и нека да бъде равно на 1 за поне един вектор. Тогава за всички елементи от множеството $y_i(w^t x_i + b) > 1$ и съществуват support vectors, за които неравенството се превръща в равенство. Разстоянието от хиперравнината е $r = y_i(\frac{w^t x_i + b}{|w|})$, геометричното разстояние е $= \frac{2}{|w|}$. Нашата цел е да максимираме геометричното отстояние. Така че трябва да намерим w и b такива, че :

- $p = \frac{2}{|w|}$ е максимално
- За всяко $(x_i, y_i) \in D$, $y_i(w^t x_i + b) \geq 1$

За да максимираме $\frac{2}{|w|}$ е същото като да минимираме $\frac{|w|}{2}$. Това дава финална формализация на SVM за минимизирання проблем:

Да се намерят w и b такива че

- $\frac{1}{2} w^t w$ е минимално и
- За всяко $\{(x_i, y_i)\}$, $y_i(w^t x_i + b) \geq 1$

Решението изисква използването на множителите на Лагранж a_i , които се асоциират с всяко $y_i(w^t x_i + b) \geq 1$. Тоест трябва да намерим $a_1 \dots a_N$, такива че $\sum a_i - \frac{1}{2} \sum i \sum j a_i a_j y_i y_j x_i^t x_j$ е максимално малко и

- $\sum i a_i y_j = 0$
- $a_i \geq 0$ за всяко $1 \leq i \leq N$

Това води до

- $w = \sum a_i y_i x_i$
- $b = y_k - w^t x_k$ за някое x_k такова че $a_k \neq 0$

В решението повечето a_i са нули. Всяко ненулево a_i означава, че съответното x_i е support vector. Тогава получаваме класификационната функция:

$$f(x) = \text{sign}(\sum i a_i y_i x_i^t x + b)$$

15.2 Разширения на SVM модела

15.2.1 Класификация

За много високи размерности на пространството, а именно каквато е текстовата класификация, понякога проблемите свързани с тях са линейно разрешими, но в общия случай не са, а и дори да са решими, по-добре е да предпочетем решение, което прави по-добро разделение на данните, пренебрегвайки някои малки грешки в документите. Ако множеството от данни D не може да бъде линейно разделено, стандартното решение да позволим на по-голямо разстояние да направи някои грешки. Тогава, разбира се, си плащаме за всяки грешно класифициран елемент, като всяка такава грешка зависи от това колко голямо е било разстоянието в сравнение с ограниченията, които дадохме малко по-горе във формулите. За да постигнем това обаче, ще използваме променливата ξ_i . Ненулева стойност за тази променлива позволява на x_i да не отговори на ограниченията за разстоянието на цена пропорционална на стойността на ξ_i . Формулировката на SVM оптимизационния проблем със слаба променлива ξ_i е следната:

Да се намерят w, b и $\xi_i \geq 0$, такива че

- $\frac{1}{2} w^t w + C \sum i \xi_i$ е минимално
- и за всяко $\{(x_i, y_i)\}, y_i(w^t x_i + b) \geq 1 - \xi_i$

Оптимизационния проблем се основава на това да определи колко голямо да се направи разстоянието пред това колко точки да бъдат преместени за да може да се позволи това разстояние. Разстоянието може да бъде по-малко от 1 за точка x_i като вземем $\xi_i \geq 0$, но тогава ще имаме грешка $C \xi_i$ в минимизацията. Сумата на ξ_i ни дава горна граница за броя на допуснатите грешки. Параметърът C се използва за регулатор – когато нараства се намаля геометричното разстояние, когато намалява може да използваме слабата променлива ξ_i и да си позволим по-голямо разстояние, така че тя моделира размера на данните. Тогава проблема за класификацията с използване на гъвкаво разстояние е следния:

Да се намерят $a_1 \dots a_N$ такива че $\sum a_i - \frac{1}{2} \sum i \sum j a_i a_j y_i y_j x_i^t x_j$ е максимално и

- $\sum i a_i y_j = 0$
- $0 \leq a_i \leq C$ за всяко $1 \leq i \leq N$

Оттук виждаме че нито слабата променлива ξ_i , нито множителите на Лагранж не участват в формулировката на проблема. Всичко, което ни остава е константата C ограничаваща възможният размер на множителите на Лагранж за support vector точките. Както и по-горе x_i , с ненулеви a_i , ще бъдат support vectors. Решението на текущия проблем е следното:

$$w = \sum a_i y_i x_i$$

$$b = y_k(1 - \xi_k) - w^t x_k \text{ за } k = \arg \max_k a_k$$

В повечето случаи, support vectors са малък процент от обучаващото се множество данни. Ако данните не могат да бъдат разделени или могат да бъдат, но с малко разстояние, тогава всяка точка, която е грешно класифицирана или с някакво разстояние ще има ненулево a_i . Сложността от тестването с линейният метод е показана на следната таблица. Работата напоследък е съсредоточена да се намали тази сложност, често като се достигне до някакво приблизително и задоволително решение. Стандартната сложност е около $O(|D|^{1.7})$

Classifier	Mode	Method	Time complexity
NB	training		$\Theta(D L_{ave} + C V)$
NB	testing		$\Theta(C M_a)$
Rocchio	training		$\Theta(D L_{ave} + C V)$
Rocchio	testing		$\Theta(C M_a)$
kNN	training	preprocessing	$\Theta(D L_{ave})$
kNN	testing	preprocessing	$\Theta(D M_{ave}M_a)$
kNN	training	no preprocessing	$\Theta(1)$
kNN	testing	no preprocessing	$\Theta(D L_{ave}M_a)$
SVM	training	conventional	$O(C D ^3 M_{ave});$ $\approx O(C D ^{1.7} M_{ave}), \text{ empirically}$
SVM	training	cutting planes	$O(C D M_{ave})$
SVM	testing		$O(C M_a)$

15.2.2 Мнокласови SVMs

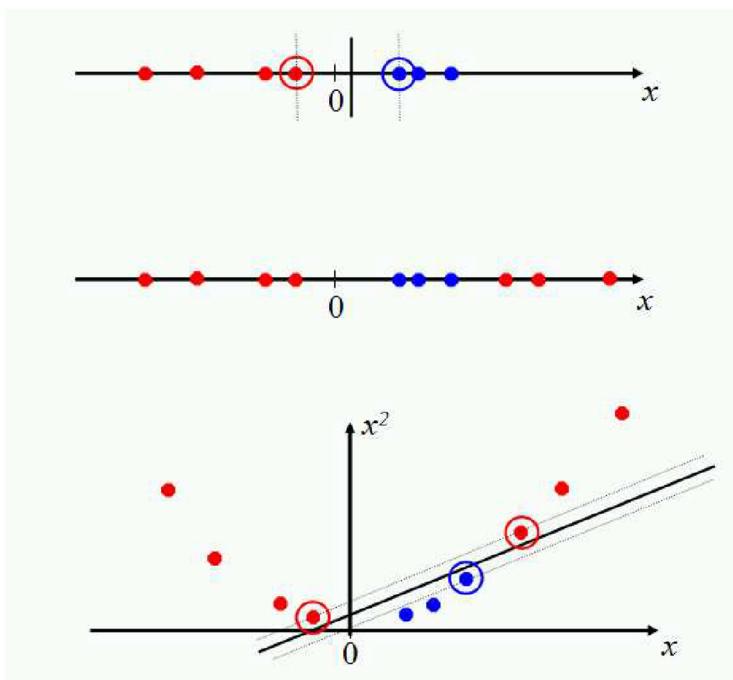
SVM са двукласови класификатори. Най-използваната техника в практиката показва да използваме „един срещу всички“ класификатори и да избираме клас, който класифицира данните с възможно най-голямо разстояние. Друг подход е да се построи множество от „всеки срещу всеки“ класификатори, и да се избере класът, който е бил избран от повечето класификатори. Тези методи обаче се оказват не толкова елегантни за решаването на многокласовата класификация. По-добра алтернатива се постига чрез конструирането на мултикласови SVMs, където се използва двукласов класификатор чрез вектор $\phi(x, y)$. При класифицирането, класификатора избира клас $y = \arg \max_{y'} w^t \phi(x, y)$. Разстоянието тогава

е мястото между разстоянието на точния клас и най-близкият клас, и формулировката тогава ще изисква

$\forall i \forall y \neq y_i w^t \phi(x_i, y_i) - \phi(x_i, y) \geq 1 - \xi_i$ Този обобщен метод може да бъде разширен да дава формулировка за различни типове линейни класификатори, които да се използват при повече от два класа.

15.2.3 Нелинейни SVMs

Досега представихме, обучаващо се множество данни, което е линейно разделимо (може би с малко изключения) и дотук всичко изглеждаше доста добре. Но как стои въпросът ако данните не могат да бъдат класифицирани с линеен класификатор. Нека да видим първо едномерният случай. На картилката се вижда, че е невъзможно да разделим данните. Едно решение е да прехвърлим данните в двумерното пространство и да използваме линеен класификатор както е показано на картилката по-долу



Тук основната идея е да се премесят данните в пространство с по-голяма размерност, където евентуално да могат да бъдат разделени, като разбира се се стремим да запазим пропорциите между точките от множеството така че полученият класификатор да работи добре. SVMs и също няколко други линейни класификатори предлагат лесен и ефективен начин за това преструктуриране на данните от в пространство с по-голяма размерност. Това действие е известно като „the kernel trick“. То не е никакъв трик обаче а просто извества така данните, че е трудно да се проследи какво точно е станало. SVM линейният класификатор се основава на скаларното произведение между векторите на точките. Нека $K(x_i, x_j) = x_i^t x_j$. Тогава класификатора, който имахме досега е

$$f(x) = \operatorname{sign} \left(\sum_i a_i y_i K(x_i, x) + b \right)$$

Сега да предположим че преобразуваме данните в пространство с по-голяма размерност посредством трансформацията $\phi: x \mapsto \phi(x)$. Тогава скаларното произведение става $\phi(x_i)^t \phi(x_j)$. Функцията K е такава функция, която съответства на скаларното произведение в някои обобщени векторни пространства.

Светът на SVM идва със своя собствен език, който е доста по различен от езика, който се използва в машинното самообучение. Терминологията се основава да доста сложни математически термини, но даже и в тази тема засегнахме само някои от по-простите неща, но въпреки това трябва да кажем, че разбирането на чисто математическият модел на нещата няма да навреди на никого.

15.2.4 Експериментални резултати

Чрез експериментални резултати е установено че SVM е много ефективен текстов класификатор и именно на това е изградил репутацията си за най-добрия метод за текстова класификация.

15.3 Проблеми при класификацията на текстови документи

Има редица софтуерни приложения за класификация на текстови документи из комерсиалния свят. Филтрирането на спам в електронните пощи е едно от най-често срещаните тези дни. Jackson и Moulinier (2002) пишат: „Разбирането на данните е един от ключовете към успешната категоризация и въпреки това, за повечето разработчици на приложения за класифициране, това е едно от най-слабите места“ В тази глава ще се върнем стъпка назад и ще обърнем повече внимание на пространството от възможни решения и на евристиките, които биха могли да се използват в определен ситуация за подобряване на ефективността и прецизността.

Първото нещо, което се гледа, преди избора на класификатор е наличието на данни. Колко такива има – николко, малко, доста, много... ? Дали нарастват всеки ден? Най-интуитивното решение, което ни идва на ум, е ръчно въвеждане на правила от типа на:

Ако условие тогава класифицирай като клас

За реални задачи, това е почти неприложимо. На практика, за да се създаде добра система за класифициране, използвайки такива условия, са необходими изключително много подобни ограничения и то такива, че да не се припокриват, да няма противоречия, да не се включват едно в друго (освен, ако не е умислено, например при йерархична класификация) и прочие. Всъщност, такива системи съществуват и дават много добри резултати (например Hayes и Weinstein (1990) постигат 94% прецизност при над 675 категории). Но създаването на подобна система отнема много време (което често не е налично), много усилия, внимаване и трудна поддръжка, имайки предвид, че базата от данни се променя постоянно.

Един от основните проблеми при създаването на системи за извличане на информация е набавянето на достатъчно голямо множество данни. За целта, може да се приложат различни стратегии, които да подтикват потребителите сами да класифицират някои документи, но това не е предмет на тази статия и няма да се задълбочаваме над него.

Ако има достатъчно голямо множество от вече класифицирани данни, тогава и повече алгоритми може да бъдат използвани. В това число и SVM. Но пък от друга страна, ако се

предложи линеен SVM, то той трябва да е по-добър от някоя стандартна система с булеви правила. Особено затруднение тук е, че ако даден клиент поиска да се поправи някоя конкретна грешка на класифициране, то при система със стандартните булеви правила това може по-лесно да се нагласи, докато при SVM е по-трудно да се нагласят теглата така, че да се прекласифицира даден пример без да се засягат такива примери, които са класифицирани правилно.

В случаите, когато се прави система за класифициране за определена тематика или за определена област, могат да се приложат стратегии за подобряване на класифицирането и ефективността ѝ. Например задаване на допълнителни „насоки“ на самия класifikатор (като определени думи с по-голяма тежест, например), които да помогнат за по-голяма точност.

Когато текстовете са добре разграничими и категориите са сравнително малко, тогава по-голяма част от алгоритмите за класифициране на документи ще работят доста прецизно. Но в повечето случаи, при реални задачи, категориите са много на брой и често се припокриват. Освен това, често категориите са йерархични и това допълнително усложнява нещата. За целта, може да се опитат йерархични класификации. Въпреки това, на този етап, използването на такъв йерархичен подход много слабо надминава ефективността на стандартните подходи, които директно работят с листата от йерархичната структура на отделните класове.

Обикновено, повечето документи имат т. нар. „зоni“. Например, електронните писма имат хедъри, в които се записват автор, тема, различни ключови думи (помагащи при търсене) и прочие. Класifikаторите на текст могат да се възползват от тези специфични зони, като извличат по-компресирана и важна информация, директно подадена от потребителя. Така може да се въведе отегляване на различните зони, като например даване на по-силна тежест на думите, намиращи се в заглавието на дадена статия.

Друга подобна стратегия, която използва различни тегла за термите в текста, е да се създават цели множества от особености и съответни параметри за всяка дума, появяваща се в определена зона. С други думи, за някои терми се дава различна тежест, в зависимост от частта, в която се намират.

Подобрене може да се постигне и чрез друг подход – при класификация на даден документ, да не се гледа целия му текст, а само определени части – определени изречения, които са на определена позиция или имат специфично значение. Например Koltz et. al. (2000) използват такава форма за откриване на спецификите на даден документ, която е базирана изцяло на думи, намиращи се в определени зони. Използвайки различни проучвания за обобщение на текстове, методът използва само: (1) заглавието, (2) първия параграф, (3) параграфът, съдържащ най-много думи от заглавието или ключови думи, (4) първите два параграфа или първия и последния такъв или (5) всички изречения, които съдържат някакъв минимален брой ключови думи или думи от заглавието на текста. Оказалось се, че това води до подобрене на класифицирането. Ко et. al. (2004) пък използват задаване на по-голяма тежест на изреченията, които съдържат думи от заглавието или такива, които са по-важни за съдържанието на документа и това довело до повишаване на точността с около 1%.

15.4 Машинно самообучение и извлечане на информация

Вместо да измисляме сами различни теглови функции на документи, можем да оставим това на машинното самообучение. Накратко – един класификатор може да бъде обучен от множество примери, които са предварително класифицирани, т.е. даденият класификатор може сам да определи тегловите коефициенти, само на базата на обучаващи примери. Така нашият проблем може лесно да бъде превърнат в задача за оценка на това дали дадена заявка може да се свърже с даден документ или не.

Без да се задълбочаваме много над машинното самообучение, тъй като то е отделна дисциплина с множество различни подходи, само ще споменем как то може да бъде приложено за нашата задача.

Ще развием идеята за по-елементарен случай – нека имаме линейна функция с два параметъра – единият е косинусовата мярка за подобие между документ и заявка, а другият – минимален прозорец w около даден документ, в който ще лежи определена заявка. За момента ще разгледаме този елементарен модел, тъй като е по-лесен и за разбиране, и за визуализация, като трябва да се има предвид, че той може да бъде обобщен за много по-серизни задачи.

Първото необходимо нещо в методите за машинно самообучение с учител, е множество от обучаващи примери, което съдържа двойки (*документ, заявка*), както и оценка на тези двойки: *отговарящ* или *неотговарящ* (касaeщо документът спрямо заявката). За всяка една такава двойка можем да изчислим косинусовото подобие и „широкината“ на прозореца w .

Нека например разгледаме примерното множество, зададено в тема №6, секция 6.1.2 (лявата колонка) и за тях пресметнем новите два параметъра (дясната колонка).

Example	DocID	Query	Cosine score	ω	Judgment
Φ_1	37	linux operating system	0.032	3	relevant
Φ_2	37	penguin logo	0.02	4	nonrelevant
Φ_3	238	operating system	0.043	2	relevant
Φ_4	238	runtime environment	0.004	2	nonrelevant
Φ_5	1741	kernel layer	0.022	3	relevant
Φ_6	2094	device driver	0.03	2	relevant
Φ_7	3191	device driver	0.027	5	nonrelevant
...

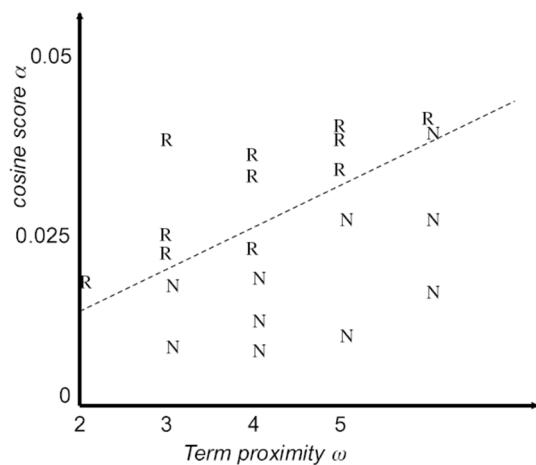
Ако означим с a косинусовата оценка и с w съответния прозорец, то може да изразим отношението на дадена заявка към даден документ така:

$$\text{Score}(d, q) = \text{Score}(a, w) = a\alpha + bw + c,$$

където a , b и c са реални коефициенти. Нашите примери могат лесно да се изобразят в двумерното пространство например така:

Example	DocID	Query	s_T	s_B	Judgment
Φ_1	37	linux	1	1	Relevant
Φ_2	37	penguin	0	1	Non-relevant
Φ_3	238	system	0	1	Relevant
Φ_4	238	penguin	0	0	Non-relevant
Φ_5	1741	kernel	1	1	Relevant
Φ_6	2094	driver	0	1	Relevant
Φ_7	3191	driver	1	0	Non-relevant

На дадената картичка, R е означение за отговарящ (relevant), а N – за неотговарящ (nonrelevant). Естествено, съществено важно е да се дефинира и функция за оценка на грешката, но няма да се задълбочаваме и над това.



За функцията Score, ще приемем, че ако резултатът от нея е близо до 1, то заявката е R, а ако е близо до 0 – N. Ясно е, че няма как тази функция да връща само нули или единици и по тази причина ще ни е необходим праг Θ и ще „бинаризираме“ резултата така: ако $\text{Score}(\alpha, w) > \Theta$, то резултатът ще е R, иначе – N.

Геометрично, можем да прокараме линия на тази графика, както е показано на фигурата (с пунктир), който да разделя резултата на два класа – тези с оценка R, и тези с оценка N. Ясно е, че тази линия (в случаите на линеен класификатор, разбира се, понеже не винаги съществува линейна функция, която разделя двата класа, но си има други подходящи методи за класифициране, например алгоритми като k-NN, или за научаване на нелинейни функции (например с невронна мрежа)) може да се определи с намиране на коефициентите a, b и с. И вместо да се опитваме ръчно да нагласим тези коефициенти, това може да стане автоматично чрез някой метод за машинно самообучение, който да итерира няколко пъти (често стотици или дори хиляди – зависи от алгоритъма, множеството примери, целите на задачата, прецизността, която гоним и прочие) само над подаденото множество примери.

Тази идея лесно може да бъде, както вече казахме, обобщена за функции с много повече параметри. Съществуват много други критерии, които да допринасят за по-качествена оценка на отношението между документ и зададена заявка. Подобно на това, машинното самообучение може да бъде използвано и за обучението на една такава SVM.

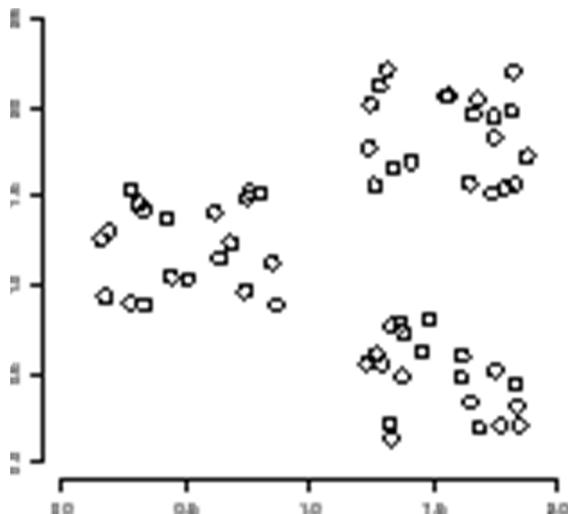
Разбира се, това не е най-добрият метод за извлечение на информация. Може да мислим за машинното самообучение като стандартна задача за регресия, при която целта е да се ранкира дадено множество от документи, на базата на вече класифицирани такива, които ще играят ролята на обучаващо множество. Това ни дава предимството, че може да се изгради и релевантност между отделните документи, а не само от една заявка към всички останали. Това е особено голям плюс за системи за търсене из мрежата, където именно ранкирането е много важно – с цел да се изведат първите няколко най-близки резултата. Такава система може лесно да бъде изградена, използвайки структурна SVM, където класът за предсказване е ранга на дадена заявка.

С малки модификации може да бъде изграден и оценяващ чрез рангове SVM, но няма да се задълбочаваме над това.

Трябва да се отбележи, че тук описаните идеи могат да се прехвърлят не само към функции с повече от две променливи, но и от по-висока степен. Това отново може да се постигне с машинно самообучение, но на този етап, изследвания показват, че машинното самообучение дава оптимални и много добри резултати за тегла, но на линейни функции. И не толкова сполучливи при нелинейни такива, където засега най-уместна е намесата на експерти в областта, които да дефинират ранкиращи функции. Причината за това е, че такива функции се изучават от години, но от сравнително скоро се прилагат методи за машинно самообучение за тази цел. Разбира се, очакванията са това да се подобри значително в бъдеще.

16 Плоско къстериране

Алгоритмите за къстериране групират множество от документи в подмножества или *къстери*. Целта на алгоритмите е, да създават къстери, които са вътрешно свързани, но ясно разграничими един от друг. С други думи казано, документите в един къстар трябва да бъдат максимално подобни един на друг и максимално различни от тези в другите къстери.



Фиг. 1 Пример за множество от данни с ясна къстарна структура.

Къстерирането е най-разпространеният вид обучение без учител. Без учител означава, че няма човек (експерт), който да е разделил документите на класове. При къстерирането, разпределението и състава на данните определят членството в къстера. Прост пример е даден на фиг.1. Визуално е ясно, че има 3 отделни къстера от точки. Тази глава представя алгоритми, намиращи къстери без човешка намеса.

Разликата между къстериране и класификация не изглежда голяма на пръв поглед. В крайна сметка и в двата случая има разделяне на множество от документи в групи, но както ще видим, тези два проблема са фундаментално различни. Класифицирането е вид контролирано обучение - целта е да се възпроизведат категориите, които човек (супервайзор) налага на данните. При обучението без учител, чийто най-важен пример е къстерирането, няма учител (супервайзор), който да дава указания.

Ключовият елемент за алгоритмите за къстериране е мярката за разстояние. В примера от фиг.1, мярката е евклидовото разстояние в равнината. Тази мярка предполага 3 различни къстера на картицата. При къстериране на документи, също често се използва евклидово разстояние. Различните мерки дават различно къстериране. И така, мярката за разстояние е важно средство, чрез което можем да влияем на изхода от къстерирането.

Плоското къстериране създава множество от къстери без ясна структура, която да свързва отделните елементи. Йерархичното къстериране създава йерархия от къстери и ще бъде изучена в глава 17. Там се разглежда и сложният проблем за автоматичното етикериране на къстерите.

Втора важна разлика може да се направи между твърди и размити алгоритми за къстериране. При *твърдото къстериране*, всеки документ е член точно на един къстар. При *размитото*

кълстериране разпределението на документите не е строго - един документ има частично членство в няколко кълстера. Латентното семантично индексиране, форма на намаляване на размерността, е алгоритъм за размито кълстериране.

Тази глава мотивира използването на кълстериране в извлечането на информация, чрез представянето на редица приложения, дефинира проблема, който се опитваме да разрешим и дискутира мерки за оценяване на качеството на кълстерирането. Тя описва два алгоритъма за плоско кълстериране - *K-фредни* (K-means), твърд алгоритъм, и *Максимизиране на очакването* (Expectation-Maximization, EM), размит алгоритъм за кълстериране. K-средни е може би най-широко използваният алгоритъм за плоско кълстериране поради своята простота и ефективност. EM алгоритъмът е обобщение на K-средни и може да бъде приложен върху голямо разнообразие от представяния и разпределения на документи.

16.1 Кълстериране в извлечането на информация

Кълстертата хипотеза е основното предположение, което правим когато използваме кълстериране в извлечането на информация.

Кълстерна хипотеза: Документите от един кълстер се държат подобно по отношение на информационните нужди.

Това означава, че ако има един документ от даден кълстер, който отговаря на заявка за търсене, то вероятно и останалите членове на този кълстер отговарят. Това е така, защото групираме заедно документите, които споделят много термини.

Нека разгледаме следната таблица:

Приложение	Какво се кълстерира?	Ползи	Пример
Кълстериране на резултатите от търсенето	Резултатите от търсенето	По-ефективно представяне на информацията на потребителя	Фиг. 2
Scatter-Gather	Подмножество на колекция	Алтернативен потребителски интерфейс, търсене без писане	Фиг. 3
Кълстериране на колекция	Колекция	Ефективно представяне на информацията за разчително сърфиране	McKeown et al. (2002) , http://news.google.com
Определяне на езика	Колекция	Увеличава прецизността и връщанието на резултати	Liu and Croft (2004)
Кълстер-базирано извлечане	Колекция	По-висока ефективност, по-бързо търсене	Salton (1971a)

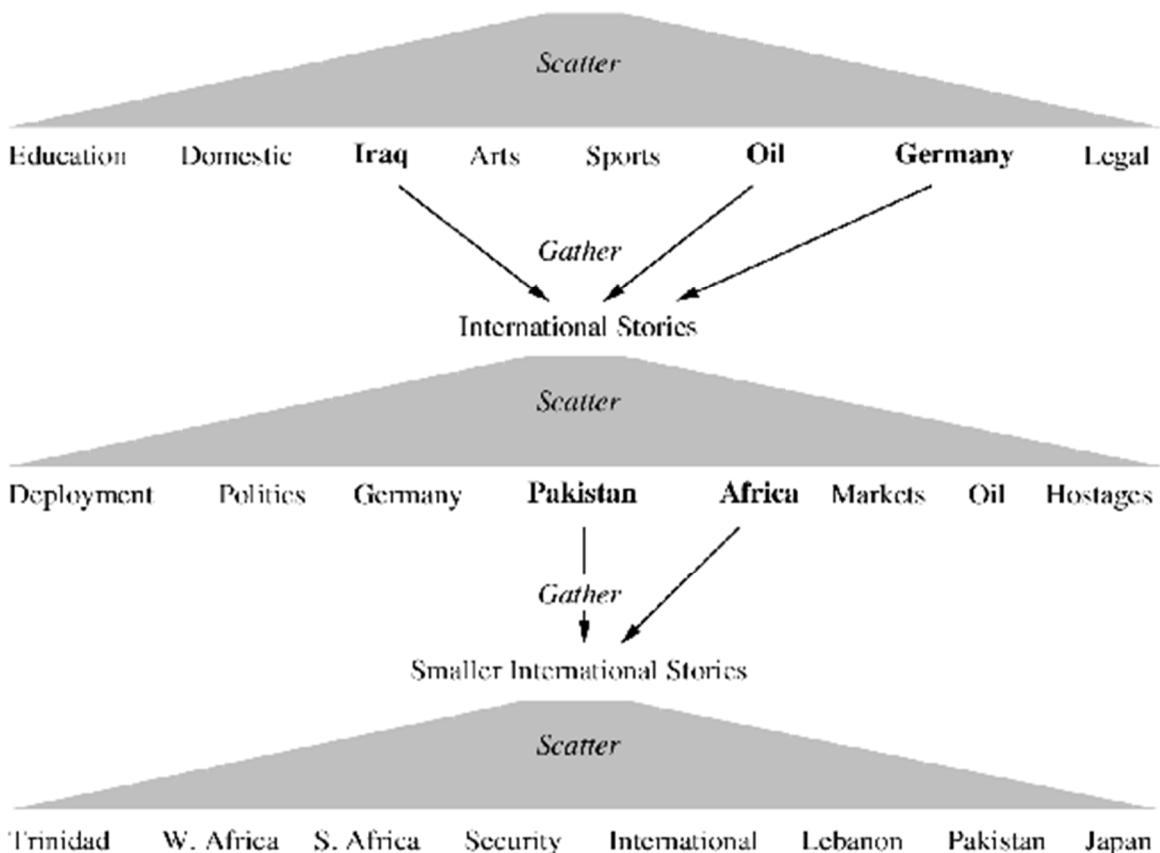
Таблица 1 Някои приложения на кълстерирането в извлечането на информация.

Тя показва някой от основните приложения на кълстерирането в извличането на информация. Те се различават според множеството данни, които кълстерират и аспекта на системата, който се опитват да подобрят – потребителско мнение, потребителски интерфейс, ефективност и ефикасност на търсачката. Всички те обаче, са базирани на предположението на хипотезата за кълстерирането.

Първото приложение споменато в таблица 1 е *кълстерирането на резултатите от търсене*, т.е. на резултатие върнати от заявка. По подразбиране в извличането на информация, тези резултати се представят като обикновен списък. Потребителят търси в списъка, докато намери търсения от него резултат. Вместо това, кълстерирането на резултатите групира документите от този списък, така че подобните се появяват заедно. Така по-лесно се сканират няколко свързани групи, отколкото много отделни документи. Това е особено полезно, ако търсената дума има няколко различни значения, какъвто е случая с думата “jaguar”, както е показано на фиг. 2.

Фиг. 2 Кълстериране на резултатите с цел подобряване на отговора. В примера никой от най-напред представените резултати не е за котката ягуар, но потребителят може лесно да достигне до търсените документи чрез избиране на кълстера котки.

Целта на Scatter-Gather също е по-добър потребителски интерфейс. Той кълстерира цялата колекция, за да получи групи от документи, които потребителят може да избере или събере. Избраният групи се обединяват и множеството получено като резултат отново се кълстерира. Този процес се повтаря, докато се намери кълстър, който ни интересува.



Фиг. 3 Пример за потребителска сесия в Scatter Gather. Колекция от новини в New York Times е кълстерирана („разпръсната“) в 8 категории (най-горния ред). Потребителят ръчно избира 3 от тях в по-малка колекция – Международни новини и извършва ново „разпръскване“. Този процес се повтаря, докато се намери малък кълстер със свързани документи (например Trinidad).

Автоматично генерираните кълстери, като тези на фиг. 3, не са толкова ясно организирани, както ръчно конструирано йерархично дърво като Open Directory. Също така, автоматичното намиране на етикети за кълстери е труден проблем. Но кълстер-базираната навигация е интересна алтернатива за търсене на ключови думи (стандартната парадигма на търсенето на информация). Това е особено вярно в случай, в който потребителите предпочитат разглеждане пред търсене, тъй като не са сигурни за термините, които да използват.

Като алтернатива на итеративното кълстериране посредством потребителски действия в Scatter-Gather, можем да изчислим статично йерархично кълстериране на колекцията, което не е повлияно от потребителя. Google News и неговият предшественик Columbia NewsBlaster system са примери за този подход. В случая с новините се нуждаем от често преизчисляване на кълстери, за да осигурим на потребителите достъп до последните актуални новини. Кълстерирането е подходящо за достъп до новини, тъй като четенето на новини не е точно търсене, а по-скоро процес на избор на подмножество от свързани събития.

Четвъртото приложение на кълстериите е пряко използване на кълстерната хипотеза за подобряване на резултатите от търсенето, чрез групиране на цялата колекция. Използваме стандартно обърнато индексиране, за да определим началния набор от документи, които отговарят на заявката, след което добавяме документи от същите кълстери, дори ако имат малко сходство със заявката. Например, ако заявката е *кала* и няколкото документа са върнати от

кълстера автомобилни документи, то може да се добавят документи от същият кълстер, в които се изпозват други термини – автомобил, превозно средство и т.н. Тази идея се използва и за определяне на езика.

Кълстерирането може също да ускори търсенето. Обърнатият индекс подкрепя бързия алгоритъм на най-близките съседи в стандартни ИИ системи. Въпреки това, понякога не можем да използваме обърнатият индекс ефикасно, например при латентно семантично индексиране (глава 18). В такива случаи бихме могли да изчислим приликата на заявката с всеки документ, но това е доста бавно. Кълстernата хипотезата предлага алтернатива: намира кълстите, които са най-близо до заявката и след това, сравнява документите само от тях. В това по-малко множество можем да изчислим приликите изчерпателно и да класираме документите по обичайния начин. Тъй като има много по-малък брой кълсти, отколкото документи, то бързо намираме най-близкия кълстер. Понеже документите, които отговарят на дадена заявка са подобни, то вероятно те ще бъдат в един и същи кълстер. Този алгоритъм може и да не е съвсем точен, но очакваното намаляване на качеството на търсене всъщност е малко.

16.2 Определяне на задачата

Можем да дефинираме целта на твърдото плоско кълстериране по следния начин: Дадено ни е множество от документи $D = \{d_1, \dots, d_N\}$, желан брой кълсти K и *целева функция*, която изчислява качеството на кълстерирането. Искаме да решим задачата $\gamma: D \rightarrow \{1, \dots, K\}$, която минимизира (или в някои случаи максимизира) целевата функция. Обикновено изискваме γ да е сюрективна, т.е да няма празен кълстер.

Целевата функция често се дефинира в термините на сходство или разстояние между документи. По-нататък ще видим, че целта на алгоритъма К-средни е да сведе до минимум средното разстояние между документите и техните центроиди, което е същото като да увеличи максимално сходството между документите и техните центроиди.

За документи, вида на сходство което искаме, обикновено е прилика в заглавието или високо съвпадение във векторно-пространствения модел. Например, документи за Китай имат високи стойности на измеренията Китайски, Пекин и Мао, а документи за Великобритания имат високи стойности за Лондон, Британия и Кралица. Апроксимираме приликата в заглавието със сходството в тъгъла или евклидовото разстояние във векторното пространство (глава 6). Ако опитваме да хванем друга прилика, различна от сходство в заглавието, например прилика в езика, тогава удачно би било различно представяне. При изчисляването на прилика в темата, можем безопасно да махнем стоп думите, но те са важни знаци за отдавяне на кълсти на английски и френски документи (в англ. ез. the се среща често la рядко, във френския е обратното).

Алтернативна дефиниция за твърдо кълстериране е, че даден документ може да бъде член на повече от един кълстер. Частичното кълстериране винаги се отнася до групиране, при което всеки документ принадлежи към точно един кълстер (при йерархичното частично кълстериране обаче, всички членове на кълстера са също членове и на родителския кълстер). Вземайки предвид дефиницията на твърдо кълстериране, позволяваща членство в няколко кълстера, разликата между твърдите и размитите алгоритми за кълстериране е в това, че при твърдите

членството в даден кълстътер е с тегло 0 или 1, докато при размитите алгоритми теглата могат да бъдат всички неотрицателни числа.

Някои изследвания правят разлика между *изчерпателни* кълстътери, които разпределят всеки документ в кълстътер, и неизчерпателни, при които документи няма да бъдат разпределени към никой кълстътер. Неизчерпателни кълстътерирания, при които документ или не е член на никой кълстътер или е член на един кълстътер се наричат изключванци. Ние ще дефинираме кълстътерирането като изчерпателно.

16.3 Кардиналност - броят на кълстътерите

Труден въпрос при кълстътерирането е определянето на броя кълстътери или иначе казано кардиналността на кълстътерирането, която ще означаваме с K . Често K е просто добро предположение, базирано на опит или други знания. Но за K -средни ще въведем евристичен метод за избиране на K и опит да включим избора му в целевата функция. Понякога приложението налага ограничения върху избора на K . Например Scatter-Gather метода от фиг. 3 не може да покаже повече от $K = 10$ кълстътера в един пласт поради размера и резолюцията на мониторите в началото на деветдесетте години.

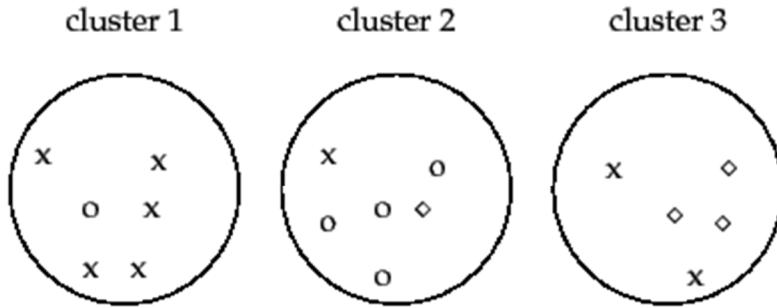
Тъй като нашата цел е да оптимизираме целевата функция, то кълстътерирането по същество е решаване на задача. Решение с пълно изчерпване е да се изброят всички възможни кълстътерирания и да се избере най-доброто. Но тъй като този брой расте експоненциално, такъв подход е неуместен. По тази причина, повечето алгоритми за плоско кълстътериране прецизират първоначалното разделяне итеративно. Ако търсенето започва в неблагоприятна точка може да се пропуснем глобалния оптимум, така че намирането на отправна точка е също важен въпрос.

16.4 Оценяване на кълстътерирането

Целевата функция в кълстътерирането формализира целта да се постигне сходството на елементите в един кълстътер и разлика между кълстътерите. Това е *вътрешен критерий* за качеството на кълстътерирането. Но добрите резултати на вътрешния критерий не е задължително да са се превърнат в добра ефективност в приложение. Алтернатива на вътрешната оценка е директна оценка в приложението. При кълстътериране на резултатите от търсене може да искаме да измерим времето, за което потребителя намира нужната му информация с различни кълстътериращи алгоритми. Това е най-директната оценка, но е доста скъпа, особено когато имаме нужда от големи проучвания на потребители.

Като заместител на потребителското мнение можем да използваме набор от класове в еталон за оценка или златен стандарт. Златният стандарт в идеалния случай е произведено от хора отсядане с добро ниво на съгласие между отделните съдии. След това можем да изчислим външен критерий, който оценява колко добре кълстътерирането пасва на класовете от златния стандарт. Например, можем да кажем, че оптималното кълстътериране на резултатите от търсенето за *ядер* от фиг. 2 трябва се състои от три групи съответстващи на трите значения на думата – коли, животни, операционна система. При този вид оценка, използваме само златния стандарт, без никакви класови етикети.

Този раздел представя четири външни критерия за качество на клъстерирането. Чистотата (purity) е проста и прозрачна мярка за оценка. Нормализирана споделена информация (normalized mutual information) може да бъде информационно-теоретично тълкувана. Индексът на Ранд (Rand index) наказва и фалшиво положителните и фалшиво отрицателните решения по време на клъстериране. F-мярката (F measure) поддържа различни тегла на тези два вида грешки.



Фиг. 4 Чистотата като външен критерий за оценка на качеството на клъстериране. Преобладаващият клас и броя на членовете на този клас за тези три клъстера са: x, 5 (клъстер 1); o, 4 (клъстер 2) и \diamond , 3 (клъстер 3). Чистотата е $(1/17) \times (5+4+3) \approx 0.71$.

За да се изчисли чистотата, всеки клъстер се съпоставя с класа, който най-често се среща в клъстера, и след това точността на това съпоставяне се изчислява, чрез броене на правилно класифицираните елементи и деление на N . Формално:

$$(1) \quad \text{purity}(\Omega, C) = \frac{1}{N} \sum_k \max_j |\omega_k \cap c_j|$$

където $\Omega = \{\omega_1, \omega_2, \dots, \omega_k\}$ е множеството от клъстери и $C = \{c_1, c_2, \dots, c_j\}$ е множеството от класове. Тълкуваме ω_k и c_j като множествата документи съответно в ω_k и c_j .

Лошото клъстериране има чистота близка до 0, а перфектното 1. Чистотата е сравнена с останалите 3 мерки в следната таблица:

	Purity	NMI	RI	F
Долна граница	0	0	0	0
Максимум	1	1	1	1
Стойност за фиг. 4	0.71	0.36	0.68	0.46

Таблица 2 Четирите външни критерия за оценка приложени върху клъстерирането на фиг. 4

Висока чистота е лесно да се постигне, когато броят на клъстери е много голям, в частност чистотата е 1, ако всеки документ има свой собствен клъстер. Това означава, че не можем да използваме тази мярка за компромис между качество и брой клъстери.

Мярка, която ни позволява такъв компромис е нормализираната споделена информация (НСИ/NMI):

$$(2) \quad NMI(\Omega, C) = \frac{I(\Omega, C)}{[H(\Omega) + H(C)]/2}$$

I е общата информация.

$$(3) \quad I(\Omega, C) = \sum_k \sum_j P(\omega_k \cap c_j) \log \frac{P(\omega_k \cap c_j)}{P(\omega_k)P(c_j)}$$

$$(4) \quad = \sum_k \sum_j \frac{|\omega_k \cap c_j|}{N} \log \frac{N(\omega_k \cap c_j)}{|\omega_k||c_j|}$$

където $P(\omega_k)$, $P(c_j)$ и $P(\omega_k \cap c_j)$ са вероятностите документ да бъде съответно в клъстер ω_k , клас c_j и в сечението на ω_k и c_j .

H е ентропията, както е дефинирана по-рано в книгата (глава 5):

$$(5) \quad H(\Omega) = - \sum_k P(\omega_k) \log P(\omega_k)$$

$$(6) \quad = - \sum_k \frac{|\omega_k|}{N} \log \frac{|\omega_k|}{N}$$

$I(\Omega, C)$ в уравнение (3) измерва количеството информация, с която нашето знание за класовете се увеличава, когато ни е казано какви са клъстери. Минимума на $I(\Omega, C)$ е 0, ако клъстерирането е произволна по отношение на класовата принадлежност. В този случай, да знаем, че документ е в определен клъстер, не ни дава никаква информация за неговата класова принадлежност. Максимална взаимна информация се постига за клъстериране Ω_{exact} , която перфектно пресъздава класовете, както и ако Ω_{exact} се раздели на по-малки клъстери. Понконкретно, клъстериране с $K = N$, където всеки документ има свой клъстер има максимална споделена информация. Така споделената информация има същия проблем като чистотата - не санкционира големия брой клъстери и по този начин не формализира нашето желание, при равни други условия да има по-малък брой клъстери.

Нормализирането чрез знаменател $[H(\Omega) + H(C)]/2$ в уравнение 2 решава проблема с големия брой клъстери, тъй като ентропията се увеличава с увеличаване броя на клъстери. Така например $H(\Omega)$ достига своя максимум $\log n$ за $K = N$ и така гарантира, че НСИ е ниска за $K = N$. Именно затова можем да използваме НСИ за сравняване на клъстериранятия с различен брой клъстери в тях. Точния вид на знаменателя е такъв, защото точната горна граница на $I(\Omega, C)$ е $[H(\Omega) + H(C)]/2$ и така НСИ е винаги число между 0 и 1.

Алтернатива на информационно-теоретичното тълкуване на клъстерирането е да се разглежда като поредица от решения, по едно за всяка от $N(N - 1)/2$ двойки документи в колекцията. Искаме да свържем 2 документа в един клъстер, само ако те са подобни. Вярно положително решение (true positive decision) разпределя 2 близки документа в един клъстер, а верните отрицателни (true negative) разделят 2 различни документа в различни клъстери. Могат да възникнат два типа грешки: фалшиво положителни решения (false positive) – свързват различни документи в общ клъстер, и фалшиво отрицателни (false negative) – разделя подобни документи в различни клъстери. При индекса на Ранд (RI) се измерва процентът на верните решения.

$$RI = \frac{TP + TN}{TP + FP + FN + TN}$$

Ще покажем как се изчислява ИР за примера от фиг. 4. Първо изчисляваме $TP + TN$. Трите клъстера съдържат 6, 6 и 5 точки, така че общият брой на „положителните“ (или иначе казано двойки, които са са в един клъстер) е:

$$TP + FP = (\frac{6}{2}) + (\frac{6}{2}) + (\frac{5}{2}) = 40$$

От тях x в кълстер 1, o в кълстер 2, \diamond в кълстер 3 и x -двойката в кълстер 3 са истински положителни.

$$TP = \left(\frac{5}{2}\right) + \left(\frac{4}{2}\right) + \left(\frac{3}{2}\right) + \left(\frac{2}{2}\right) = 20$$

Следователно $FP = 40 - 20 = 20$.

FN и TN се изчисляват аналогично и резултатите можем да запишем в следната таблица:

	Един кълстер	Различни кълстери
Един клас	$TP = 20$	$FN = 24$
Различни класове	$FP = 20$	$TN = 72$

И тогава $RI = (20 + 72)(20 + 20 + 24 + 72) \approx 0.68$.

Индексът на Ранд дава еднаква тежест на неверни положителни и неверни отрицателни резултати. Но разделянето на подобни документи е понякога по-лошо от поставянето на различни такива в един кълстер. Можем да използваме F-мярката за санкциониране на грешни отрицателни решения по-силно, отколкото грешни положителни, като изберем стойност $\beta > 1$, давайки на грешните отрицателни решения по-голяма тежест.

$$P = \frac{TP}{TP+FP} \quad R = \frac{TP}{TP+FN} \quad F_\beta = \frac{(\beta^2+1)}{\beta^2 P + R}$$

Въз основа на предната таблица, имаме: $P = \frac{20}{40} = 0.5$ и $R = 20/44 \approx 0.455$. Тогава имаме $F_1 \approx 0.48$ за $\beta = 1$ и $F_5 \approx 0.456$ за $\beta = 5$. В извличането на информация, оценките получени с F-мярката имат предимството, че тя е вече позната на научната общност.

16.5 K-Means (К-средни)

К-средни е най-важният алгоритъм за нейерархично(плоско) кълстериране. Неговата цел е да минимизира средното квадратично евклидово разстояние между документите и центъра на техния кълстер. Центърът на кълстера е дефиниран като средата или центроида $\vec{\mu}$ на документите в кълстера ω :

$$\vec{\mu}(\omega) = \frac{1}{|\omega|} \sum_{\vec{x} \in \omega} \vec{x}$$

Тази дефиниция предполага документите да бъдат представени като нормализирани вектори в реалното пространство по познатия начин. Центроидите тук играят роля, подобна на тази в Rocchio класификацията, разгледана в глава 14. Идеалният кълстер при *K-средни* е сфера, а неговият центроид е като център на гравитация на тази сфера. В идеалния случай кълстериите не трябва да се припокриват. Изискването за класовете при Rocchio класификацията е същото. Разликата е в това, че при кълстериране нямаме трениращо множество, за което да знаем кои документи трябва да бъдат в един и същи кълстер.

Мярка за това колко добре центроидите представят членовете на своите кълстери е *остатъчната сума от квадратите* или RSS - това е сумата от квадратите на разстоянията между всеки от векторите и неговия центроид.

$$RSS_k = \sum_{\vec{x} \in \omega_k} |\vec{x} - \vec{\mu}(\omega_k)|^2$$

$$(7) \quad RSS = \sum_{k=1}^K RSS_k$$

RSS е целевата функция на алгоритъма К-средни и нашата цел е да я минимизираме. Тъй като K е фиксирано, минимизирането на RSS е еквивалентно на минимизиране на средното квадратично разстояние - мярка за това колко добре центроидите представят своите документи.

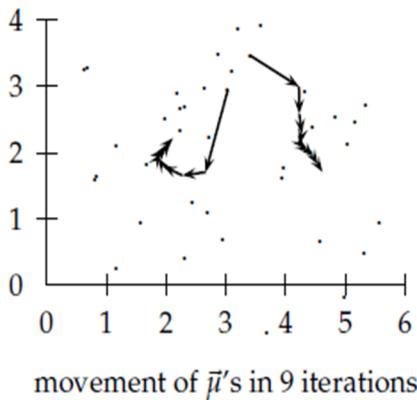
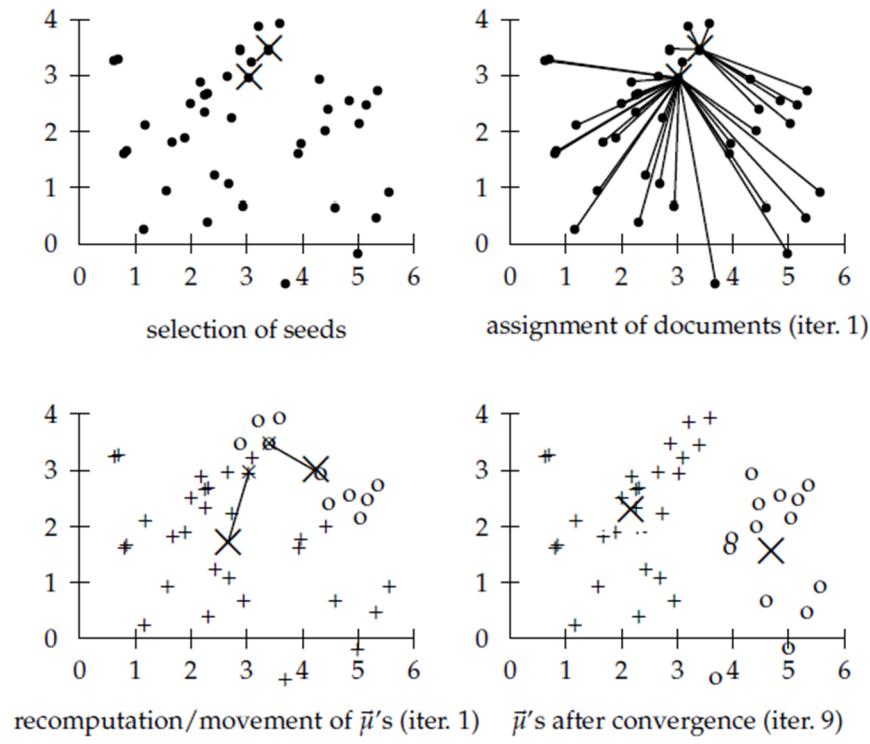
Нека разгледаме алгоритъма:

K-MEANS($\{\vec{x}_1, \dots, \vec{x}_N\}, K$)

```

1  $(\vec{s}_1, \vec{s}_2, \dots, \vec{s}_K) \leftarrow SelectRandomSeeds(\{\vec{x}_1, \dots, \vec{x}_N\}, K)$ 
2 fork  $\leftarrow 1 \text{to} K$ 
3 do  $\vec{\mu}_k \leftarrow \vec{s}_k$ 
4 while stopping criterion has not been met
5 do  $fork \leftarrow 1 \text{to} K$ 
6 do  $\omega_k \leftarrow \{\}$ 
7 for  $n \leftarrow 1 \text{to} N$ 
8 do  $j \leftarrow argmin_{j'} |\vec{\mu}_{j'} - \vec{x}_n|$ 
9  $\omega_j \leftarrow \omega_j \cup \{\vec{x}_n\}$  (reassignment of vectors)
10 fork  $\leftarrow 1 \text{to} K$ 
11 do  $\vec{\mu}_k \leftarrow \frac{1}{|\omega_k|} \sum_{\vec{x} \in \omega_k} \vec{x}$  (recomputation of centroids)
12 return  $\{\vec{\mu}_1, \dots, \vec{\mu}_K\}$ 
```

Първата стъпка от алгоритъма K -средни е да изберем като начални центрове на кълстери K на брой случаини документи, наречени „семена” (seeds). След това алгоритъмът премества центровете наоколо в пространството, така че да минимизира RSS . Както се вижда, това се прави итеративно, като се повтарят две стъпки, докато критерият за завършване не бъде изпълнен: документите се разпределят в кълстера с най-близък центроид и всеки центроид се определя отново, на базата на текущите членове на неговия кълстер. На фиг. 5 по-долу е даден пример, онагледяващ промяната на центроидите и кълсторите при девет итерации на алгоритъма K -средни за множество от точки в двумерното пространство, където броя на кълсторите е два.



Фиг. 5 Пример за алгоритъма K -средни при $K = 2$ в \mathbb{R}^2 . Позицията на двата центроида се проеменя едва забележимо след девет итерации.

За завършване на алгоритъма могат да бъдат приложени следните условия:

- Да бъдат завършени фиксиран брой I итерации. Това условие ограничава времето за изпълнение на кълстерирация алгоритъм, но в някои случаи качеството на кълстериране ще бъде ниско, тъй като не са били направени достатъчен брой итерации.
- Разпределението на документите по кълстерите да не се променя между итерациите. Така получаваме добро кълстериране, освен в случаите с лош локален минимум, но пък времето за изпълнение може да бъде неприемливо дълго.

- Центроидите да не се променят между итерациите. Това условие е еквивалентно на горното.
- Процесът да бъде прекратен, когато RSS падне под дадена гранична стойност. Това условие гарантира, че клъстерирането ще бъде с желаното качество след завършване на алгоритъма. В практиката се налага да комбинираме това условие с ограничение на броя на итерациите, за да гарантираме завършване.
- Процесът да бъде прекратен, когато стойността, с която RSS намалява, падне под дадена гранична стойност θ . За малки стойности на θ , това означава, че сме близо до търсения резултат. Това условие за завършване отново се налага да бъде комбинирано с условие за броя на итерациите, за да се предпазим от твърде дълго време за изпълнение.

Сега ще погажем, че алгоритъма K -средни дава все по-точен резултат след всяка итерация, доказвайки, че RSS монотонно намалява при всяка итерация. В тази секция ще използваме думата *намалява* в смисъла „*намалява или не се променя*“. Първо, RSS намалява при преразпределението на документите, тъй като всеки вектор се причислява към клъстера на най-близкия центроид, тъй че разстоянието, с което този вектор допринася за стойността на RSS намалява. Второ, RSS намалява и в стъпката за преизчисляване на центроидите, тъй като новия центроид е векторът \vec{v} , за който RSS_k достига своя минимум.

$$(8) \quad RSS_k(\vec{v}) = \sum_{\vec{x} \in \omega_k} |\vec{v} - \vec{x}|^2 = \sum_{\vec{x} \in \omega_k} \sum_{m=1}^M (v_m - x_m)^2$$

$$(9) \quad \frac{\partial RSS_k(\vec{v})}{\partial v_m} = \sum_{\vec{x} \in \omega_k} 2(v_m - x_m)$$

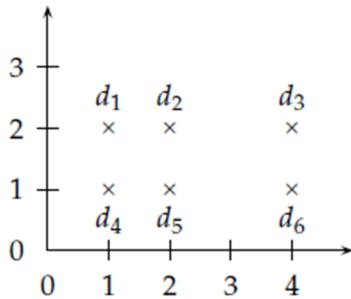
където x_m и v_m са m -тите компоненти на съответните вектори. Приравнявайки частната производна на нула, получаваме:

$$(10) \quad v_m = \frac{1}{|\omega_k|} \sum_{\vec{x} \in \omega_k} x_m,$$

което е покомпонентната дефиниция на центроид. По този начин минимизираме RSS_k като заменяме стария центроид с нов. Така RSS, сумата на RSS_k , трябва също да намалява по време на преизчисляването на центроидите.

Тъй като има само крайно множество възможни клъстерирации, то един монотонно намаляващ алгоритъм рано или късно ще достигне своя (локален) минимум.

Това доказва, че алгоритъма K -средни намира все по-добър резултат след всяка итерация, но за съжаление няма гаранция, че глобалния минимум на целевата функция ще бъде достигнат. Това е значителен проблем, ако множество от документи съдържа много документи, наречени *outliers* - документи, които са твърде далеч от всички останали и поради това не пасват добре на никой от клъстериите. Често, ако някой такъв документ бъде избран за начално “семе”, никой друг вектор не бива причислен към неговия клъстер при следващите итерации. Така получаваме клъстер-синглетон (клъстер само с един документ), въпреки че вероятно съществува клъстериране с по-нисък RSS. Фиг. 6 показва пример за неоптимално клъстериране в резултат на лош избор на начални „семена“.



Фиг. 6 Изходът от кълстерилизацията с алгоритъма K-средни зависи от началните „семена“. За семена d_2 и d_5 , K-средните достигат до резултат $\{d_1, d_2, d_3\}$, $\{d_4, d_5, d_6\}$, което е неоптимална кълстерилизация. За семена d_2 и d_3 достига до $\{d_1, d_2, d_4, d_5\}$, $\{d_3, d_6\}$, глобалния оптимум за $K = 2$

Друг тип неоптимално кълстериране, което се среща често е това, при което има налични празни кълстери.

Ефективни евристични подходи за избиране на семената са: 1. изключване на *outlier*-ите от множеството „семена“, 2. изprobване на няколко начални точки и избор на кълстериране с най-ниска стойност и 3. добиване на „семена“ чрез друг метод, например йерархично кълстериране. Тъй като детерминистичните методи за йерархичното кълстериране са по-предвидими от K-средните, то йерархично кълстериране на малка извадка с големина iK (например $i = 5$ или $i = 10$) често предоставя добри „семена“ (може да бъде разгледано описанието на Buckshot алгоритъма от глава 17).

Други методи за инициализация изчисляват „семена“, които не са сред векторите, които ще бъдат кълстериирани. Не много прецизен метод, който върши добра работа за голямо разнообразие от разпределения на документите, е този, при който се избрат i (например $i = 10$) случаини вектора за всеки кълстери и се използват техните центроиди за семена на съответните кълстери.

Каква е сложността по време на K-средни? Повечето време се използва за пресмятане на векторни разстояния. Една такава операция отнема $\Theta(M)$ време. При преизчисляване на центроидите се пресмятат KN разстояния, което прави общата сложност $\Theta(KNM)$. При преизчисляването всеки вектор еднократно бива добавен към центроид, така че сложността за тази стъпка е $\Theta(NM)$. Така за фиксиран брой итерации I общата сложност става $\Theta(IKNM)$. Това означава, че алгоритъма K-средни е линеен по отношение на всеки един от релевантните фактори: итерации, брой на кълстери, брой на векторите и измерност на пространството. Това означава, че K-средни е по-ефикасен алгоритъм от йерархичния, представен в глава 17. Трябва да фиксираме брой итерации, което може да бъде подвеждащо на практика, но пък в повечето случаи, алгоритъма K-средни бързо достига или до най-доброто възможно кълстериране или до друго, близко до него. Във втория случай няколко документа биха сменили принадлежността си, ако бъдат направени следващи итерации, но това има твърде малък ефект върху цялото качество на кълстериране.

Има обаче малка, но важна подробност свързана с предходното твърдение. Дори един линеен алгоритъм може да се окаже твърде бавен, ако един от аргументите на $\Theta(\dots)$ е голям, а M обикновено наистина е голям. Голямият размер не е проблем за изчисляване на разстоянието

между два документа. Техните вектори са силно „разредени” (sparse, има се предвид, че много от компонентите им са 0), така че само малък брой от теоретично възможните M на брой покомпонентни разлики трябва да бъдат изчислени. Центроидите, обаче, са „гъсти” (dense, има се предвид, че компонентите им не са нули), тъй като те обединяват всички термини, които се съдържат в кой да е от документите на техните клъстери. В резултат на това, пресмятанията на разстоянията отнемат време при наивна имплементация на K -средни. Все пак съществуват прости и ефективни евристики, чрез които близостта между центроид и документ се изчислява също толкова бързо, колкото и тази между два документа. Съкращаването на центроидите до най-значимите к термина (например $k = 1000$) почти не намалява качеството на клъстера, докато в същото време значително ускорява етапа на преразпределение на документите.

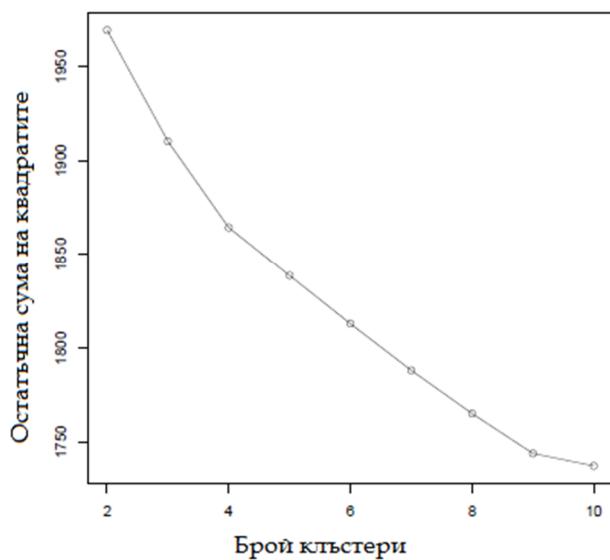
Същия проблем касаещ ефективността се среща и при K -medoids, разновидност на K -средни, която изчислява медоиди вместо центроиди като центрове на клъстери. Дефинираме медоида на един клъстер, като вектора на документ, който е най-близо до центроида. Тъй като медоидите са “разредени” документни вектори, то изчисляването на разстоянията е бързо.

16.5.1 Клъстерна кардиналност при алгоритъма K -средни

По-рано в този урок казахме, че броят на клъстери K е сред входните данни на повечето алгоритми за клъстериране. Но какво се случва, когато не можем да направим правдоподобно предположение за K ?

Един наивен подход би бил да изберем оптималната стойност за K спрямо целевата функция, а именно стойността на K , която минимизира RSS. Дефинирайки $RSS_{min}(K)$ като минималната RSS за всички клъстерирания с K клъстери, виждаме че $RSS_{min}(K)$ е монотонно намаляваща функция по K , която достига своя минимум 0 за $K = N$, където N е броят на документите. По този начин ще получим клъстериране, при което всеки документ е в свой собствен клъстер. Ясно е, че това изобщо не е оптимално клъстериране.

Един евристичен метод, който преодолява този проблем е да бъде пресметната $RSS_{min}(K)$ по следния начин: първо правим i (например $i = 10$) клъстерирания с K клъстери (всеки с различен избор на начални семена) и пресмятаме RSS за всяка една от тях. След това взимаме минимума на получените i на брой стойности на RSS. Означаваме този минимум с $\widehat{RSS}_{min}(K)$. По този начин можем да изследваме стойностите на $\widehat{RSS}_{min}(K)$ при увеличение на K и да открием „коляното“ в кривата - точката, за която последващото намаляване на \widehat{RSS}_{min} става забележимо малко. На фиг.7 например има две такива точки - при $K = 4$, където наклона слабо намалява и при $K = 9$, когато се наблюдава значително намаляване (при намаляване на наклона кривата все повече клони към хоризонтална). Така се налага да бъде въведено външно ограничение, за да бъде избрана една стойност за K от всички възможни (в случая избираме между 4 и 9).



Фиг. 7 Минималната остатъчна сума на квадратите, пресметната като функция на броя на кълстери в K-средните. В това кълстериране на 1203 Reuters-RCV1 документи има две точки, в които се наблюдава хоризонтиране на кривата на \overline{RSS}_{\min} : в точките за 4 и 9 кълстера. Тези документи са били избрани от категориите Китай, Германия, Русия и Спорт, така че кълстерирането при $K = 4$ е най-близкото до класификацията на Reuters.

Друг тип критерий за кардиналост на кълстери налага санкция за всеки нов кълстер. Концептуално започваме от единствен кълстер, съдържащ всички документи, а след това търсим оптимален брой кълстери K , като последвателно инкрементираме K с 1. За да определим кълстерираната кардиналност по този начин, създаваме обобщена целева функция, която комбинира два елемента: дисторция, мярка за това какво е отклонението между документите и прототипа на техните кълстери (например RSS за алгоритма K-средни) и мярка за сложността на модела. Тук интерпретираме кълстерирането като модел на данните. Сложността на модела при кълстерирането е обикновено броя на кълстери или тяхна функция. За K-средни имаме следния критерий за избор на K :

$$(11) \quad K = \arg \min K [RSS_{\min}(K) + \lambda K]$$

където λ е тегловен коефициент. Големи стойности на λ водят до решения с малък брой кълстери. За $\lambda = 0$ няма санкция за повече кълстери, а $K = N$ е най-доброто решение.

Видимата трудност при уравнение (11) е, че се налага да определим λ . Освен ако това не е задача, по-лесна от тази да определим директно K , то отново се намираме в начална позиция. В някои случаи можем да изберем стойности на λ , за които от предишни опит знаем, че са били подходящи за подобни множества от данни. Например, ако периодично кълстерираме статии от източник на новини, тогава е вероятно да има фиксирана стойност на λ , която да ни дава вярното K за всяко последващо кълстериране. В това приложение не бихме могли да определим K на базата на предишни опит, тъй като K се мени.

Теоретична обосновка на уравнение (11) е критерият *Akaike Information* (AIC) - информационно-теоретична мярка, която съпоставя (trades off) дисторция и сложност на модела. Общата форма на AIC е:

$$(12) \quad \text{AIC:} \quad K = \arg \min K [-2L(K) + 2q(K)]$$

където $L(K)$, отрицателния максимум на логаритмичната вероятност (log-likelihood) на данните за K клъстера, е мярка за дисторция, а $q(K)$, броя на параметрите на модел с K клъстера, е мярка за сложност на модела. Тук няма да се опитваме да извеждаме AIC, но е лесно критерия да бъде разбран интуитивно. Първото свойство на един добър модел на данните е, че всяка точка от данните е добре моделирана от модела. Това е целта на ниската дисторция. Но моделите също така трябва да бъдат малки (например с ниска сложност), тъй като модел, който просто описва данните (и следователно има нулема дисторция) е безсмислен. AIC предоставя теоретична обосновка на един определен начин за отегляване на тези два фактора (дисторция и сложност на модела), при избора на модел.

За K -средни AIC може да бъде представен по следния начин:

$$(13) \quad \text{AIC:} \quad K = \arg \min K [RSS_{min}(K) + 2MK]$$

Това уравнение е частен случай на уравнение (11) за $\lambda = 2M$.

За да се изведе уравнение (13) от уравнение (12) трябва да се обърне внимание, че $q(K) = KM$ в алгоритъма K -средни, тъй като всеки елемент от K на брой центроидите е параметър, който може да бъде променян независимо от останалите и че $L(K) = -(1/2)RSS_{min}(K)$.

Извеждането на AIC е базирано на няколко предположения. Едно от тях е, че данните са независими и идентично разпределени. Тези предположения са само приблизително верни за множествата от данни в извлечането на информация. В резултат AIC много рядко може да бъде приложен без модификация в текстовото клъстериране. На фиг. 7 измерността на векторното пространство е $M \approx 50\,000$. Тогава, условието $2MK > 5000$ е по-силно от по-малкото RSS-базирано условие $(\widehat{RSS}_{min}(1) < 5000)$, което не е показано на фигурата) и минимума на израза се достига за $K = 1$. Но както знаем, $K = 4$ (в съответствие с четирите класа Китай, Германия, Русия и Спорт) е по-добър избор от $K = 1$. В практиката, уравнение (11) е често по-полезно от уравнение (13) - със забележката, че трябва да открием приблизителна стойност за λ .

16.6 Клъстериране базирано на модели

В тази секция ще бъде описано обобщение на алгоритъма K -средни - алгоритъма ЕМ. Той може да бъде приложен върху по-голямо разнообразие от представяния и разпределения на документи отколкото K -средни.

В алгоритъма K -средни се опитваме да открием центроиди, които са добри представители. Можем да гледаме на множеството от K центроиди като на модел, който генерира данни. Генерирането на документи в този модел се състои в първоначален избор на центроиди на случаен принцип и след това добавяне на някои „шумове”. Ако шума е нормално разпределен, то тази процедура ще даде като резултат клъстери със сферична форма. *Базираното на модели клъстериране* предполага, че данните са били генериирани от модел и опитва да възстанови

оригиналния модел от данните. Моделът, който ще бъде възстановен от данните, дефинира клъстерите и разпределението на документите по клъстерите.

Често използван критерий за определяне на параметрите на модела е максималната вероятност. При K -средни, количеството $\exp(-\text{RSS})$ е пропорционално на вероятността да едърен модел (например множество от центроиди) да е генериран данните. За K -средни максимална вероятност и минимална RSS са еквивалентни критерии. Означаваме параметрите на модела с Θ . При K -средни $\Theta = \{\vec{\mu}_1, \dots, \vec{\mu}_K\}$.

По-общо, критерия за максимална вероятност изисква да изберем параметрите Θ , които максимилизират логаритмичната вероятност за генериране на данни D :

$$\Theta = \arg \max \Theta L(\Theta) = \arg \max \Theta \log \prod_{n=1}^N P(\Theta) = \arg \max \Theta \sum_{n=1}^N \log P(\Theta)$$

$L(\Theta)$ е целевата функция, която измерва колко добро е клъстерирането. Ако имаме две клъстерирации с еднакъв брой клъстери, то за предпочитане е този с по-висока $L(\Theta)$.

Това е същия подход, който е използван в глава 12 за моделиране на езици и в глава 13 за текстова класификация. При текстовата класификация се избира класа, който максимиизира вероятността за генериране на даден документ. Тук избираме клъстериране Θ , което максимиизира вероятността за генериране на дадено множество от документи. След като веднъж имаме Θ , можем да изчислим разпределителната вероятност $P(\omega_k; \Theta)$ за всяка двойка клъстера-документ. Това множество от разпределителни вероятности дефинира размито клъстериране.

Пример за размито разпределение е, че документ за китайски коли може да участва наполовина във всеки от клъстериите *Китай* и *Автомобили*, отразявайки факта, че и двете теми са уместни. Твърдо клъстериране като K -средни не може да моделира тази едновременна принадлежност към две теми.

Базираното на модели клъстериране предоставя рамка за обединяване на знанията ни за дадена област. K -средни и йерархичните алгоритми, разгледани в глава 17 правят сравнително „груби“ предположения за данните. Например, предполага се, че клъстери при K -средни са сфери. Базираното на модели клъстериране предлага много повече гъвкавост. Клъстерирация модел може да бъде адаптиран към това, което знаем за основното разпределение на данните, било то бернулиево, гаусово с несферично оклонение или друго.

Често използван алгоритъм за базирано на модели клъстериране е алгоритъмът *EM* (*Expectation-Maximization algorithm*). EM клъстериране е итеративен алгоритъм, който максимиизира $L(\Theta)$. EM може да бъде приложен към много различни типове вероятностни модели. Тук ще работим със смесица от многовариантни бернулиеви разпределения, разпределения, които са разгледани в глава 11 (секция 3) и глава 13 (секция 3):

$$(14) \quad P(\omega_k; \Theta) = (\prod_{t_m \in d} q_{mk})(\prod_{t_m \notin d} (1 - q_{mk}))$$

където $\Theta = \{\theta_1, \dots, \theta_K\}$, $\theta_k = (\alpha_k, q_{1k}, \dots, q_{Mk})$ и $q_{mk} = P(U_m = 1 | \omega_k)$ са параметрите на модела. $P(U_m = 1 | \omega_k)$ е вероятността документ от клъстера ω_k да съдържа терма t_m (U_m е случайна величина, приемаща стойност 1, ако термина t_m присъства в документа и 0 в

противен случай). Вероятността α_k е априорното вероятностно разпределение на ω_k : вероятността документа d да е в кълстера ω_k , в случай, че нямаме никаква информация за d .

Смесения модел тогава е:

$$(15) \quad P(\Theta) = \sum_{k=1}^K \alpha_k (\prod_{t_m \in d} q_{mk}) (\prod_{t_m \notin d} (1 - q_{mk}))$$

В този модел генерираме документ, като първо изберем кълстер k с вероятност α_k , а след това генерираме термините на документа според параметъра q_{mk} . Да си припомним, че представянето на документа с Бернулиевите опити на много променливи (multivariate Bernoulli) е вектор от M булеви стойности (а не вектор от реални числа).

Как използваме ЕМ, за да извлечем параметрите на кълстерирането от данните или как избираме параметрите Θ , за да максимираме $L(\Theta)$? ЕМ е близък до K -средни в това, че редува последователно стъпките „очакване“ (expectation step), отговаряща на преразпределението и стъпката „максимизация“ (maximization step), отговаряща на преизчисление на параметрите на модела. Параметрите на K -средни са центроидите, параметрите на случая на ЕМ, разглеждан в тази секция, са α_k и q_{mk} .

Максимиращата стъпка преизчислява условните параметри q_{mk} и приорните вероятности α_k по следния начин:

$$\text{Максимираща стъпка: } q_{mk} = \frac{\sum_{n=1}^N r_{nk} I(t_m \in d_n)}{\sum_{n=1}^N r_{nk}}, \alpha_k = \frac{\sum_{n=1}^N r_{nk}}{N}$$

където $I(t_m \in d_n) = 1$, ако $t_m \in d_n$ и 0 в противен случай, а r_{nk} е размито разпределение на документа d_n в кълстера k , както е било изчислено в предната итерация. Това са максималните вероятности, пресметнати за параметрите на Бернулиевото разпределение на много променливи от глава 13 (таблица 13.3), с разликата, че документите тук са частично разпределени по кълстериите.

Стъпката „очакване“ пресмята размито разпределение на документите по кълстериите по дадените параметри q_{mk} и α_k :

$$\text{Стъпка „Очакване“: } r_{nk} = \frac{\alpha_k (\prod_{t_m \in d_n} q_{mk}) (\prod_{t_m \notin d_n} (1 - q_{mk}))}{\sum_{k=1}^K \alpha_k (\prod_{t_m \in d_n} q_{mk}) (\prod_{t_m \notin d_n} (1 - q_{mk}))}$$

Тази стъпка прилага уравнения (14) и (15), за да изчисли вероятността документа d_n да е бил генериран от ω_k . Това е процедурата за класификация за Бернулиев опит на много променливи разглеждана в глава 13 (таблица 13.3). Така тази стъпка е просто Bernoulli Naive Bayes класификация.

Нека разгледаме следната таблица:

(a)	docID	document text	docID	document text
	1	hot chocolate cocoa beans	7	sweet sugar
	2	cocoa ghana africa	8	sugar cane brazil
	3	beans harvest ghana	9	sweet sugar beet
	4	cocoa butter	10	sweet cake icing
	5	butter truffles	11	cake black forest
	6	sweet chocolate		

(b)	Parameter	Iteration of clustering							
		0	1	2	3	4	5	15	25
	α_1	0.50	0.45	0.53	0.57	0.58	0.54	0.45	
	$r_{1,1}$	1.00	1.00	1.00	1.00	1.00	1.00	1.00	
	$r_{2,1}$	0.50	0.79	0.99	1.00	1.00	1.00	1.00	
	$r_{3,1}$	0.50	0.84	1.00	1.00	1.00	1.00	1.00	
	$r_{4,1}$	0.50	0.75	0.94	1.00	1.00	1.00	1.00	
	$r_{5,1}$	0.50	0.52	0.66	0.91	1.00	1.00	1.00	
	$r_{6,1}$	1.00	1.00	1.00	1.00	1.00	0.83	0.00	
	$r_{7,1}$	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	$r_{8,1}$	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	$r_{9,1}$	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	$r_{10,1}$	0.50	0.40	0.14	0.01	0.00	0.00	0.00	
	$r_{11,1}$	0.50	0.57	0.58	0.41	0.07	0.00	0.00	
	$q_{\text{africa},1}$	0.000	0.100	0.134	0.158	0.158	0.169	0.200	
	$q_{\text{africa},2}$	0.000	0.083	0.042	0.001	0.000	0.000	0.000	
	$q_{\text{brazil},1}$	0.000	0.000	0.000	0.000	0.000	0.000	0.000	
	$q_{\text{brazil},2}$	0.000	0.167	0.195	0.213	0.214	0.196	0.167	
	$q_{\text{cocoa},1}$	0.000	0.400	0.432	0.465	0.474	0.508	0.600	
	$q_{\text{cocoa},2}$	0.000	0.167	0.090	0.014	0.001	0.000	0.000	
	$q_{\text{sugar},1}$	0.000	0.000	0.000	0.000	0.000	0.000	0.000	
	$q_{\text{sugar},2}$	1.000	0.500	0.585	0.640	0.642	0.589	0.500	
	$q_{\text{sweet},1}$	1.000	0.300	0.238	0.180	0.159	0.153	0.000	
	$q_{\text{sweet},2}$	1.000	0.417	0.507	0.610	0.640	0.608	0.667	

Кълстерирано е множество от 11 документа в два кълстера с помощта на ЕМ. След достигане на търсения резултат при итерация 25, първите 5 документа са разпределени в кълстер 1 ($r_{i,1} = 1.00$), а последните 6 в кълстер 2 ($r_{i,1} = 0.00$). Нещо нетипично е, че крайното разпределение тук е строго. ЕМ обикновено дава като резултат размито разпределение. При итерация 25 априорната вероятност α_1 за кълстер 1 е $5/11 \approx 0.45$, тъй като 5 от 11те документа са в кълстер 1. Някои термини бързо са асоциирани с един кълстер, защото първоначалното разпределение може ясно да се „разпръсне“ по тях. Например, членството в кълстер 2 се разпространява от документ 7 към документ 8 на първата итерация, тъй като и двата документа споделят термина „sugar“ ($r_{8,1} = 0$ при итерация 1). За параметрите на термини, участващи в неясен контекст, разпределението в правилния кълстер отнема повече време. И двата семенни (seed) документа 6 и 7 съдържат „sweet“. В резултат - отнема 25 итерации термина да бъде ясно асоцииран с кълстер 2 ($q_{\text{sweet},1} = 0$ при итерация 25).

Намирането на добри семена е дори по-критично за ЕМ отколкото за K -средни. ЕМ е склонен да попадне в локален оптимум, ако семената не са добре подбрани. Това е основен проблем, който също се появява в други приложения на ЕМ. Така, също като при K -средни, началното разпределение на документите по кълстери често се изчислява от друг алгоритъм. Например, твърдо K -средно кълстериране може да предостави началното разпределение, което ЕМ после да „размие“.

Допълнителна литература, както и различни теоретични и практически упражнения могат да бъдат намерени в глава 16 на страници 24 ÷ 27.

17 Йерархично кълстериране

Плоското кълстериране е ефективно и концептуално просто, но то има редица недостатъци. Алгоритмите, които се използват при плоско кълстериране изискват предварително зададен брой на кълстери и са недетерминистични. Йерархичното кълстериране произвежда като изход йерархия – структура, която дава повече информация от неструктурiranето множества от кълстери, което произвежда плоското кълстериране. Също така, йерархичното кълстериране не изисква предварително задаване на броя на кълстерите. Тези преимущества, обаче идват на цената на повече изчисления. Най-често ползваните алгоритми за йерархично кълстериране имат сложност, която е поне квадратична спрямо броя на документите.

В този раздел първо е представено агломеративното йерархично кълстериране (Секция1) и представя четири различни агломеративни алгоритма в секции 2-4, които се различават в използваните мерки за подобие: единично свързване, пълно свързване, осреднени подобия и центроидно подобие. След това се обсъждат условията за оптималност на йерархичното кълстериране в Секция 5. Секция 6 представя разделящо йерархично кълстериране. Секция 7 разглежда автоматизирано етикиране на кълстери.

Има известни разлики между приложението на плоското и йерархичното кълстериране в извлечането на информация. Като цяло, плоското кълстериране е по-подходящо, когато ефикасността е важна, а йерархичното кълстериране, когато проявата на някой от потенциалните проблеми на плоското кълстериране (не добре структуриран резултат, предварително дефиниран брой кълстери, недетерминираност) може да окаже сериозен ефект на резултатите. Също така, много изследователи в областта са на мнение, че йерархичното кълстериране произвежда по-добри кълстери от плоското кълстериране. Въпреки това, не е постигнато съгласие относно въпроса.

17.1 1. Йерархично агломеративно кълстериране

Алгоритмите за йерархично кълстериране са два основни типа – агломеративни и разделящи. Алгоритмите за йерархично агломеративно кълстериране (ЙАК) инициализират отделен кълстер за всеки документ и последователно сливат двойки кълстери докато всички кълстери не се обединят в един кълстер съдържащ всички документи. Разделящите алгоритми работят в обратната посока – те започват от един кълстер съдържащ всички документи последователно разделят кълстери докато не достигнат състоянието, в което всеки документ е в отделен кълстер.

Агломеративното кълстериране обикновено се визуализира с *дендрограма*, където всяко сливане се представя с хоризонтална линия. Като у-координатата показва подобието на двата кълстера, които се сливат. Това подобие се нарича *комбинирано подобие* на получения кълстер. Подобието на кълстер с един документ е самоподобието на документа (1.0 в случая на косинусово подобие).

Дендрограмата позволява да се реконструира историята на сливанията, който са довели до това представяне.

Фундаментално предположение на ЙАК е, че сливането е монотонно. Монотонно значи, че ако s_1, s_2, \dots, s_{K-1} са комбинираните подобията на последователните сливания на ЙАК, то $s_1 \geq s_2 \geq \dots \geq s_{K-1}$. Немонотонно йерархично клъстериране съдържа поне една *инверсия* $s_i < s_{i+1}$ и противоречи на предположението, че се избира най-доброто сливане на всяка стъпка.

Йерархичното клъстериране не изиска предварително известен брой клъстери. В някои приложения обаче, целта е да се получат добре оформени клъстери, подобно на резултата на плоското клъстериране. В тези случаи, йерархията трябва да бъде „отрязана“ в дадена точка. Има редица критерии, с които тази точка може да бъде определена:

- Отсичане на предопределено ниво на подобие. Например, дендрограмата може да бъде отсечена на 0.4 ако целта е да се получат клъстери с минимално подобие 0.4.
- Отсичане там, където разликата между две последователни комбинирани подобия е най-голяма. Такива големи пространства могат да индикират „естествено“ клъстериране. Добавянето на нов клъстер значително намалява качеството на клъстерирането и следователно отсичането преди този рязък спад е силно желано.
- Прилагането на уравнението:

$$K = \arg \min_{K'} [RSS(K') + \lambda K'],$$

където K' описва отсичането на йерархията, което дава K' клъстера, RSS е остатъчната сума от квадратите, а λ е „наказание“ за всеки допълнителен клъстер. Вместо RSS , може да бъде използвана друга мярка за деформация.

- Подобно на плоското клъстериране, могат да се предопределят броя на клъстерите K и да се избере отсичането, което произвежда K клъстера.

Прост, наивен ЙАК е показан на Фигура 1. Първо се изчислява матрицата на подобие C с размерност $N \times N$. След това алгоритъмът извършва $N - 1$ сливащи стъпки. На всяка стъпка се сливат двата най-подобни клъстера и редовете и колоните на получения клъстер i се обновяват. Клъстерирането се запазва в списък от сливанията A . I съдържа списъка от клъстери, които все още са свободни за сливане. Функцията $SIM(i, m, j)$ пресмята подобието между клъстера j с клъстера получен при сливането на клъстери i и m . В някои ЙАК алгоритми SIM е просто функция от $C[i][j]$ и $C[j][m]$, например максимум от тези две стойности при единично свързване.

Алгоритъмът ще бъде подобрен за различните мерки за подобие – единично свързване и пълно свързване (Секция 2) и средно разстояние и центроидно подобие (секции 3 и 4). Критерийните за сливане на тези четири варианта на ЙАК са показани на Фигура 2.

```

SimpleHAC(d1, ..., dN)
for n ← 1 to N
    do for i ← 1 to N
        do C[n][i] ← SIM(dn, di)
I[n] ← 1
A ← []
for k ← 1 to N - 1
    do (i, m) ← arg max{(i, m): i != m & I[i] == 1 & I[m] == 1} C[i][m]
    A.Append((i, m))

```

```
for j ← 1 to N
```

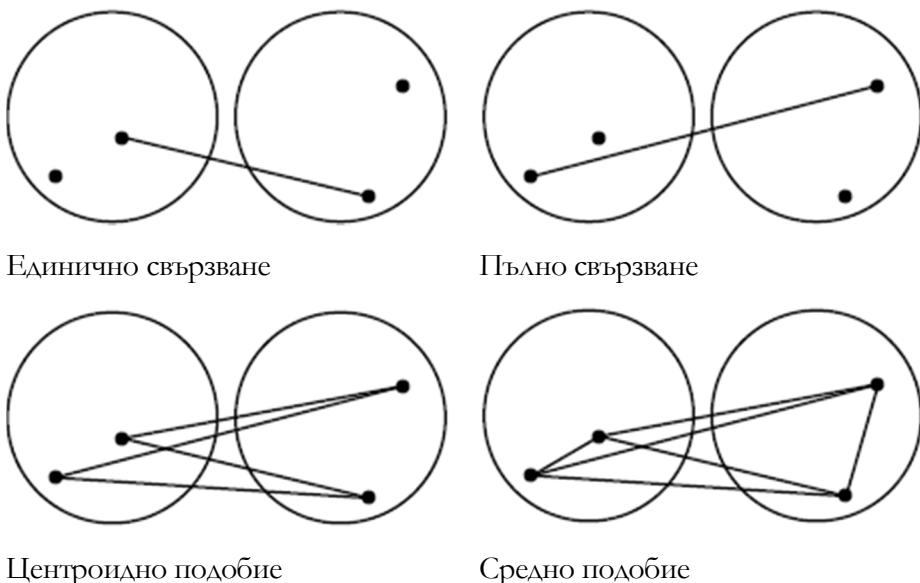
```
do
```

```
    C[i][j] ← SIM(i, m, j)  
    C[j][i] ← SIM(j, m, i)
```

```
I[m] ← 0
```

```
return A
```

Фигура 1. Прост, но неефективен ЙАК алгоритъм.



Фигура 2. Критерии за свързване при четирите варианта на ЙАК.

17.2.2. Клъстериране с единично свързване и пълно свързване

При *клъстерирането по метода на единичното свързване*, подобието на два клъстера е равно на подобието на двата им най-подобни член. Този критерий за сливане е локален. Под внимание се вземат само частите от двата клъстера, които са най-близо една до друга. По-далечните части от клъстера не се вземат под внимание.

При *клъстерирането по метода на пълното свързване*, подобието на два клъстера е равно на подобието на двата им най-различни член. Това е еквивалентно на избирането на двойката клъстери, които минимизират диаметъра на новосформирания клъстер. Тези критерии не е локален, тъй като решението за сливане могат да се повлияят от цялата структура на клъстерирането. Това довежда до явно изразено предпочтение към компактните клъстери с малък диаметър. Единствен документ, лежащ далеч от центъра може значително да увеличи диаметъра на кандидатите за нов клъстер и по този начин напълно да промени крайното клъстериране.

Единичното свързване и пълното свързване имат интерпретация в теория на графите. Ако дефинираме s_k да бъде комбинираното подобие на два клъстера слети на стъпка k , и $G(s_k)$ е графът, които свързва всички елементи с подобие поне s_k . Тогава клъстериите след стъпка k при единичното свързване са свързаните компоненти в $G(s_k)$ а клъстериите след стъпка k при пълното свързване са максималните клики на $G(s_k)$.

Именно от тези интерпретации произлизат термините *единично свързване* и *пълно свързване*. Кълстерите получени при единичното свързване на стъпка k са максималните множества от точки, които са свързани с поне една връзка на подобие $s \geq s_k$, а тези получени при пълно свързване са максималните множества от точки, които са напълно свързани с връзки на подобие $s \geq s_k$.

Единичното и пълното свързване ограничават оценяването на качеството на кълстери до едно подобие между двойка документи. Оценка базирана на една двойка не може напълно да отрази разпределението на документите в кълстера. Поради това не е изненадващо, че и двата алгоритъма често довеждат до нежелани кълстери. Единичното свързване произвежда рехави кълстери. Тъй като критерия за сливане е локален, се получава така наречения *ефект на веригата*, при който се получава верига от точки.

От друга страна, пълното свързване страда от друг проблем. При него се обръща прекалено голямо внимание на крайности – точки, които не пасват добре на глобалната структура на кълстера. Пълното свързване невинаги открива най-интуитивната структура от кълстери.

17.2.1 2.1. Сложност на ЙАК

Сложността на наивния ЙАК алгоритъм е $\Theta(N^3)$, тъй като на всяка от $N - 1$ итерации се провежда пълно изчерпване на матрицата C (с размери $N \times N$). И за четирите ЙАК методи, съществува по-ефикасен подход чрез използването на приоритетна опашка. Сложността на тази модификация на алгоритъма е $\Theta(N^2 \log N)$. Редовете на матрицата на подобие се сортират в намаляващ ред в приоритетните опашки P . Тогава $P[k].MAX()$ връща кълстера в $P[k]$, който има най-високо подобие с ω_k (където ω_k е k -тия кълстер). След сливането на ω_{k1} и ω_{k2} , за техен представител се използва ω_{k1} . Функцията SIM изчислява подобието за потенциалните двойки, които ще бъдат слети: най-високо подобие за единично свързване, най-ниско за пълно свързване, средно подобие за агломеративно кълстериране чрез наосреднени подобия (АКОП) и центроидно подобие за центроидно кълстериране (Секция 4).

Процедурата EfficientHAC приема като вход множество от вектори (в отличие от множество от документи), тъй като АКОП и центроидното кълстериране изискват вектори като вход. Версията на EfficientHAC за пълно свързване може да бъде използвана и над документи.

```

EfficientHAC(d1, ..., dN)
for n ← 1 to N
    do for i ← 1 to N
        do
            C[n][i].sim ← dn.di
            C[n][i].index ← i
        I[n] ← 1
        P[n] ← priority queue for C[n] sorted on sim
        P[n].Delete(C[n][n])
        A ← []
        for k ← 1 to N - 1
            if C[n][I[n]].sim >= C[n][I[n + 1]].sim then
                I[n + 1] ← I[n]
                C[n][I[n + 1]].sim ← C[n][I[n]].sim
                C[n][I[n + 1]].index ← I[n]
            end if
        end for
    end for
end for

```

```

do
     $k_1 \leftarrow \arg \max_{\{k: I[k] == 1\}} P[k].MAX().sim$ 
     $k_2 \leftarrow P[k_1].MAX().index$ 
    A.Append(( $k_1, k_2$ ))
     $I[k_2] \leftarrow 0$ 
     $P[k_1] \leftarrow []$ 
    for each  $i$  with  $I[i] == 1 \& i != k_1$ 
    do
         $P[i].Delete(C[i][k_1])$ 
         $P[i].Delete(C[i][k_2])$ 
         $C[i][k_1].sim \leftarrow SIM(i, k_1, k_2)$ 
         $P[i].Insert(C[i][k_1])$ 
         $C[k_1][i].sim \leftarrow SIM(i, k_1, k_2)$ 
         $P[k_1].Insert(C[k_1][i])$ 
    return A

```

Като допълнителна оптимизация на единичното свързване се използва масив за следващото най-добро свързване. Този масив пази информация за най-доброто свързване за всеки един кълъстер. Така сложността на алгоритъма достига $\Theta(N^2)$.

```

SingleLinkClustering( $d_1, \dots, d_N$ )
    for  $n \leftarrow 1$  to  $N$ 
        do for  $i \leftarrow 1$  to  $N$ 
            do
                 $C[n][i].sim \leftarrow SIM(d_n, d_i)$ 
                 $C[n][i].index \leftarrow i$ 
                 $I[n] \leftarrow n$ 
                 $NBM[n] \leftarrow \arg \max_{X \in \{C[i][j]: I[j] == i \& j != i\}} X.sim$ 
                 $A \leftarrow []$ 
                for  $n \leftarrow 1$  to  $N - 1$ 
                    do
                         $i1 \leftarrow \arg \max \{i: I[i] = i\} NBM[i].sim$ 
                         $i2 \leftarrow I[NBM[i1].index]$ 
                        A.Append(( $i1, i2$ ))
                        for  $i \leftarrow 1$  to  $N$ 
                            do
                                if  $I[i] == i1 \& i != i1 \& i != i2$ 
                                then  $C[i1][i].sim \leftarrow C[i][i1].sim \leftarrow \max(C[i1][i].sim, C[i2][i].sim)$ 
                                if  $I[i] == i2$ 
                                then  $I[i] \leftarrow i1$ 
                                 $NBM[i1] \leftarrow \arg \max_{X \in \{C[i][j]: I[j] == i \& j != i1\}} X.sim$ 
    return A

```

Възможно ли е тази оптимизация да се приложи и на другите ЙАК алгоритми? Не е възможно, защото само единичното свързване е мярка, която се запазва при най-доброто сливане. Например ако за кълстер ω_k най-доброто сливане при единично свързване е ω_b , тогава, след като слеем ω_i с кълстер ω_p , получения кълстер пак ще бъде най-доброто сливане за ω_k . Тази закономерност обаче не е вярна за другите три ЙАК алгоритъма.

17.3 3. Агломеративно кълстериране чрез осреднени подобия

Агломеративното кълстериране чрез осреднени подобия (АКОП) изчислява качествата на кълстер използвайки всичките подобия между документите, като по този начин избягва проблемите на единичното и пълното свързване, които приравняват кълстера подобие до подобие между една двойка документи. АКОП изчислява средното подобие SIM-GA на всички двойки документи, включително документи от един и същ кълстер, но самоподобията не се включват в крайната оценка:

$$SIMGA(\omega_i, \omega_j) = \frac{1}{(N_i + N_j)(N_i + N_j - 1)} \sum_{d_m \in \omega_i \cup \omega_j} \sum_{d_m \in \omega_i \cup \omega_j, d_n \neq d_m} \overrightarrow{d_m} \cdot \overrightarrow{d_n}$$

където векторите са нормализирани, N_i и N_j са броя на документите в кълстера ω_i и ω_j . Целта на АКОП е при сливането на два кълстера да получим съгласуван кълстер. За да се оцени съгласуваността трябва да се разгледат всички елементи от новия кълстер.

SIM-GA може да се пресметне по-ефективно като се вземе предвид, че сумата от индивидуалните подобия на векторите е равна на подобията на техните суми:

$$\sum_{d_m \in \omega_i \cup \omega_j} \sum_{d_m \in \omega_i \cup \omega_j, d_n \neq d_m} \overrightarrow{d_m} \cdot \overrightarrow{d_n} = \left(\sum_{d_m \in \omega_i} d_m \right) \cdot \left(\sum_{d_n \in \omega_j} d_n \right)$$

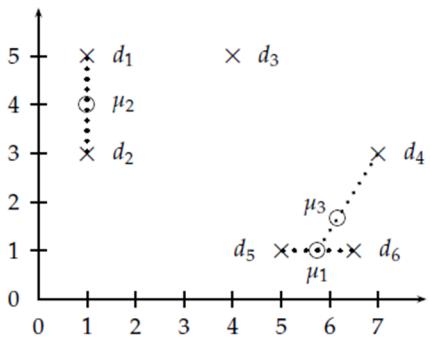
Следователно:

$$SIMGA(\omega_i, \omega_j) = \frac{1}{(N_i + N_j)(N_i + N_j - 1)} \left[\left(\sum_{d_m \in \omega_i \cup \omega_j} \overrightarrow{d_m} \right)^2 - (N_i + N_j) \right]$$

Термът $N_i + N_j$ се изважда, за да се отчетат самоподобията със стойност 1.0. По този начин подобието на два кълстера може да бъде изчислено за константно време (приемайки, че разполагаме с векторните суми за всеки кълстер). Това позволява функцията *SIM* в *EfficientHAC* да бъде изчислена за константно време.

Важно е да се отбележи, че скаларното произведение е дистрибутивно относно събирането. Това позволява за ефективното изчисление на АКОП. Поради това, този метод не може да бъде приложен за представяния на документи различни от вектор от реални числа. Това е фундаментална разлика между единично и пълно свързване и АКОП. Докато единичното и пълното свързване разчитат само на матрица на подобия, без да има значение как тези подобия са изчислени, АКОП разчита на скаларното произведение.

Накратко АКОП изисква документите да са представени като вектори, векторите да бъдат нормализирани, така че самоподобията да са 1.0 и скаларното произведение да е мярката за подобие между вектори и суми на вектори.



Фигура3. Три итерации на центроидно кълстериране. Всяка итерация слива тези два кълстера с най-близки центроиди.

17.4 4. Центроидно кълстериране

При центроидното кълстериране, подобието между два кълстера се дефинира като подобието между техните центроиди по следния начин:

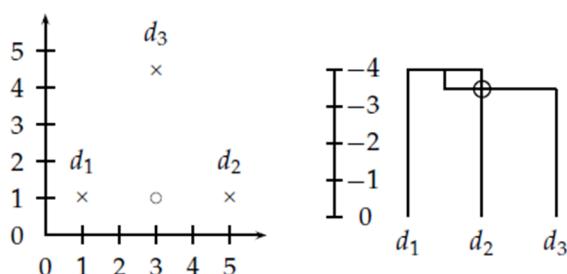
$$(1) \text{ SIM - CENT}(\omega_i, \omega_j) = \vec{\mu}(\omega_i) \cdot \vec{\mu}(\omega_j) = \\ = \left(\frac{1}{N_i} \sum_{d_m \in \omega_i} \vec{d}_m \right) \cdot \left(\frac{1}{N_j} \sum_{d_n \in \omega_j} \vec{d}_n \right) = \\ (2) = \left(\frac{1}{N_i N_j} \sum_{d_m \in \omega_i} \sum_{d_n \in \omega_j} \vec{d}_n \vec{d}_m \right)$$

Уравнение (1) дефинира мярка за подобието между центроиди. Уравнение (2) показва, че центроидното подобие е еквивалентно на средното подобие между всички двойки документи от различни кълстери. Така разликата между АКОП и центроидното кълстериране се състои в това, че при едното всички двойки документи се взимат предвид, а при другото само двойки от различни кълстери.

Фигура 3 илюстрира първите три стъпки на центроидното кълстериране. Първите две итерации образуват кълстери $\{d_5, d_6\}$ с центроид μ_1 и $\{d_1, d_2\}$ с центроид μ_2 , тъй като двойките (d_5, d_6) и (d_1, d_2) имат най-високо центроидно подобие. В третата итерация, най-високото центроидно подобие е между μ_1 и d_4 и поради това се образува кълстер $\{d_4, d_5, d_6\}$ с центроид μ_3 .

Също както АКОП, центроидното кълстериране не е запазва най-доброто сливане и затова е със сложност $\Theta(N^2 \log N)$.

За разлика от другите три ЙАКалгоритъма, центроидното кълстериране не е монотонно. Възможно е да се получат, така наречени *инверсии*: Подобието може да се



Фигура4. Центроидното кълстериране не е монотонно. Документите $d_1(1 + \epsilon, 1)$, $d_2(5, 1)$, и $d_3(3, 1 + 2\sqrt{3})$ са почти равноотдалечени, като d_1 и d_2 са по близки едно до друго отколкото до d_3 . Немонотонната инверсия при йерархичното кълстериране на трите точки изглежда като пресичаща линия на сливане в дендрограмата. Пресечната точка е оградена.

увеличава както е показано в примера на Фигура 4, където подобието е дефинирано като отрицателно разстояние. При първото сливане, подобието между d_1 и d_2 е $-(4 - \epsilon)$. Във второто сливане подобието между центроида на d_1 и d_2 (къргът) и d_3 е $\approx -\cos(\frac{\pi}{6}) \times 4 = -\frac{\sqrt{3}}{2} \times 4 \approx -3.46 > -(4 - \epsilon)$. Това е пример за инверсия: подобието нараства в тези две кълстериращи стъпки. В един монотонен ЙАК алгоритъм, подобието е монотонно намаляващо между последователни итерации.

Нарастващото подобие в поредица от ЙАК кълстериращи стъпки противоречи на фундаменталното допускане, че малките кълстери са по-съгласувани от големите. Инверсия в дендрограмата се изобразява като хоризонтална линия на сливане, която е по-ниска от предишната линия на сливане. Всички линии на сливане във Фигури 1 и 5 са по-високи от техните предшестващи такива, тъй като методите за единично и пълно свързване са монотонни.

Въпреки немонотонността си, центроидното кълстериране се използва често заради мярката на подобие, която използва – тя е концептуално по-проста от тази използвана в АКОП. Фигура 3 е напълно достатъчна за запознаване с центроидно кълстериране. Няма еквивалентно опростена графика, която да обясни как точно работи АКОП.

17.5 5. Оптималност на ЙАК

За да се определят точно оптималните условия за йерархично кълстериране първо трябва да се дефинира комбинирано подобие на кълстериране $\Omega = \{\omega_1, \dots, \omega_K\}$ като най-малкото комбинирано подобие на някой от неговите K кълстера:

$$COMB - SIM(\{\omega_1, \dots, \omega_K\}) = \min_k COMB - SIM(\omega_k)$$

По-горе дефинираме $\Omega = \{\omega_1, \dots, \omega_K\}$ да е оптимално ако ако всички Ω' с k кълстера, $k \leq K$, имат по-малки комбинирани подобия:

$$|\Omega'| \leq |\Omega| COMB - SIM(\Omega') \leq COMB - SIM(\Omega)$$

Фигура 4 показва, че центроидното кълстериране не е оптимално. Кълстерирането $\{\{d_1, d_2\}, \{d_3\}\}$ (за $K=2$) има комбинирано подобие $-(4 - \epsilon)$ и $\{\{d_1, d_2, d_3\}\}$ (за $K=1$) има комбинирано подобие -3.46 . Така кълстерирането $\{\{d_1, d_2\}, \{d_3\}\}$ не е оптимално, тъй като съществува кълстериране с по-малко кълстери $\{\{d_1, d_2, d_3\}\}$ с по-високо комбинирано подобие. Центроидното кълстериране не е оптимално, поради възможността да има инверсии.

По-горната дефиниция за оптималност не би била достатъчно полезна ако е единствено приложима върху кълстериране ако е известна историята му на сливане. Може, обаче, да се покаже, че комбинираното подобие за трите алгоритми, в които няма инверсия, може да бъде изведеното от някой кълстер без да е известна историята му. Тези дефиниции за комбинирано подобие се дават както следва:

Метод на единичното свързване	Комбинираното подобие на кълстер ω е най-малкото подобие на някое негово двуразделяне, като подобието на едно двуразделяне е най-голямото подобие между двойка документи от различна част: $COMB - SIM(\omega) = \min_{\{\omega': \omega' \subset \omega\}} \max_{d_i \in \omega'} \max_{d_j \in \omega - \omega'} SIM(d_i, d_j)$
Метод на пълното свързване	Комбинираното подобие на кълстер ω е най-малкото подобие между двойка точки в $\omega: \min_{d_i \in \omega} \min_{d_j \in \omega} SIM(d_i, d_j)$
АКОП	Комбинираното подобие на кълстер ω е средното от всички подобия между двойки различни кълстери: Уравнение (3).

Ако използваме тези дефиниции за комбинирано подобие, тогава оптималността може да се разглежда като свойство на множество от кълстери, а не на процеса, чрез който то е образувано.

Сега вече може да се докаже, че методът на единично свързване е оптимален, като се използва индукция за броя кълстери K . Ще бъде разгледано доказателство за случая, в който няма двойки документи с еднакво подобие, но то лесно може да бъде разширено за случая, в който има.

Индукционна база: Кълстериране с $K = N$ кълстери има комбинирано подобие 1.0 и това е най-голямата стойност за комбинирано подобие.

Индукционна хипотеза: Нека Ω_K е кълстериране получено по метода на единичното свързване. Допускаме, че Ω_K е оптимално, т.е. $COMB - SIM(\Omega_K) \geq COMB - SIM(\Omega'_K)$, за всички Ω'_K .

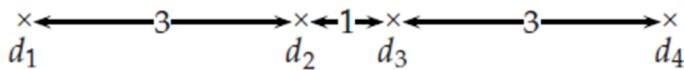
Индукционна стъпка: Нека допуснем, че Ω_{K-1} , получено след сливането на двета най-подобни кълстера в Ω_K , не е оптимално, и че различна поредица от слиивания Ω'_K, Ω'_{K-1} води до оптималното кълстериране с $K-1$ кълстера. Допускането, че Ω'_{K-1} е оптимално и че Ω_{K-1} не е, може да бъде записано като $COMB - SIM(\Omega'_{K-1}) > COMB - SIM(\Omega_{K-1})$.

Случай 1: Двата документа свързани от $s = COMB - SIM(\Omega'_{K-1})$ са в същия кълстер в Ω_K . Те може да са в един и същ кълстер ако сливане с подобие по-малко от s е било извършено в поредицата от слиивания, чрез които се е получило Ω_K . Така имаме $s > \text{КОМБ} - \text{ПОДОБ}(\Omega_K)$, и следователно $COMB - SIM(\Omega'_{K-1}) > COMB - SIM(\Omega_K) > COMB - SIM(\Omega'_K) > COMB - SIM(\Omega'_{K-1})$, което е противоречие.

Случай 2: Двата документа свързани от $s = COMB - SIM(\Omega'_{K-1})$ не са в същия кълстер в Ω_K . Имаме $s = COMB - SIM(\Omega'_{K-1}) > COMB - SIM(\Omega_{K-1})$ следователно по правилото за сливане тези два различни кълстера е трябвало да бъдат слети и отново получаваме противоречие.

Така доказваме, че Ω_{K-1} е оптимално.

За разлика от метода на единичното свързване, методът на пълното свързване и АКОП не са оптимални. Това може да се види в следния пример:



И двата метода първо сливат двете точки с разстояние 1 (d_2 и d_3) и не могат да намерят кълстерирането с $K=2$ $\{\{d_1, d_2\}, \{d_3, d_4\}\}$, което е оптимално.

Въпреки това, критериите за сливане на двата метода удовлетворяват желанието за приблизителна сферичност по-добре от метода на единично свързване. Много често се желаят сферични кълстери. Така метода на единичното свързване е оптимален, но спрямо неправилните критерии желани в повечето приложения за кълстериране на документи.

Метод	Комбинирано подобие	Сложност	Оптимален?	Забележка
Единично свързване	Максимално подобие между всички двойки документи в различни кълстери	$\Theta(N^2)$	Да	Ефектът на веригата
Пълно свързване	Минимално подобие между всички двойки документи в различни кълстери	$\Theta(N^2 \log N)$	Не	Чувствителен към крайности
Средно подобие	Средно на всички подобия	$\Theta(N^2 \log N)$	Не	Най-добър избор
Центроид	Средно на всички подобия между различни кълстери	$\Theta(N^2 \log N)$	Не	Може да има инверсии.

Таблица 1. Сравнения на ЙАК алгоритмите.

Таблица 1 обобщава свойствата на четирите ЙАК метода описани по-горе. АКОПсе препоръчва за използване, тъй като създава кълстерирания с най-добри свойства.

17.6 6. Разделящо кълстериране

Досега беше разгледано само агломеративно кълстериране, но йерархия от кълстери може да бъде създадена и отгоре-надолу. Този вариант на йерархично кълстериране се нарича разделящо кълстериране. Започваме отгоре с всички документи събрани в един кълстер. След това кълстерът се разделя използвайки алгоритъм за плоско кълстериране. Тази процедура се изпълнява рекурсивно, докато всеки документ не стои в свой собствен кълстер.

Кълстерирането отгоре-надолу е концептуално по-сложно от това отдолу-нагоре, тъй като използва допълнителен плосък алгоритъм. Въпреки това има предимството да бъде по-бърз ако не се генерира цялата йерархия до най-долния и слой. За фиксиран брой нива, използвайки алгоритъм като K-means, алгоритмите отгоре-надолу са с линейна сложност по броя на документи и кълстери и работят доста по бързо от ЙАК алгоритмите, които са с поне квадратична сложност.

17.7.7. Етикиране на клъстери

В много от приложението на плоско или йерархично клъстериране, особено там, където е намесен потребителски интерфейс, се налага потребители да използват клъстери по един или друг начин. В такива ситуации е полезно всеки клъстер да има етикет, който го описва.

Разграничаващото етикиране избира етикети за клъстери като сравнява разпределението на термини в един клъстер, с това в останалите. Най-често използван е методът за обща информация, който идентифицира такива етикети, които характеризират даден клъстер спрямо останалите. Комбинация от разграничаващ тест с негативни оценки за рядко срещани термини най-често дава най-добри резултати, тъй като рядко срещаните термини в повечето случаи не дават информация за клъстера като цяло.

Вътрешно-клъстърното етикиране избира етикети въз основа на самия клъстер без да се интересува от останалите. Един такъв метод е използване на заглавието на документа, най-близък до центроида на клъстера, за етикет. Заглавията по-лесно се четат и възприемат от поредица термини и също така могат да съдържат важна информация, която е иначе игнорирана при създаването на списъка с термини.

Брой документи	Метод за етикиране:	Обща информация	Заглавие
4	622	Oil plant mexico production crude power 000 refinery gas bpd	Plant oil production barrels crude bpd mexicodolly capacity petroleum MEXICO: Hurricane Dolly heads for Mexico coast
9	1017	Police security Russian people military peace killed told grozny court	Police killed military security peace told troops forces rebels people RUSSIA: Russia's Lebed meets rebel chief in Chechnya
10	1259	00 000 tonnes traders futures wheat prices cents September tonne	Delivery traders futures tonne tonnes desk wheat prices 000 00 USA: Export Business – Grain/oilseeds complex

Таблица 2. Автоматично избрани етикети за клъстери. Използван е K-means алгоритъм приложен върху три от десет клъстера (4, 9 и 10) за първите 10000 документа взети от Ройтерс.

Друг вътрешно клъстърен метод използва списък от термини с високи тегла в центроида на клъстера като етикет. Такива термини често са по-описателни за клъстера отколкото заглавия, но по-трудно се възприемат от потребителите. В Таблица 2 са дадени няколко примера за гореизброените методи.

18 Разлагане на матрици и латентно семантично индексиране

Разлагането на матрица представлява изразяването ѝ чрез произведение на матрици, получени от собствените ѝ вектори.

Теорема за матричната диагонализация:

Нека A е квадратна, реална матрица с n реда и колони, и n линейно независими собствени вектори. Съществува разлагане по собствени вектори

$$A = B^*C^*B^{-1},$$

където колоните на B са собствените вектори на A , а C е диагонална матрица, чийто стойности са собствените стойности на A , в намаляващ ред:

$$\begin{aligned} &c_{11} \\ &c_{22} \\ &\dots \\ &c_{nn} \end{aligned}$$

като $c_{ii} \geq c_{i+1,i+1}$. Ако собствените стойности са различни, разлагането е уникално.

$B = (b_1 \ b_2 \ \dots \ b_n)$, където b са колони, и са собствените вектори на A .

Така че $A^*B = A^*(b_1 \ b_2 \ \dots \ b_n) = (c_{11}b_1 \ c_{22}b_2 \ \dots \ c_{nn}b_n)$, следователно $A^*B = B^*C$, и съответно $A = B^*C^*B^{-1}$.

Теорема за симетричната диагонализация:

Нека S е квадратна, симетрична $M \times M$ матрица от реални стойности, с M линейно независими собствени вектора. Тогава съществува симетрично диагонално разлагане:

$$S = Q^*C^*Q^T,$$

където колоните на Q са ортогоналните и нормализираните собствени вектори на S , и C е диагоналната матрица, чиито стойности са собствените стойности на S . Освен това всички стойности на Q са реални и имаме:

$$Q^{-1} = Q^T$$

Използвайки разлагането по единична стойност можем да получим приближение на матрица за термини в документ, под формата на матрица от по-нисък ранк. Това приближение предлага ново представяне на всеки документ, представен в матрицата. Към това представяне можем да подаваме заявки, посредством които можем да изчислим рейтинзи на сходност между документите и заявките. Именито този процес се нарича латентно семантично индексиране.

Представянето на документи и заявки чрез векторно пространство има най-различни предства. Такива са еднаквото третиране на заявки и документи като вектори, индуцираният резултат изчислен на базата на косинусово сходство, различната оценка за тежестта на различните термини, и приложимостта му отвъд извлечането на документи, например при

кълстеризиране и класификация. То, за сметка на това, не може да се справи с два много често срещани проблема при естествените езици - синонимите и многозначността. Както знаем, синонимите са две различни думи, носещи еднакво значение. Във векторното пространство предоставя различно измерение за всеки от група синоними, въпреки че те означават едно и също, като по този начин не улавя отношението между двете думи. В последствие, изчисленото сходство $q \cdot d$ между заявката q (дума), и документ d съдържащ дума, както и нейни синоними, няма да отразява истинската близост между тези синоними, която иначе е лесно забележима за всеки човек. При многозначността пък, изчисленото сходство пък ще бъде прекалено високо, тъй като ще бъде надчислено за различните смисъли на думата, въпреки че се е търсено сходството само за един.

Дори при набор от документи със скромен размер, матрицата за термини в документите най-вероятно би имала десетки хиляди редове и колони, и ранк в същия диапазон. При латентното семантично индексиране, наричано още латентен семантичен анализ, се използва разлагането по единична стойност за да се конструира ниско-ранково приближение на матрицата, чийто ранк е далеч по нисък от оригиналния. Всеки ред и колона, съответстващи на дума и документ, се обозначават на k -измерно пространство, където k е ранкът на приближението. Това пространство е дефинирано от k основни собствени вектора, съответстващи на най-големите собствени стойности, на CC^T и C^TC , където C е матрицата на термините в документите. Матрицата C_k (приближението), е независима от k .

След това използваме новото ЛСИ представяне точно както първото - за изчисление на сходства между вектори. Векторът заявка q се разполага в представянето си в ЛСИ пространството чрез преобразованието

$$\vec{q}_k = \Sigma_k^{-1} U_k^T \vec{q}. \quad (1)$$

Косинусовото сходство може да се използва, за да се сравни сходството между заявка и документ, два документа или две думи. Да се има предвид, че уравнението (1) по никакъв начин не е повлияно от това че q е заявка, а от това че е вектор в пространството с думите. Това означава, че ако имаме ЛСИ представяне на колекция документи, нов документ който не е в колекцията може да бъде "вмъкнат" към представянето посредством уравнение (1). Така имаме възможността постепенно да добавяме документи към ЛСИ представянето. При постепенното добавяне не се отчитат появите на словосъчетания в новите документи (дори игнорира новите думи, които съдържат). По този начин с добавянето на все повече нови документи ЛСИ представянето деградира като дори може да се наложи да бъде прекомпилирано.

Точността на приближението на C_k към C ни кара да се надяваме, че относителните стойности на косинусовите прилики са запазени: ако заявка е близка до документ в оригиналното пространство, тя остава сравнително близо в k -измерното пространство. Но това само по себе си не е интересно, особено като се има предвид, че накъсаният вектор q на заявката се сгъстява в ниско-измерното пространство. Това е значителна изчислителна цена, сравнена с цената за пропедиране на q в естествената му форма.

Пример. Нека имаме матрицата за термини в документите $C =$

		d_1	d_2	d_3	d_4	d_5	d_6	
	ship	1	0	1	0	0	0	
	boat	0	1	0	0	0	0	
	ocean	1	1	0	0	0	0	
	voyage	1	0	0	1	1	0	
	trip	0	0	0	1	0	1	

Нейното единствено число разлагане е съвкупността на трите матрици по-долу. Първо имаме U която е:

		1	2	3	4	5	
	ship	-0.44 —	-0.30 —	0.57 —	0.58 —	0.25 —	
	boat	-0.13 —	-0.33 —	-0.59 —	0.00	0.73	
	ocean	-0.48 —	-0.51 —	-0.37 —	0.00	-0.61 —	
	voyage	-0.70 —	0.35	0.15	-0.58 —	0.16	
	trip	-0.26 —	0.65	-0.41 —	0.58	-0.09 —	

Когато прилагаме разлагане по единична стойност към матрицата за термини в документите, U се нарича разложена по единична стойност матрица за термините. Единичните стойности са $\Sigma =$

2.16	0.00	0.00	0.00	0.00
0.00	1.59	0.00	0.00	0.00
0.00	0.00	1.28	0.00	0.00

0.00	0.00	0.00	1.00	0.00
0.00	0.00	0.00	0.00	0.39

На края имаме V^T , която се нарича разложена по единична стойност матрица за документите:

		d_1	d_2	d_3	d_4	d_5	d_6	
	1	-0.75 —	-0.28 —	-0.20 —	-0.45 —	-0.33 —	-0.12 —	
	2	-0.29 —	-0.53 —	-0.19 —	0.63	0.22	0.41	
	3	0.28	-0.75 —	0.45	-0.20 —	0.12	-0.33 —	
	4	0.00	0.00	0.58	0.00	-0.58 —	0.58	
	5	-0.53 —	0.29	0.63	0.19	0.41	-0.22 —	

Чрез зануляването на всички стойности освен 2-те най-големи на \sum , получаваме $\sum_2 =$

2.16	0.00	0.00	0.00	0.00
0.00	1.59	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00

От тук получаваме $C_2 =$

	d_1	d_2	d_3	d_4	d_5	d_6	
1	-1.62 —	-0.60 —	-0.44 —	-0.97 —	-0.70 —	-0.26 —	

2	-0.46 —	-0.84 —	-0.30 —	1.00	0.35	0.65	
3	0.00	0.00	0.00	0.00	0.00	0.00	
4	0.00	0.00	0.00	0.00	0.00	0.00	
5	0.00	0.00	0.00	0.00	0.00	0.00	

19 Основи на търсенето в Уеб

(Web search basics)

19.1 Произход и история

Мрежата (the Web) е уникална по своя размер, по почти пълната липса на координация в създаването ѝ и разнообразието на произхода и мотивите на участниците ѝ. Ето защото мрежата се различава значително от традиционното търсене в документи. Създаването на хипертекстът (the hypertext) през 40-те години предшества създаването на World Wide Web или просто the Web през 1990г. От тогава насам уеб използването се е разрастало до толкова, че в момента голяма част от човечеството я използва. Базира се на прост, отворен клиент-сървър дизайн: сървърт си комуникира с клиента чрез http протокол, като асинхронно пренася разнообразно съдържание (текст, изображения, медиа-аудио и видео файлове), което е закодирано в HTML (hypertext markup language); клиентът, обикновено е браузър, приложение с графична потребителска среда. Той игнорира това, което не разбира.

Основната операция, която се извършва, е следната: клиентът изпраща http заявка до уеб сървъра. Браузърът специфицира URL (Universal Resource Locator/Универсален локатор на ресурси), примерно <http://www.stanford.edu/atoz/contact.html>. Където низът http се отнася до протокола за пренасяне на данните, www.stanford.edu се нарича домейн и се явява корен на йерархия от уеб страници, *atoz/contact.html* е път в тази йерархия, а *contact.html* е файлът, който съдържа информацията, която трябва да бъде върната като отговор от сървъра.

Първите браузъри са предоставяли възможност да се виждат таговете в съдържанието на URL, което позволило на новите потребители да създават свое HTML съдържание без обширно обучение или опит. Благодарение на друга черта на браузара- игнориране на това, което не разбира, потребителите могат свободно да експериментират и да се учат от новосъздадените уеб страници, без да се притесняват, че грешка в синтаксиса ще срине системата. По този начин мрежата бързо се превърнала в най-добрият начин за снабдяване и консумиране на информация.

Всичко това би било безмислено, ако богатството от информация не е откриваемо и достъпно от други потребители. Първите опити за извлечане на информация от мрежат се разделят на две категории: 1) търсещи машини, които индексират целия текст, като Altavista, Excite и Infoseek и 2) таксономии съдържащи уеб страници по категории като Yahoo!. Първите предоставят на потребителя интерфейс за търсене по ключови думи, базиран на инвертиран индекс и механизми за оценяване. Вторите позволяват на потребителя да търси в йерархично дърво от категории.

Първото поколение уеб търсещи машини използвало традиционните методи за търсене, изправяйки се пред предизвикателството с размерността на мрежата. Ранните уеб търсещи машини е трявало да се справят с индексиране на съдържанието на десетки милиони документи. Особеността на създаване на съдържание в мрежата е наложило създаването на нови техники за оценяване и справяне със спама, които да осигурят качество и релевантност на върнатите резултати.

19.2 Уеб характеристики

Основните характеристики, довели до експлозивното нарастване на мрежата – децентрализирано публикуване на съдържание без почти никакъв контрол на авторството – се оказаха най-голямого предизвикателство за уеб търсещите машини. Създалите на уеб страници създават съдържание на десетки (естествени) езици и диалекти, изпитвайки необходимост от различни форми на „подрязване” и други лингвистични операции.

Съществуват два вида уеб страници: статични и динамични. 2 Anchor A B

Статични уеб страници са страници, чието съдържание не се променя от една заявка към страницата до друга. Пример за статична страница е личната страница на преподавател в училище или университет, който всекидневно я актуализира, докато страница със статусите на полетите на дадено летище е динамична уеб страница. *Динамичните страници* са механично генериирани от приложен сървър като отговор на заявка към база от данни.

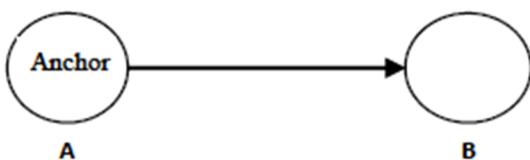
19.3 Уеб граф

Може да разглеждаме статичния уеб, състоящ се от статични HTML страници заедно с хипервръзките между тях, като ориентиран граф, в който всяка уеб страница е връх, а всяка хипервръзка ориентирано ребро.

На Фигура 1 са показани 2 върха A и B от уеб граф, всеки отговарящ на определена уеб страница с хипервръзка от A до B.

Фигура 1 Уеб граф

Фигура 1 Уеб граф



Има три основни категории уеб страници, които понякога са наричани: IN, OUT и SCC (Strongly Connected Components). Един уеб потребител може да премине от която и да е страница от IN, до която и да е страница от SCC, следвайки хипервръзките. По същия начин потребителят може да премине от страница от SCC, до която и да е страница от OUT. И накрая потребителят може да премине от която и да е страница от SCC до всяка друга страница от SCC, но не е възможно да премине от страница от SCC, до която и да е страница от IN, или от страница от OUT до страница от SCC.

Според някои проучвания IN и OUT са почти еднакви по размер, докато SCC са по-големи. Повечето уеб страници попадат в едно от тези три множества. Останалите страници формират *търби*, които са малки множества от страници извън SCC, които водят директно от IN до OUT или *мустачета*, които или водят от IN до никъде или от никъде до OUT.

19.4 Спам

Още от самото създаване на уеб пространството ясно, че търсещите машини са важно средство за свързване на рекламирани с потенциални клиенти. Потребител, който търси *Mayi golf недвижими имоти*, едва ли търси новини или забавления свързани с голф курсове, по-скоро се интересува от закупуване на такъв имот. Продавачите на такива имоти и техните агенти са стимулирани да създават уеб страница, която се оценява високо спрямо тази заявка. В търсеща машина, чието оценяване се базира на честотата на срещане на даден термин, уеб страница с множество повторения на *Mayi golf недвижими имоти* би се оценила високо. Това е довело до появата на първото поколение спам, което (в контекста на търсене в мрежата) е манипулиране

на съдържанието на уеб страница с цел появяването ѝ в началото на резултата от търсенето по избрани клечови думи. За да избегнат дразнене на потребителя с повторенията на думите, спамарите прибегнали до оцветяване и сливане на термините със същия цвят като този на фона. Въпреки че тези думи са невидими за човек, индексаторът на търсещата машина ще изведи невидимите думи от HTML представянето на уеб страницата и ще ги индексира като налични в тази страница.

Много създаватели на уеб съдържание имат търговски цели и затова са склонни да печелят от манипулиране на резултатите от търсенето. При някои търсещи машини е възможно чрез заплащане да се дадена страница да бъде включена в индекса на търсещата машина - този модел е известен като *заплатено включване*. Различните търсещи машини имат различни политики относно заплатеното включване и дали такова включване да се отрази в оценяването на резултатите от търсенето.

Скоро търсещите машини станали достатъчно добри в засичнето на спам при сканиране на голям брой повторения на определени клечови думи, но спамърите отговорили с по-богато множество от техники. Една от тях е *покриването (cloaking)*. Уеб сървърът на спамърите връща различни страници в зависимост от това дали http заявката идва от *паж* или от потребителски браузър. Първият случай води до индексиране на страницата от търсещата машина с подвеждащи думи. Когато потребителят търси с тези клечови думи и избере да прегледа страницата, той отваря уеб страница с различно съдържание от тази, която е била индексирана.

Doorway страница съдържа текст и метаданни внимателно подбрани да бъдат високо оценени при избрани клечови думи. Когато браузърът извика *doorway страница*, той е препратен към страница с рекламно съдържание. По-напреднала спамърска техника включва манипулиране на метаданните на страницата, включително и линковете в нея.

Като се има предвид, че спаменето е икономически мотивирана дейност, покрай нея се разви индустрия за Оптимизиране на търсещи машини (Search Engine Optimizers - SEO). SEO предлагат консултантски услуги на клиенти, които се стремят уеб страниците им да бъдат високо оценени спрямо избрани клечови думи.

За борба със спамърите, които манипулират съдържанието на уеб страниците, се използва структурата от връзки на мрежата – техника известна като *анализ на връзките*. Първата търсеща машина, която приложи анализ на връзките върху голямо съдържание, е Google като в момента повечето търсещи машини използват тази техника.

19.5 Рекалмирането като икономически модел

В ранната история на мрежата, компаниите използвали графични реклами банери в уеб страниците на популярни уеб сайтове. Основната цел на този вид реклама било промотиране на марката (*branding*): предаване на позитивно чувство на потребителя относно марката на компанията, поставила рекламата.

Обикновено такива реклами датели се таксуват с *цена за хиляда (cost per mil – CPM)*: цената, която трябва да плати компанията за показване на банера ѝ 1000 пъти. Някои уеб сайтове сключват договори с реклами датели, в които реклами датата не се заплаща спрямо броя на показванията ѝ, а спрямо броя на потребителите, които са кликали върху нея. Този модел е известен като *цена за*

кликоване (*cost per click* – CPC). В този случай чрез кликване върху рекламиата, потребителят е препратен към страница подготвена от рекламодателя, където потребителят се подтиква към транзакция.

Пionер в тази област е компания наречена *Goto*. Не може да се каже, че в традиционния смисъл *Goto* е търсеща машина; по-скоро за всеки търсен термин q , тя приема цена за наддаване от компании, които искат техните страници да бъдат показани при заявка q . Като отговор на заявката q *Goto* ще върне всички страници на рекламодатели, които са наддавали за q , подредени по цени. Освен това, ако потребителят кликне върху някой от върнатите резултати, съответният рекламодател ще заплати на *Goto*.

Няколко аспекта от *Goto* модела заслужават внимание. Първо, потребител, който задава заявка q в интерфейса за търсене на *Goto* активно показва интерес и намерение към тази заявка. Второ, *Goto* получава заплащане само, ако потребителят наистина изрази интерес към рекламиата – кликне върху нея. Взети заедно тези две неща са създали мощн механизъм за свързване на рекламодатели с потребителите. Този тип търсене е станало известно като *спонсорирано търсене* или *рекламиране чрез търсене*.

Настоящите търсещи машини следват този модел: те предоставят чисти резултати (*алгоритмични резултати*) като основен отговор на заявката на потребителя, заедно със спонсирирани резултати, които се извеждат отделно и отдясно на алгоритмичните резултати.

Икономическите мотиви, които стоят под спонсираното търсене, карат някои участници да обърнат системата в тяхна полза. Това може да се прояви под много форми, една от които е *click fraud*. За момента няма универсално прието определение за *click fraud*. Може да се дефинира като кликвания върху спонсирирани резултати, които не са извършени от истински потребител. Например, нечестен рекламодател може да се опита да изчерпи бюджета за реклама на своя конкурент, чрез многократно кликане върху неговия спонсиран резултат. Търсещите машини са изправени пред предизвикателството да различат кои от кликванията, които наблюдават са част от *click fraud*, за да избегнат таксуване на клиентите си за такива кликвания.

19.6 The search user experience. Потребителско търсене

Много е важно да разберем потребителите, които търсят в мрежата. За разлика от традиционното извличане на информация, където потребителите са професионалисти със знания за формирането на заявки върху добре-структурниана колекция, мрежовите потребители не притежават знания (или не се интересуват) относно хетерогенността на съдържанието в мрежата, синтаксисът на езичите за заявки и изкуството за създаване на заявка. В действителност търсенето в мрежата не трябва да изисква такива знания от потребителите. Различни проучвания са установили, че средният брой на ключовите думи при търсене в мрежата е между 2 и 3, като рядко се използват синтактични оператори (логически оператори, маски и др.).

Колкото повече потребителски трафик привлича една търсеща машина, толкова повече доход може да спечели от спонсираното търсене. Но по какво се различават търсещите машини и как генерират потребителски трафик? Google откри 2 принципа, които й помогнаха да наделее над конкуренцията: 1) фокус върху релевантността (прецизност на върнатите резултати) и 2)

улеснено потребителско изживявяне, т.е както страницата за заявки, така и върнатите резултати са подредени и почти изцяло текстови, с много малко графични елементи.

19.7 Потребителски нужди (User query needs)

Може да разделим заявките за търсене в мрежата в три категории: 1) информационни; 2) навигационни; 3) транзакционни. Някои заявки могат да попаднат в повече от една категория, докато други в никој една.

Информационните заявки търсят основна информация върху обширна тема, като Образование или България. Обикновено не същества уеб страница, в която да се съдържа цялата търсена информация. Потребители, които задават информационни заявки, се опитват да асимилират информацията от няколко уеб страници.

Навигационните заявки търсят уеб страници или главни страници на една единствена същност, която потребителите имат в предвид, като например БългарияЕр. В този случай потребителят очаква, че първият върнат резултат ще бъде главната страница на БългарияЕр.

Транзакционната заявка е прелюдия към осъществяването на транзакция от страна на потребител – като закупуване на даден продукт, изтегляне на файл или резервация. В този случай резултатите от търсения трябва да предоставят списък от услуги с форма за извършване на транзакцията.

19.8 Индексен размер и оценяване

На пръв поглед, изчерпателността на търсения в интернет расте с размера на индексираните web страници. Практически обаче е невъзможно да бъдат индексирани всички те, защото техният брой е безкраен. Например, адрес от вида http://www.yahoo.com/any_string, като *any_string* е произволена комбинация от символи, винаги ще върна валидна HTML страница, която информира потребителя, че това е грешен адрес в *Yahoo!*. Това е само един пример как web сървърите могат да генерират безкраен борй валидни web pages, които представляват истинска паяжина-капан за търсещите машини.

Индексите включват множество класове на страници, поради което няма ясна и еднозначна мярка за тяхното измерване. Това все още е актуален проблем за областта, въпреки че са измислени голем брой техники за грубо оценяване на радиуса на индексния размер на две търсещи машини (E_1 и E_2). Основанта хипотеза, на която се основават тези техники е, че всяка една търсеща машина по отделно индексира част от **Web пространството, избрана независимо и неформално на случаен принцип**. Приемайки това твърдение, можем да използваме един класически метод за оценяване на индексния размер познат като ***capture-recapture method***.

Нека предположим, че можем да отделим случайна страница от индекса на търсеща машина E_1 , можем да проверим дали тя принадлежи на индекса на търсещата машина E_2 и обратното. Тези експерименти биха ни дали две променливи x и y , като x представлява оценката на това, кои страници от индекса на E_1 принадлежат на индекса на E_2 , докато y – обратното. Т.е.:

$$x|E_1| \approx y|E_2|,$$

от където получаваме следната формула:

$$\frac{|E_1|}{|E_2|} \approx \frac{y}{x}.$$

За да се имплементира този сценарии, бихме могли да създадем произволна страница в Web пространството и да тестваме дали присъства във всички търсещи машини. За съжаление, отделянето на произволна страница е трудна задача, но са възможни няколко подхода за осъществяването ѝ (вземайки предвид допустимото отклонение на всеки от тях). Такива са:

- Random searches;
- Random IP addresses;
- Random walks;
- Random queries.

На кратко ще разгледаме последния от списъка (**Random queries**) поради две причини: той все по-успешно се използва в редица специализирани оценки и обратното, той е подходът, при който най-често се допускат грешки в имплементацията, водещи до заблуждаваща оценка на индексирането. Неговата идея е да отдели произволна страница от индекс на търсеща машина изпращайки произволна заявка към нея. Важно е да се отбележи, че този подход би могъл да се имплементира най-успешно при търсенето на термини, които не се срещат в стандартните (разговорни) речници, но за сметка на това са често използвани в по-стеснен кръг от ресурси. За да пристъпим към разрешаване на проблема е нужно да натрупаме примерен web речник. Това може да бъде осъществено чрез crawl-ването на ограничена част от web пространството, например *Yahoo!* Използваме произволна съединителна заявка (*conjunctive query*) върху търсещата машина E_1 и отделяме от нея първите 100 резултата като произволна страница \hat{p} . След това проверяваме за наличие на \hat{p} в търсещата машина E_2 чрез избиране на 6-8 рядко срещащи се термина в \hat{p} и ги използваме в *conjunctive query* за E_2 . Можем да подобрим оценката като повторим този експеримент голям брой пъти, като проблемите, които се срещат са следните:

нашият пример е предубеден към по-дълги документи;

отделятнето на първите 100 резултата от E_1 убеждават в отклонение от ранкинг алгоритъма на E_1 , докато отделянето на всички резултати от E_1 би направило експеримента много бавен. Това е така, защото повечето уеб търсещи машини предоставят защита срещу прекомерните автоматизирани заявки;

по време на етапа на проверка, се въвеждат допълнителни отклонения. Например E_2 не може да отговори правилно на *conjunctive query* съдържащата 8 думи;

нито E_1 , нито E_2 може да отговорят на тестовите запитвания, третирайки ги като автоматизиран спам, а не като добросъвестни запитвания;

Проведени са редица изследвания върху тази основна парадигма, за да се премахнат някои от тези проблеми. За момента все още не е намерено идеалното решение, но равнището на сложност в статистката за разбиране на отклоненията значително се е увеличило. Също така, новите експерименти използват и други методи за сравнение освен произволни заявки, като например *document random walk sampling*, в който документът се избира от произволна „разходка“ из виртуалния граф на изходните документи. В този граф, възлите представляват документи, които са свързани със стрелки ако съдържат две или повече общи думи.

19.9 Близки дубликати и shingling

Един от аспектите на размера на индексираните страници, който не сме разгледали до момента е *дубликатите*. Интернет пространството съдържа множество копия на едно и също съдържание, като според някои проучвания то достига до 40%. Някои от тези копия, разбира се са легитимни, например целиящи да подсигурят достъпа до съответна информация. Въпреки това търсещите машини се опитват да избегнат тяхното индексиране за да се намали използването на излишно пространство и допълнителните процеси за обработка на информацията.

Най-простият подход за разпознаване на дубликатите е чрез изчисляване за всяка една страница на така наречените „пръстови отпечатъци“ (*fingerprint*), които представляват сбито 64 битово представяне на символите от конкретната страница. Когато пръстовите отпечатъци на две страници са еднакви се проверява дали съдържанието на самите страници е еднакво, т.e. дали страниците са дубликати. Недостатъкът на пръстовите отпечатъци е, че не успява да се справи с така наречения феномен „блезки дубликати“ (*near duplication*). В много случаи съдържанието на две страници се различава единствено по няколко символа, например датата на публикуване или модификацията, като за тях бихме искали да сметнем че са еднакви.

Решението за справяне с дублицираните интернет страници е техника, наречена *shingling*. При нея имаме положително число k и последователност от термини в документа d . Например да разгледаме следния текст: „*a rose is a rose is a rose*“. 4-те думи в този текст ($k = 4$) образуват последователността от термини на текста: „*a rose is a*“, „*rose is a rose*“ и „*is a rose is*“. Първите два термина се срещат два пъти в текста, съответно два документа биха били близки дубликати ако термините в него са образувани от същите 4 думи.

Изразено математически, нека $S(d)$ представя набора от думи в документа d . Прилагайки коефицианта на Жакард (*Jaccard*), който измерва степента на припокриване между $S(d1)$ и $S(d2)$ като

$$|S(d_1) \cap S(d_2)| / |S(d_1) \cup S(d_2)|$$

с означение:

$$J(S(d_1), S(d_2))$$

Нашият тест за близки дубликати между $d1$ и $d2$ ще изчисли коефициента на Жакард. Ако той превишава прагът от 0,9 то може да приемем, че двете страници са близки дубликати и да премахнем едната от индекса.

19.10 Речник:

статична уеб страница – страница, чието съдържание не се променя от една заявка към страницата до друга;

динамичнича страница – механично генериирани страници от приложен сървър като отговор на заявка към база от данни;

in-links (входящи връзки) – връзки към дадена страница;

out-links (изходящи връзки) – връзки от дадена страница;

спам – манипулиране на съдържанието на уеб страница с цел появяването ѝ в началото на резултата от търсения по избрани ключови думи;

заплащено включване – чрез заплащане дадена страница се включва в индекса на търсещата машина;

покриване (cloaking) – техника използвана от спамарите, при която уеб сървърът връща различни страници в зависимост от това дали http заявката идва от пак или от потребителски браузър;

doorway страница – страница, която съдържа текст и метадани внимателно подбрани да бъдат високо оценени при избрани ключови думи;

алгоритмични резултати – чисти резултати, отговор от търсения на заявката на потребителя, не включват спонсорирани резултати;

click фрат – кликвания върху спонсорирани резултати, които не са извършени от истински потребители;

информационна заявка – заявка, която търси основна информация върху обширна тема;

навигационна заявка – заявка, която търси уеб страници или главни страници на една единствена същност;

транзакционна заявка – прелюдия към осъществяването на транзакция от страна на потребителя;

пръстови отпечатъци (fingerprint) – сбито 64 битово представяне на символите от конкретната страница.

20 Обхождане и индексиране на мрежата

(web crawling and indexing)

20.1 Уеб роботи

Обхождането на уеб страници (web crawling) е процес, чрез който събираме страници от мрежата (интернет), за да ги индексираме и да ги използваме като база за машина за търсене. Целта на обхождането е бързо и ефективно да съберем колкото е възможно повече полезни уеб страници заедно със структурата на хипервръзките, които ги свързват. В глава 19 показахме сложността на мрежата, която произтича от съдържанието, което милиони некоординирани индивиди създават. В тази глава обсъждаме трудностите, които произтичат от това за обхождането на мрежата. Фокусът тук е компонентът показан на фиг. 19.7 – обхождащ уеб робот, който понякога се нарича паяк (crawler).

В главата не се цели да се обясни как да се създаде такъв робот за пълномащабна търсеща машина с търговска цел. Вместо това, се фокусираме върху набор от проблеми, които са присъщи на обхождането на ниво от студентски проект до сериозни научни изследвания. Започваме с изброяване на необходимите свойства, които един обхождащ робот трябва да притежава, като се спирате на всяко свойство отделно. Остатъкът от главата описва архитектурата и някои детайли по реализацията на робот за обхождане, който работи разпределено върху много машини и притежава съответните свойства. В последната част от главата обсъждаме разпределението на индекси между много машини в рамките на реализация, предназначена да работи в пълния мащаб на мрежата.

Тук ще изброим свойствата за обхождащи мрежата роботи в две категории: свойства, които роботите трябва да притежават и такива, които е препоръчително да притежават.

20.1.1 Свойства, които уеб роботите трябва да притежават

Устойчивост: Мрежата съдържа сървъри, които създават клопки за роботите, генерирайки уеб страници, които ги поставят в ситуация да обхождат безкраен брой страници в даден домейн. Роботите трябва да бъдат проектирани така, че да бъдат устойчиви на подобни клопки. Не всички капани са поставени със злонамерена цел, някои се получават като следствие от неправилната разработка на сайтове.

Съобразителност (Учивост): Уеб сървърите имат както неявни, така и явни политики, регулиращи колко често даден обхождащ мрежата робот може да ги посещава. Тези политики трябва да бъдат уважавани.

20.1.2 Свойства, които уеб роботите е препоръчително да притежават

Разпределеност: Роботът трябва да има способността да се изпълнява едновременно на много машини, възползвайки се от по-високата пропускателна способност.

Мащабируемост: Архитектурата на робота трябва да позволява повишаване на бързодействието му, чрез добавяне на допълнителни машини и увеличаване на пропускателната способност.

Бързодействие и ефективност: Работът трябва да използва ефективно системни ресурси, като процесор, твърд диск и мрежова пропускателна способност.

Качество: Като се има предвид, че значителна част от страниците в мрежата са с твърде ниска полезност, за да могат да отговарят на потребителски заявки, работите трябва да обхождат първо страниците с висока полезност.

Актуалност: В много приложения, работът трябва да действа в режим без прекъсване – той трябва непрекъснато да набавя нови копия на вече обходените страници. По този начин робот на търсеща машина (Google, Bing), подсигурява сравнително актуален индекс от обхожданите страници. За реални цели, работът трябва да обхожда дадена страница с честота, сходна с тази на промяна на страницата.

Разширяемост: Дизайнът на робот за обхождане, трябва да позволява разширяване за работа с нови формати, нови протоколи за извлечане на страници и т.н. Това изисква модулна архитектура.

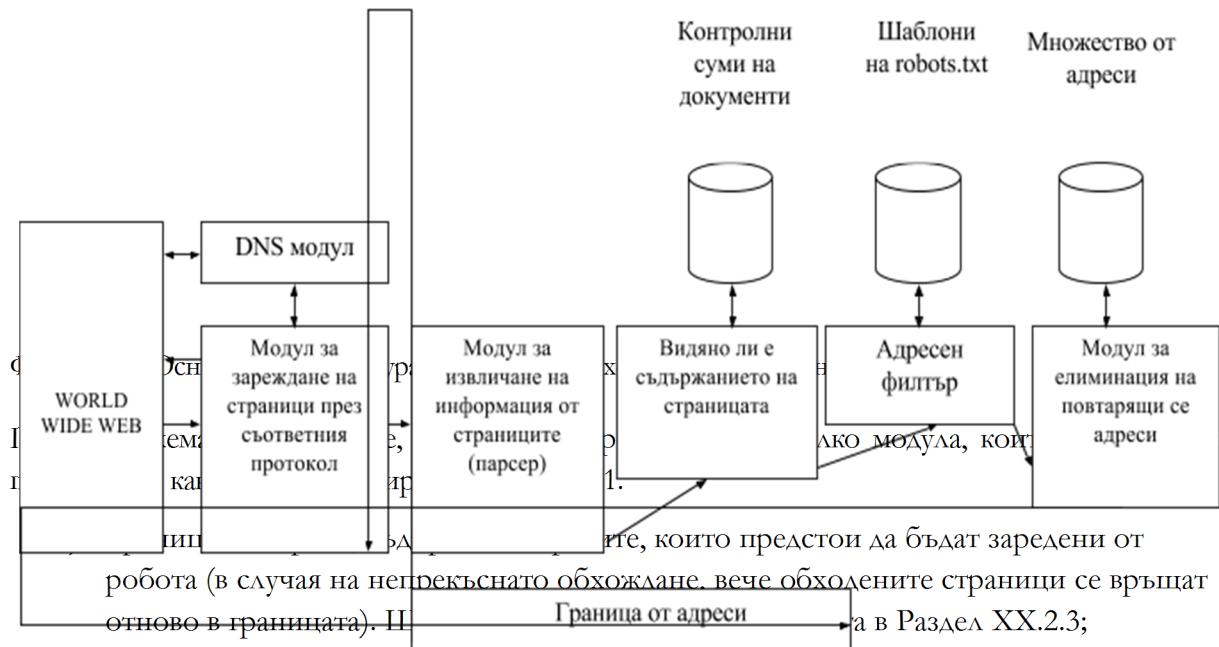
20.2 Обхождане

Следва да изясним метода на работа на един робот за обхождане на хипертекст (независимо дали става въпрос за мрежата, инtranет или друга колекция от документи с хипертекст). Работът започва с един или няколко адреса (URLs), които съставляват началното множество (seed set). Той взима адрес от множество с адреси и извлича страницата, към която той сочи. Страницата впоследствие се обработва (parse), за да се извлекат от нея текстът и връзките (линкове) към други страници. Извлеченият текст се захранва в текстов индекс (описан в глави 4 и 5). Извлечените връзки към други страници се добавят в т.нар. граница от адреси (или гранични адреси – URL frontier), която във всеки един момент съдържа всички адреси, чиито страници предстои да бъдат заредени от робота за обхождане. Първоначално границата от адреси съдържа началното множество от адреси. Когато роботът зареди дадена страница, съответстващият ѝ адрес се изтрива от границата с адреси. На процеса може да се гледа като на обхождане на уеб граф (виж глава 19). В продължително обхождане, адресът на заредена страница се връща обратно в границата, за да бъде заредена отново в бъдеще.

Това наглед просто рекурсивно обхождане на графа на Интернет, на практика е усложнено от многото изисквания към една реална система за обхождане: роботът трябва да е разпределен, мащабируем, ефективен, съобразителен, устойчив и разширяем, като освен това трябва да извлича първо страниците с най-високо качество (полезност). Ще разгледаме аспектите на всяко свойство на робота, така както те са реализирани в Меркатор (Mercator) – реализация на уеб робот, на която се основават редица обхождащи роботи с научна и търговска цел. Нека да отбележим, че за да обходим един милиард страници (малка част от статичния уеб – на 25 юли 2008 инженери от Гугъл обявяват, че системите им за обработка на връзки са открили 1 трилион уникални адреса в мрежата) в рамките на един месец изисква извлечане и обработка на няколкостотин (около 386, ако приемем, че месеца има 30 дни) страници в секунда. Ще видим как може да използваме многонишковия дизайн, за да преодолеем някои препятствия, пред които е изправена системата за обхождане при постигане на тази скорост. Преди да се впуснем в детайли, ще изброим минималния набор от свойства, на които всеки непрофесионален робот за обхождане трябва да отговаря.

- Във всеки един момент не трябва да има повече от една отворена връзка (connection) към даден сървър;
- Между последователните заявки към даден сървър, трябва да има интервал от по няколко секунди;
- Ограниченията описани в следващия Раздел XX.2.1 трябва бъдат спазвани.

20.2.1 Архитектура на уеб робота



- a) Модул за извличане на адреса от текстовия адрес на уеб сайт (пресвърждане на имената или DNS resolution), който да определя, от кой сървър да бъде заредена съответната страница. Ще се спрем по-подробно на този модул в Раздел XX.2.2;
- b) Модул за зареждане, който използва http, за да зарежда уеб страници от съответните адреси;
- c) Модул за обработване на уеб страници, който извлича текста и връзките от тях;
- d) Модул за елиминиране на повторенията, който определя дали извлечената връзка е вече заредена в границата от адреси или е била обходена наскоро.

Обхождането може да се извърши както от една, така и от стотици нишки, всяка от които се движи през логическия цикъл, изобразен на фиг. 20.1. Тези нишки могат да работят в един процес или да бъдат разпределени между много процеси, които работят на различните възли на една разпределена система. Ще отложим описанietо на реализацията на границата от адреси за Раздел XX.2.3., като засега приемаме, че в нея е заредено начално множество от адреси. Ще проследим един адрес от момента на зареждането на страницата, към която сочи, през преминаването ѝ през различни проверки и филтри, до (при непрекъснато обхождане) връщането на адреса в границата от адреси.

Нишката, от която се извършива обхождането, първо взима адрес от границата, след което зарежда съответната уеб страница, най-често с помощта на http. Заредената страница се съхранява временно. След това тя се обработва, за да бъдат извлечени текста и връзките от нея. Текстът (с всяка информация за него – например кои думи са с удебелен шрифт и т.н.) се

предава за индексиране. Информация за хипервръзките, включително техния текст, също се предава за индексиране с цел използването ѝ при определяне на ранга на съответната уеб страница по начини, описани в глава XXI. Всяка извлечена връзка също така преминава през серия от тестове, за да се определи дали да бъде добавена към границата с адреси.

Първоначално, нишката проверява дали страница със същото съдържание вече не е била спрещана на друг адрес. Най-лесният подход за това включва използване на прост отпечатък на страницата, какъвто например е контролната сума (сумата се съхранява в база данни с контролни суми на страници - виж фиг. 20.1). За по-усъвършенстван подход виж Раздел 19.6.

На следващо място се използва адресен филтър, който определя дали извлеченият адрес трябва да се изключи от границата на база на няколко теста. При обхождането може да има изискване за изключване на някои домейни - например тези, завършиващи на .com. В такъв случай тестът ще филтрира адреса, ако в него се съдържа .com. Такъв тест може да е както изключващ, така и включващ. Много сървъри на мрежата не позволяват на роботи да обхождат определени части от техните сайтове чрез стандарта наречен „Протокол за изключване на роботи“. На практика това става, като се постави файл съобразно протокола с името robots.txt в основата на адресната йерархия на сайта. По-долу прилагаме примерно съдържание на robots.txt файла, което указва, че никой робот не трябва да посещава адрес, чиято позиция в йерархията от файлове започва с /yoursite/temp/, освен роботът, наречен „searchengine“.

```
User-agent: *
Disallow: /yoursite/temp/
User-agent: searchengine
Disallow:
```

Файлът robots.txt трябва да бъде извлечен от уеб сайта, за да бъде проверено дали текущия адрес отговаря на изискванията, описани в него, за да бъде добавен в границата от адреси. Вместо да бъде зареждан при всеки тест на нов адрес, сравнително скорошна версия на robots.txt може да бъде кеширана. Това е от особено значение, т.к. много от адресите, извлечени от страница на даден уеб сайт, сочат към други страници от същия сайт. По този начин, когато филтрирането се извършва по време на процеса на извлечение на адреси (хипервръзки) и поставянето им в границата, ще можем ефективно да използваме кеширания файл. Това за съжаление е в разрив с очакванията на уеб администраторите за съобразителност от страна на робота. Адрес (в случая такъв, сочещ към страница с ниско качество или рядко променян документ) може да е в границата от адреси дни или дори седмици. Когато тестваме адреса преди той да се добави в границата, при настъпването на момента на неговото извлечение и зареждане на страницата, към която сочи, файлът robots.txt вече може да е променен така, че да изключи съответния адрес. Следователно трябва да филтрираме по robots.txt точно преди да заредим страницата от даден адрес. Въпреки това, кеширането на robots.txt се оказва високо ефективно, тъй като адресите от даден уеб сайт влизат почти един след друг в границата.

На следващата стъпка трябва да извършим нормализация на относителните хипервръзки. Например на страницата http://en.wikipedia.org/wiki/Main_Page съществува следната относителна хипервръзка:

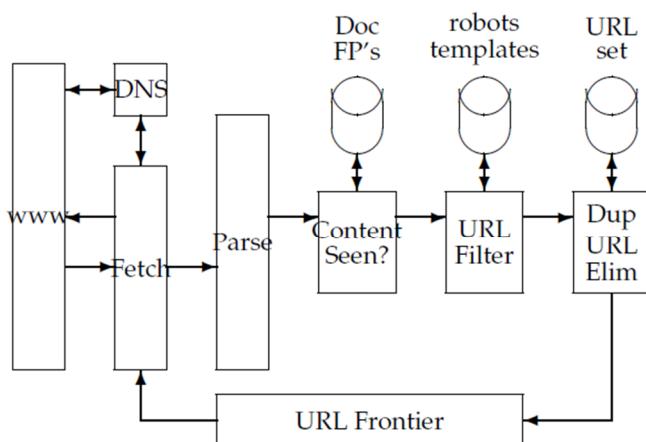
Disclaimers, която сочи към http://en.wikipedia.org/wiki/Wikipedia:General_disclaimer. Накрая се филтрират еднаквите адреси - ако даден адрес е включен в границата или (при непрекъснато обхождане) е вече обходен, той не се прибавя отново. При включване в границата на всеки уеб адрес се присвоява приоритет, на база на който той се извлича за обхождане. Приоритетното подреждане е разгледано в Раздел XX.2.3.

Съществува нишка, която записва статистическа информация за процеса на обхождане. Най-често тя е в процес на изчакване, като на даден интервал (от няколко секунди) се включва и записва данни за състоянието на обхождането (обходени адреси, размер на границата от адреси и други), взема решение дали да се прекрати обхождането или запазва текущото състояние на робота. Това може да се реализира чрез запазване на адресите в границата на твърдия диск. По този начин при неочеквано прекъсване на робота в резултат на форсмажорни обстоятелства, работата може да продължи от най-скорошното съхранено състояние.

20.2.2 Разпределение на работата

Както вече споменахме, робота може да обхожда Интернет чрез много нишки, работещи в различни процеси, всеки от които на различен възел на системата за обхождане. Подобно разпределение е от съществено значение за мащабираността. То може да е полезно и при географски разпределена система за обхождане, при която всеки възел обхожда сървърите в „близост“ до него. Разделянето на обхожданите сървъри по възли, може да се реализира чрез хеш функция или чрез никакъв специално разработен метод. Можем да използваме възел, локализиран в Европа, който да се фокусира върху европейски домейни. На този възел най-вероятно обаче, ще се наложи да „излиза“ от европейския регион поради следните причини – пътищата, през които преминават пакети през Интернет не винаги отразяват географска близост, освен това домейнът на даден сървър може да не отразява неговото физическо местоположение.

Интерес представлява начинът, по който комуникират и споделят адреси помежду си възлите на разпределена система за обхождане. Той включва възпроизвеждането на цикъла от фиг. 20.1 на всеки възел с една съществена разлика - след модула за филтриране на адреси, използваме модул за разпределение на адресите по възлите, отговорни за тяхното обхождане. Така модифицираната архитектура на робота е представена на фиг. 20.2 по-долу.



Фигура 20.2 Архитектура на разпределен робот за обхождане на Интернет

Резултатът от модула за разпределяне на адреси отива в модула за елиминиране на повтарящи се адреси на съответните възли. Задачата пред модула, който определя дали дадено съдържание вече е обходено, се усложнява от следните фактори:

- За разлика от границата от адреси и модула за елиминиране на повтарящи се адреси, контролните суми на документи не могат да се разпределят на база името на сървъра. Няма пречки пред това, едно и също (или почти едно и също) съдържание да се намира на различни уеб сървъри. Следователно, множеството от отпечатъци на страниците трябва да бъде разпределено по възли, по определен начин (например вземайки предвид контролната сума по модул броя възли). В резултат от несъответствието по локализация, текстът „Виждано ли е това съдържание“ най-често ще води до обръщение към други възли (въпреки че е възможно заявките да се групират по някакъв принцип).
- Наблюдава се твърде ниска близост между потока от документи (страници) с близки отпечатъци, което означава, че кеширането на контролни суми няма да помогне.
- Документите се променят с времето и в контекста на непрекъснато обхождане се налага премахване на остарелите отпечатъци от множеството видяни документи. За да се направи това, е необходимо да запазим отпечатъка на документа в границата от адреси заедно със самите адреси.

20.2.3 Извличане на IP адреса, който седи зад даден домейн

Всеки уеб сървър, свързан с Интернет, има уникален IP адрес – последователност от 4 байта, разделени с точка – например 91.198.174.225 е адресът на www.wikipedia.org. Процесът наречен „Превеждане на имена“ (DNS resolution или DNS lookup) представлява превеждането на текстовия адрес на даден уеб сайт (www.wikipedia.org) към IP адреса на сървъра на този сайт. В случая DNS означава Domain Name Service – услугата, която извършва превод на домейн имена към IP адреси. При превеждането на домейн имена програмата, която иска да извърши превода (в случая компонент на робота за обхождане) се свързва с DNS сървър, който връща преведения IP адрес. На практика преводът може да се извърши от няколко DNS сървъра. Първоначално запитаният DNS сървър може рекурсивно да се обърне към други DNS сървъри, за да завърши превода. За по-сложен адрес, като например http://en.wikipedia.org/wiki/Domain_Name_System, компонентът от уеб робота, който е отговорен за превеждането, извлича името на сървъра – в случая en.wikipedia.org, и търси IP адреса за него.

Превеждането на домейн имена е общиизвестна „тапа“, където се получава забавяне при обхождането на мрежата. Поради разпределената същинност на услугата за превод на домейн имена, превеждането може да включва множество заявки и обиколки през Интернет, които изискват по няколко секунди, а понякога дори и повече. Това непосредствено застрашава целта ни да извлечаме по няколкостотин страници в секунда. Метод за справяне с проблема е кеширането – домейни, чийто имена насокро сме превеждали към IP адреси, е вероятно да бъдат в кеша. По този начин можем да намалим необходимостта от това да изпращаме нови заявки за превод към тях. Спазването на правилата по съобразителност (учтивост – виж Раздел

XX.2.3) ограничава възможността ни да се възползваме от кеша на услугата за превод (the DNS cache).

Има още едно препятствие от особено значение в превеждането на домейн имената на сайтовете към техните IP адреси – процедурата е реализирана синхронно в стандартните библиотеки, т.е. тя не е пригодена за работа в многонишков режим. Това означава, че след като е отправена заявка към услугата за превеждане на имена, другите нишки, чрез които работи роботът за обхождане, влизат в режим на изчакване (блокират), докато услугата приключи и върне отговор. За да заобиколят това, повечето уеб роботи разполагат със свои собствени разработки на запитване към услугата за превод на имена. Например, нишка i , на която се изпълнява кодът за заявки към услугата за превод на имена (DNS), изпраща съобщение до DNS сървъра, след което влиза в режим на изчакване, от който излиза или при сигнал от друга нишка, или когато измине определеното за изчакване време. Нишката, която подава сигнал към i има задачата да следи на стандартния DNS порт (53 порт) за пакетите с информация, в които се съдържа отговорът от услугата за превеждане на имена. При получаване на отговор, тази нишка подава сигнал на i , която изпълнява кода за превод на имена и, ако i не е излязла от режима на изчакване, ѝ препраща пакетите с отговора от услугата. Обхождаща нишка, която приключи поради това, че периодът ѝ на изчакване е истекъл, опитва отново, като изпраща ново съобщение до сървъра за превод на имена и отново изчаква за отговор. Дизайнерите на уеб робота Меркатор (Mercator) препоръчват около 5 опита. Добра практика е, времето за изчакване да се увеличава експоненциално с всеки нов опит. Меркатор започва с време на изчакване от 1 секунда, като приключва с приблизително 90 секунди. По този начин се отразява фактът, че за превеждането на някои имена на домейни се изискват десетки секунди.

20.2.4 Граница от адреси (the URL frontier)

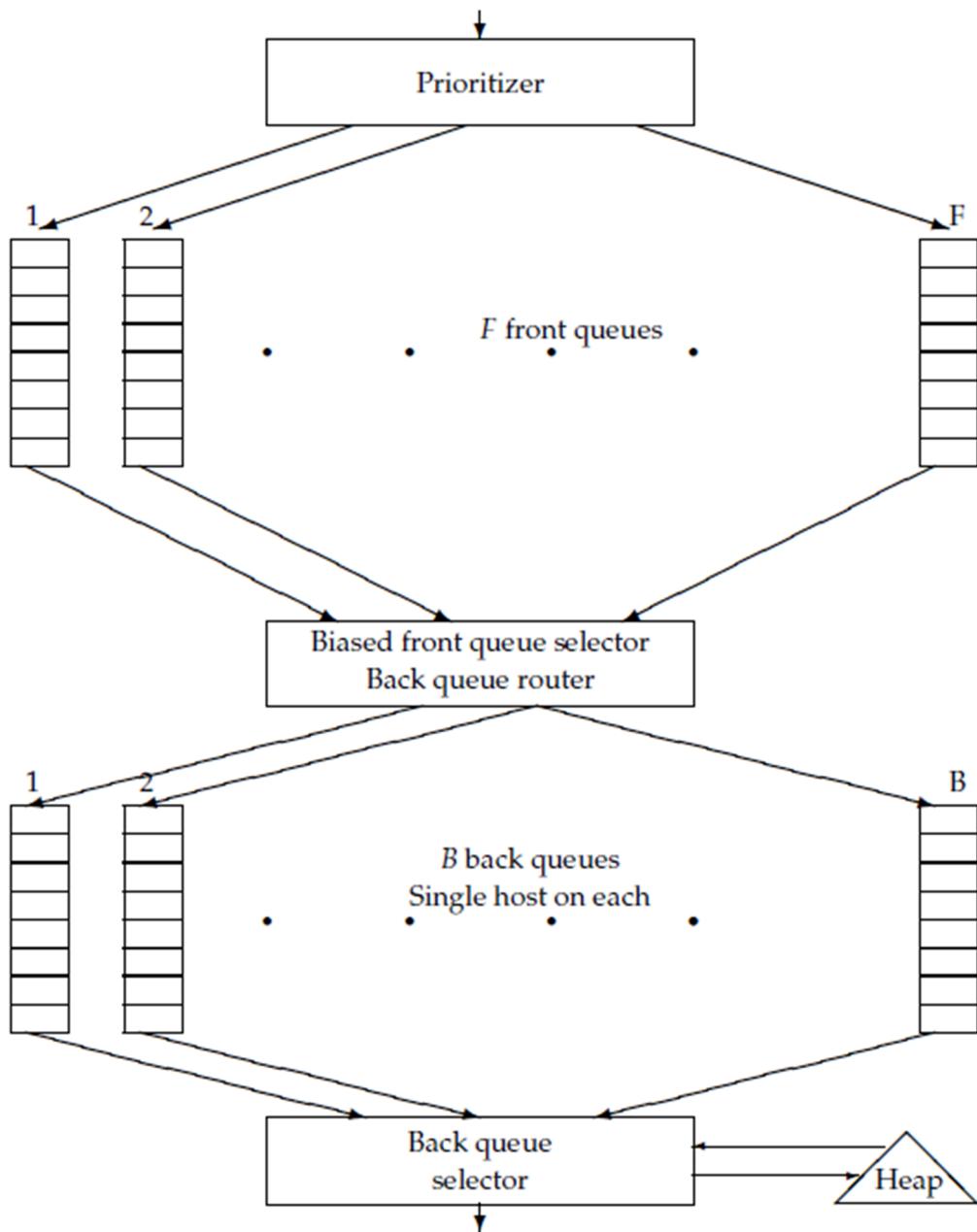
Към границата от адреси на даден node, се добавя URL от процеса който обхожда (или от разпределителя на хостове на друг crawling процес). Той поддържа URL-ите в границата и ги подава в определен ред когато обхождащата нишка поиска URL за обхождане. Две важни съображения определят реда, по който се връщат URL адреси от URL границата. Първо, високо-качествени страници, които сменят съдържанието си често, трябва да бъдат приоритизирани за по-често обхождане. По този начин, приоритет на страницата трябва да бъде функция на неговата честота на промяна и на неговото качество (с помощта на някакво приемлива оценка за качество). Тази комбинация е необходима, тъй като голяма част от спам страниците, сменят напълно съдържанието си на всяко обхождане.

Второто съображение е учитивост. Трябва да избегнем многоократни заявки към един хост в кратък интервал от време. Вероятността за това се задълбочава, заради формата на локална референтност - много URL-ли от дадена страница сочат към други страници на същият хост. В резултат на това, границата от адреси, имплементирана като проста приоритетна опашка, може да доведе до прилив от заявки до даден хост. Това може да се случи, дори ако ограничим обхожданция робот така, че максимум една нишка да може обхожда даден хост във всеки един момент. Често срещана практика е, да се постави пауза между успешните обхождания на даден хост, която е от порядък по-голям от времето отнело за последното обхождане на страница от този хост.

Фигура 20.3 показва, учтива и приоритизирана имплементацията на граница от адреси. Неговите цели са да се гарантира, че само една връзка е отворена по всяко време към даден хост, че период на изчакване от няколко секунди се появява между две последователни заявки към хост и, че страниците с висок приоритет се обхождат преференциално.

Двата основни подмодула са набор от F (front/граница) опашки в горната част на фигурата и набор от B (back) опашки в долната част. Всички тези опашки са от вида FIFO (first in, first out) опашки. Границните опашки имплементират приоритизацията, а back опашките имплементират учтивостта. Докато се изпълнява цикълът на URL-лите в границата, минавайки през front и back опашките, приоритизаторът им слага даден приоритет под формата на число от 1 до F, базирано на историята на обхождане (като се взимат предвид скоростта, с която уеб страница на този адрес се е променила между предишните обхождания). Например, на страница която се променя често ще бъде сложен висок приоритет. Друга практика, например, може да бъде зависимост към приложението и експлицитност – например URL адресите на новинарските услуги да бъдат винаги назначавани с най-висок приоритет. Сега, когато му е сложен приоритет I, URL-а се добавен към i-тата front опашка.

Всяка от back опашките поддържа следните инварианти: (I) е непразно когато обхождането е в ход и (II) съдържа само URL адреси от един хост. Следователна таблица T (Фигура 20.4) се използва за поддържане на mapping-а между хост и back нишките. Всеки път когато back нишка се освободи, се запълва от front нишка и таблица T трябва да бъде актуализирана.



Фигура 20.3 Граница от адреси (URL frontier). Адресите, извлечени от вече обходени страници се вливат в горната част на фигурата. Обхождащата нишка взима адресите от долната част на фигурата. По пътя си, адресът преминава през една от няколко front опашки, които управляват неговият приоритет за обхождане, последвани от един или няколко back опашки които управляват учтивостта.

Host	Back queue
stanford.edu	23
microsoft.com	47
acm.org	12

Фигура 20.4 Пример за помощна host-to-back queues таблици

В допълнение, поддържаме heap с един запис за всяка back опашка, както и най-ранното време t , в което може да се свържем отново със съответстващия хост на тази опашка.

При извличането на адресите от границата, нишката на обхождане извлича корена на този heap и (ако е нужно) изчаква докато дойде съответното време t . Тогава взима адреса u , което е в началото на back нишката j съответстващо на извлеченият корен на heap-а и го обхожда. След обхождане на u , извикващата нишка проверява дали j е празна. Ако да, тя взима front нишка и извлича от нея адреса v . Изборът на front нишка е насочен (обикновено се прави от отделен случаен процес) към нишки с висок приоритет, осигурявайки придвижването на адреси с по-висок приоритет към back нишките. Проверява се v , за да се провери дали вече има back опашка, която съдържа адреси от този хост. Ако да, v се добавя към тази опашка и отново се обръщаме към front опашките, за да намерим друг адрес, който да вкарваме в сега празната опашка j . Този процес продължава докато j не е празна отново. Във всеки случай, нишката вкарва запис в heap-а за j с ново най-ранно време за достъп t , на базата на свойствата на адреса в j , който е бил последно обходен (като например, кога за последно е бил обходен хоста и времето, което е отнело на нишката да го обходи), след което продължава своето изпълнение. Например, новият запис t , може да бъде текущото време + десет пъти последното време за обхождане.

Броят на front опашките, заедно с политиката за слагане на приоритети и избор на опашки, определят приоритетните свойства които искаме да вкарваме в системата. Броят на back опашки определя размерите, до които може да държим всички обхождащи нишки заети, докато запазваме учитивост. Дизайнърите на Mercator препоръчват, говорейки грубо, да има три пъти повече back опашки отколкото обхождащи нишки.

При паяк/робот пуснат в мрежата, границата от адреси може да нарастне до такава степен, че да изисква повече памет, отколкото е налична на даден node. Решението е да се остави по-голямата част от границата от адреси да пребивава на диск. Част от всяка опашка да се държи в паметта, като с извличането ѝ, да се презарежда от диска.

УПРАЖНЕНИЕ 20.1:

Зашо е по-добре да се разделят хостовете (вместо индивидуалните url-ли) между различните nodes на разпределена обхождаща система?

УПРАЖНЕНИЕ 20.2:

Зашо трябва хост сплитера да е преди duplicate url eliminator?

УПРАЖНЕНИЕ 20.3:

В предишната дискусия се натъкнахме на две препоръчителни „hard constants” – увеличаването на t с десет пъти последното време на обхождане на хоста, и броят на back опашките да са три пъти поне колкото е броят на обхождащите нишки. Как са свързани тези две константи?

20.3 Разпределение на индексите

В секция 4.4 описахме разпределението на индекси. Нека сега разгледаме разпределението на индекс върху голяма група компютри, поддържаща заявки. Две очевидни алтернативни реализации на индекса са следните: разделяне спрямо термините, още известно като глобална индекс организация и разделяне спрямо документите или така наречената локална индекс организация. В първата, речникът на индексните термиини е разположен в подмножество, а всяко подмножество се намира в отделен възел. Заедно с условията в възела се съхраняват и техните публикации. Заявка се отправя към възела в зависимост от определените условия. По

принцип това позволява по-голяма конкурентност, тъй като потока от различни заявки може да се разпредели между отделните множества от машини.

На практика, разделянето на индекси посредством речник се оказва нетривиален въпрос. Обработването на заявки съставени от много думи изискват изпращането и сливането на дълги списъци от публикации между отделните множества от възли. Разходите направени за тези действия могат да надделят над по-голямата конкурентност. Балансирането на натоварването не се регламентира от предварителен анализ на свързаните честотни понятия, а по-скоро от условията на заявките и съществуващите ги събития. Постигането на добри дялове е функция на съществуващите ги събития и води до струпване на условия за оптимизиране на цели, които не са лесни за количествено определяне. Тази стратегия прави имплементирането на динамична индексация по-трудно.

По разпространена имплементация е да се направи разделяне спрямо документите - всеки възел съдържа индекс за подмножество на всички документи. Всяка заявка се разпределя към всички възли заедно с резултата от различните възли, обработени преди представянето на потребителя. Тази стратегия определя повече локални търсения в диска, за сметка на по-малка комуникация между различни възли. Една от трудностите в този подход е, че глобалните статистики, които се използват за оценяването, като idf , трябва да се изчислява на базата на цялата колекция от документи, въпреки че индексът на всеки един възел съдържа само набор от документи. Те се изчисляват от разпределени процеси, които периодично опресняват индексите на възлите с актуални статистики.

Как се взима решението за подялбата на различните документи между отделните възли? Въз основа на развитието направено по архитектурата на работа в секция XX.2.1, един прост подход би бил да се възлагат всички страници от един хост на един възел. Това разделение може да доведе до поделяне на хостовете до различните възли на работа. Опасността при такова разделяне е, че при отправяне на много заявки, превес на резултатите ще имат тези от документите, които са от хостове с малък брой (а оттам и малък брой на индекс nodes).

Съответствие между всеки адрес в пространството и индексен възел има, като резултат на по-равномерно разпределение на времето за заявки между отделните възли. По време на изгълнението на заявката, всяка се разпространява до всички възли. Една обща имплементационна практика е, да се раздели колекцията от документи на различни документни индекси. Разделянето се осъществява на базата на по-голяма и по-малка вероятност за удовлетворяване на заявка (могат да се ползват техники от глава 21 за определяне на високо и ниско оценени индекси). Търсим в тези с по-слаба оценка само при недостатъчно попадения при тези с по-висока, както е обяснено в секция 7.2.1.

20.4 Свързващи сървъри

Поради причини, които ще бъдат изяснени в глава 21, търсачките изискват сървър чиято свързаност поддържа бързи заявки в уеб графата. Типични заявки за свързаност са тези, в които адресите сочат към даден адрес и други в които адресите изграждат връзката до адрес. За тази цел ние искаме да се съхранява връзката в паметта която сочи от адреса към външните линкове и от адреса към вътрешните линкове. Програмите включват обхождащ контрол, уеб

графичен анализ, сложна обхождаща оптимизация и анализ на свързаността (ще бъде засегнато в глава 21).

Да предположим, че в мрежата има четири милиарда страници, всяка от тях с по десет линка към друга страница. В най-простата форма, ние ще имаме 32 бита или 4 байта, за да се уточни всеки край (източник и местоназначение) на всяка връзка, която изисква общо „ $4 \times 10^9 \times 10 \times 8 = 3.2 \times 10^{11}$ “ байта памет. Някои от основните свойства на уеб графиката могат да бъдат така експлоатирани, че да използват добре под 10% от изискванията на тази памет. На пръв поглед, ние очакваме да имаме проблем с компресирането на данни, което е възможно да доведе до вариация от стандартни решения. Въпреки това, нашата цел не е просто да компресираме уеб графиките, за да се поберат в паметта. Ние трябва да го направим по начин, който ефикасно поддържа свързаността на заявките. Това предизвикателство напомня на компресията по индекс (глава 5).

Предполагаме, че всяка уеб страница е представена от уникално число - специалната схема за използване присвояването на тези числа е описано по-долу.

Ние изграждаме съседна таблица, която наподобява обърнат индекс. Тя има ред за всяка уеб страница, с редовете, заявени от съответните числа. Редът за всяка страница „ p “, всяко от което съответства на уеб страница която сочи към „ p “. Тази таблица ни позволява да отговаряме на заявки от формата: „коя страница сочи към „ p “? По подобен начин ние изграждаме таблици, чито записи са страници сочещи към „ p “.

```
1: www.stanford.edu/alchemy
2: www.stanford.edu/biology
3: www.stanford.edu/biology/plant
4: www.stanford.edu/biology/plant/copyright
5: www.stanford.edu/biology/plant/people
6: www.stanford.edu/chemistry
```

Фигура 20.5 Лексикографски подреден набор от адреси

Това таблично представяне съкращава мястото заемано от стандартното представяне (в което изрично представяме всеки линк чрез неговите две крайни точки, всяка 32 битово цяло число) с 50%. Описанието ни по-долу ще се фокусира върху таблицата за линковете от всяка страница - трябва да е ясно, че техниките са приложими също както и при таблицата с връзките до всяка страница. Колкото повече намалим мястото за съхранение на таблицата, можем да развием следните идеи:

- Подобие между списъци: Много редове в таблица имат много общи записи. В такъв случай, ако изрично представим примерен ред за няколко подобни реда, оставащите могат да бъдат кратко и ясно изразени по отношение на примерния.
- Местоположение: много линкове от една страница отиват до съседни страници, например такива от същия хост. Това предполага, че в кодирането на целта на линка, можем често да използваме малки цели числа и така да спестим място.
- Използваме пропуски в кодирането на сортирани списъци: вместо да пазим целта на всеки линк, ние съхраняваме отместването от предишния запис в реда.

Сега ще разработваме по-подробно всяка от тези техники.

В лексикален ред на всички адреси, обработваме всеки, като низ от числа и букви и подреждаме тези низове. Фигура 20.5 показва част от тази подредба. За правилна лексикографска подредба на мрежови страници, частта с името на домейна от адреса трябва да бъде обърната. Това не е нужно в конкретния случай, запцото се интересуваме предимно от локални за хоста препратки.

На всеки адрес се закача позицията му в тази подредба, като уникален идентификатор. Фигура 20.6 показва пример на такова номериране и резултатната таблица.

Повечето сайтове имат шаблон с набор от връзки от всяка страница в сайта на определена серия от страници (като авторски права, условия за ползване и др.)

```
1: 1, 2, 4, 8, 16, 32, 64
2: 1, 4, 9, 16, 25, 36, 49, 64
3: 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144
4: 1, 4, 8, 16, 25, 36, 49, 64
```

Фигура 20.6 Четири редов сегмент от талибица с линкове

В този случай, редовете, съответстващи на страниците в уебсайта, ще имат много по-чести записи в таблицата. Освен това, по силата на лексикографското подреждане на адресите, е много вероятно страниците от уебсайта да се появяват като граниченци редове в таблицата. Ще приемем следната стратегия: ще ходим надолу по таблицата, кодирайки всеки ред от таблицата по отношение на седемте предходните редове. В примера на фигура 20.6, бихме могли да кодираме на четвъртия ред като „същия като реда на отместване 2 (значава два реда по–нагоре в таблицата), като 9 отмества 8. Това изисква спецификация на изместването, целите числа намаляват (както в случай 9) и се добавят (както в случай 8). Използването на седемте предходни редове има две предимства: изместването може да е изразено само с 3 бита, този избор е емпирично оптимизиран (причината за седем, а не осем предходни редове е предмет на Упражнение 20.4) и фиксирането на максимум измествания за малка стойност избягва провеждането на скъпо търсене на много кандидат–прототипи, в условия, в които да изразят текущия ред.

Какво става, ако нито един от предходните седем реда не е добър прототип за изразяване на текущия ред? Това ще се случи, например, на всяка граница между различни уеб сайтове, докато вървим през редовете на таблицата. В този случай, просто изразяваме реда като започваме от празния и добавяме всяко число в този ред. Използвайки празни кодировки, за съхраняване на пропуските (по–скоро от реалните числа) на всеки ред и кодиращи тези пропуски пълно базирани на разпределението на техните стойности, ние получаваме понататъшно намаляване на пространството. В експерименти, посочени в раздел 20.5, поредица от техники, описани тук, изглежда че се използват само по 3 бита на връзка средно – рязко намаляване от 64 изисква в наивен представяне. Докато тези идеи ни дават представа за значителни уеб графики, които удобно се побират в паметта, ние все още трябва да подкрепяме свързаните запитвания. Какво е довело в извлечане от представителството на набор от връзки от дадена страница? Първо ни трябва индекс на търсения от (хеш) адреса до неговия номер на ред в таблицата. След това трябва да реконструираме тези записи, които могат да бъдат кодирани по отношение на вписванията в други редове. Това се налага, в следствие на компенсиращи измествания на тези редове – процес, който по принцип може да доведе през много нива на околен път. На практика обаче, това не се случва много често. Евристичен за

контролиране, това може да бъде въведено в конструирането на таблици: При разглеждането на предходните седем редове като кандидати, от които да моделира текущия ред, ние настояваме за праг на сходство между текущия ред и кандидата прототип. Този праг трябва да бъде избран внимателно. Ако прагът е твърде висок, ние рядко използваме прототипи и изразяваме много редове отново. Ако прагът е твърде нисък, повечето редове се изразяват по отношение на прототипи, така че при времето за заявка, реконструкцията на ред води до много нива на околнния път през предходните прототипи.

УПРАЖНЕНИЕ 20.4

Отбелязахме, че изразяването на ред по отношение на един от седемте предходни редове, ни позволява да използваме повече от три бита, за да уточним кои от предходните редове използваме като прототипи. Защо седем, а не осем предходни редове? (съвет: разгледайте случая, когато нито един от предходните седем реда не е добър прототип)

УПРАЖНЕНИЕ 20.5

Отбелязахме, че за схемата в раздел 20.4, при декодиране на инцидент на връзки на даден адрес, може да се получат много нива на околен път. Направете пример, в който броят на нивата расте успоредно с броя на адресите

20.5 Библиография

Manning, Christopher, Raghavan, Prabhakar, Schütze, Hinrich, An Introduction to Information Retrieval, Cambridge University Press, 2009.

21 Анализ на връзките

Анализът на хипервръзки и граф-структурата на уеб мрежата играе важна роля за развитието на уеб търсенето. В тази глава ще се съсредоточим върху използването на хипервръзки, с цел подреждане (оценяване) на резултатите от търсене в мрежата. Този анализ на връзките е един от многото фактори, взети под внимание от уеб-търсачките, при изчисляването на композитна оценка за дадена уеб страница при всяка една заявка.

Анализът на връзките има предшественици в областта на анализа на цитати, чиито аспекти се препокриват с област позната като Библиометрия. Тези дисциплини измерват до колко дадена научна статия е повлияна от други, като анализират модела на цитатите сред тях. Много подобно на начинът, по който цитатите представляват предоставяне на авторитет от една научна статия към друга, анализът на връзките в уеб мрежата третира хипервръзките от една уеб страница към друга като предоставяне на авторитет. Разбира се, не всеки цитат или хипервръзка от една страница към друга предполага подобно предоставяне на авторитет,eto защо обикновената преценка на качеството на дадена страница по това колко препратки има към нея от други уеб страници, не е достатъчно надеждно. Примерно, някой умишлено би могъл да направи така, че много уеб страници да сочат към някоя целева страница, с идеята по този начин изкуствено да увеличи броя на препратките към нея. Този феномен е известен като спам. Независимо от това, методът на цитиране е често срецано явление и е достатъчно надеждно, така че е възможно за уеб търсачките да извлекат полезни сигнали за оценяване от по-сложен анализ на връзките. Също така, анализът на връзките е полезен индикатор за това, коя да е следващата обходена страница/и, докато правим уеб обхождане; това се осъществява чрез прилагането на анализ на връзките за задаване на приоритет.

21.1 Уеб мрежата като граф



Фиг. 1

Фигура 1 показва възлите А и Б (уеб страници) от граф-структурата на уеб мрежата, с хипервръзка от А до страница Б. Изучаването на анализа на връзките се базира на две предположения:

- Текстът на връзката (html anchor-тагът - <a>), сочеща към Б е адекватно описание на съдържанието на уеб страница Б.
- Хипервръзка от А до Б представлява одобрение на страница Б, от създателя на страницата А. Не винаги обаче, това е случаят. Примерно, много препратки в страниците на един уеб сайт са създадени по общ шаблон. В повечето корпоративни сайтове от всяка една страница има показалец (препратка) към страница, съдържаща авторско право. Съответно, реализации на алгоритмите за

анализ на връзки, няма да вземат под внимание тези „вътрешни” препратки при оценяването.

21.1.1 Текстът на хипервръзката и уеб графът

Следният фрагмент HTML код от уеб страница показва хипервръзка, сочеща към началната страница на Journal of the ACM:

```
<a href="http://www.acm.org/jacm/">Journal of the ACM.</a>
```

В този случай, връзката сочи към страницата е <http://www.acm.org/jacm/> и текстът на котвата е *Journal of the ACM*. Ясно е, че в този случай котвата е описание на целевата страница. Но целевата страница (Б = <http://www.acm.org/jacm/>), сама по себе си съдържа същото описание, както и значителна допълнителна информация за журналът. Така че, за какво служи текстът на котвата?

В уеб пространството е пълно с примери, в които страницата Б не предоставя точно описание на себе си. В много случаи става въпрос за това, как издателите на страница Б са избрали да се представят; в доста голяма степен това се отнася за корпоративните сайтове, където уеб представянето е маркетингова стратегия. Да вземем за пример IBM – по времето когато тази книга (Introduction to Information Retrieval) е писана, началната страница на IBM (<http://www.ibm.com>) не е съдържала понятието *компютър* никъде в HTML кода си, въпреки че IBM е смятан за един от най-големите производители на компютри. По същия начин, началната страница на Yahoo! (<http://www.yahoo.com>) не е съдържала нито веднъж думата *портал*.

Често има разминаване между термините в дадена уеб страница и начинът, по който уеб потребителите ще опишат тази страница. Следователно потребителите, които търсят в уеб пространството не е нужно да използват термините описани в тази страница, за да направят заявка за нея. Като допълнение, много уеб страници са изпълнени с графики и изображения и/или текстът е вграден в тези изображения; в такива случаи извършеният HTML разбор (парсинг) по време на обхождането, няма да успее да извлече текст, който да е полезен за индексирането на тези страници. Стандартният подход за извлечение на информация в този случай би бил да се използва „Разширен подход за моделиране на езика“ или някой от методите за даване на обратна връзка и разширение на заявката. Прозрението относно текста на хипервръзките е, че подобни методи биха могли да бъдат изместени от него.

Фактът, че текстът на котвата на много препратки към <http://www.ibm.com> съдържа думата *компютър*, може да бъде използван от уеб търсачките. Например могат да бъдат включени условия за текст на хипервръзките (...) като някое от условията за идексиране на целевите страници. По този начин, с помощта на специален индикатор който да показва, че дадени термини се срещат като текст на хипервръзка (а не в страница), когато правим търсене по думата *компютър*, <http://www.ibm.com> ще бъде намерено като резултат. Както и при текста съдържащ се в страницата, текстът на хипервръзката обикновено има теглова стойност, която се определя от честотата с която се среща, като има наказателна стойност за теглото на дадена котва, ако текстът ѝ се среща твърде често (Пример – *Кликни тук*).

Използването на хипервръзки е довело до интересен страничен ефект. Търсенето на *big blue* в повечето уеб търсачки връща като най-вероятен резултат началната страница на IBM; това е

свързано с популярният прякор който много хора използват, реферирали към IBM. От друга страна, случвало се е (и продължава да се случва), някои уронващи текстове на хипервръзки, като на пример империя на злото, да доведат до неочекани резултати при търсене с някои учебни търсачки. Този феномен може да се използва при организирани акции срещу точно определени сайтове. Такива целенасочени текстове на хипервръзки може да са форма на спам, тъй като някой уеб сайт може да създаде подвеждащи котви, сочещи към себе си, с цел да повиши рейтинга си при определени заявки за търсене. Откриването и борбата с такива систематични злоупотреби с текстовете на хипервръзките е друга форма на откриване на спам, която учебни търсачките прилагат.

Рамката от текст, която загражда хипервръзката (наричана още разширен текст на хипервръзката), в много случаи е не по-малко важна и значеща от самия текст на връзката; нека вземем за фрагмента: *there is good discussion of vedic scripture <a>here*. Това е взето предвид в наб



Фиг. 2 Случайният потребител попаднал на уеб сайт А и може да премине към Б, В или Г с вероятност 1/3

21.2 Ранкиране на страницата (PageRank) ()

Сега ще се фокусираме върху оценяването на критериите получени само от структурата на връзките. Първата техника за оценяване на връзките, която ще разгледаме, се изразява в това, да се зададат числени оценки на всеки възел от уеб граф структурата в диапазона между 0 и 1. Тази оценка е известна като неговия PageRank (ранкова страницата, PageRank е наименовано на Лари Пейдж). За всяка заявка за търсене, уеб търсачката изчислява сборна оценка на всяка страница, която съчетава стотици функции. Този комбиниран рейтинг се използва за получаването на оценен списък от резултатите от заявката.

Нека си представим случаен потребител, който започва от дадена уеб страница (възел от граф структурата на уеб пространството) и предприема произволно преминаване към друга страница. Всеки път потребителят преминава от текущата страница A, до случайно избрана страница „X“ към A прави препратка. На фиг.2 можем да разгледаме движението на потребителя от възел A до която и да е от трите страници B, В и Г, до които има хипервръзка от страницата A. Потребителят ще премине от A към всеки един от 3-те възела с един и съща вероятност – 1/3.

След като потребителят прави произволно обхождане от връх на връх, той ще посети някои възли повече пъти отколкото други. Има възли, към които сочат препратки от други, често посещавани възли. Идеята на PageRank е, че възлите посещавани по-често по време на това обхождане са по-важни.

А какво ще стане ако текущата страница А, на която се намира потребителят, няма нито една изходяща препратка? За да разрешим този казус, ще дефинираме нова операция за нашият потребител – възможността да се *телепортира*. Тази нова операция се изразява в това, че потребителя може да прескочи от дадена страница А до всяка друга страница „Х“ от граф структурата на уеб пространството. Това може да се постигне, ако той напише определен уеб адрес в URL полето на своя браузер. Крайната дестинация на една телепортираща операция се моделира като се направи единозначен избор измежду всички уеб страници. С други думи, ако крайният брой на всички възли в даден уеб граф е N, тогава телепортиращата операция отвежда потребителя до следващ възел с вероятност $1/N$. Също така, потребителят ще се телепортира до сегашната си позиция със същата вероятност - $1/N$.

При задаването на PageRank оценка за всеки възел от уеб графа, ние можем да използваме операцията по телепортиране по 2 начина:

1. Когато се достигне до възел без изходящи връзки;
2. При всеки един възел, който има изходящи връзки, потребителят може да извърши операцията телепортиране с вероятност $0 < \alpha < 1$ и стандартното произволно обхождане с вероятност $1 - \alpha$, където α е фиксиран, предварително избран параметър. Обикновено α би могло да бъде 0,1.

21.2.1 Вериги на Марков

Верига на Марков в теорията на вероятностите, е процес на Марков, който приема стойности от дискретно множество, наречено пространство на състоянията, като стойността му се изменя във фиксириани моменти от времето. Казва се, че веригата на Марков е в определено състояние. Пространството на състоянията може да бъде крайно или безкрайно, но изброимо множество. Това е една от разновидностите на случаен процес.

Въпреки, че вериги на Марков се използват за описание на процеси, чиито стойности се променят в точно определени моменти от времето (разделени на равни интервали от време или не), самият модел не зависи от времето, а само от това, колко пъти е бил активиран, т.е. броя тактове.

Веригата на Марков се характеризира от вероятностна матрица на преходите $P(N \times N)$, като всеки от елементите ѝ е в интервала $[0, 1]$; сумата от елементите по всеки ред на P е 1. Веригата на Марков на всяка стъпка попада в едно от N определени състояния. Всяка стойност P_{ij} , е известна като вероятност на прехода и зависи само от текущото състояние i ; Това е известно като свойство на Марков. Според това свойство:

$$\forall i, j, P_{ij} \in [0, 1]$$

и

$$\forall i, \sum_{j=1}^N P_{ij} = 1. \quad (*)$$

Матрица с неотрицателни елементи, която удовлетворява уравнение (*) е известна като стохастична матрица.

0.5 1



1 0.5

Фиг.3

Във веригата на Марков, разпределението на вероятностите на следващото състояние на веригата се влияе само от текущото състояние, но не и от предходните. Фигура 3 показва прости вериги на Марков, състояща се от 3 състояния. От състояние А можем да продължим към Б и В с еднаква вероятност - 0.5. От Б и В се придвижват към А с вероятност 1. Тогава матрицата на преходите Р е:

$$\begin{pmatrix} 0 & 0.5 & 0.5 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

Разпределението на вероятностите в отделните етапи на веригата на Марков може да се разгледа като вектор на вероятностите: вектор, на който всички стойности са в интервала $[0, 1]$ и събърт им не надвишава 1. Един N-мерен вероятностен вектор, чийто компоненти отговарят на N-те състояния във веригата на Марков, може да бъде разглеждан като вероятностно разпределение върху тези състояния. За нашият прост пример от фигура 3, ще имаме вектор на вероятностите с 3 компонента, чиято сума е 1.

Можем да представим произволното движение на даден потребител в уеб пространството чрез верига на Марков, като всяко нейно състояние, всъщност, е една уеб страница и всяка вероятност на прехода представлява вероятността потребителят да премине от една страница към друга. Операцията по телепортиране също спомага за тези вероятности на прехода. Матрицата на съседство A за даден уеб граф е дефинирана по следния начин: Ако съществува хипервръзка от страница i до страница j, тогава $A_{ij} = 1$ в противен случай $A_{ij} = 0$. Лесно можем да изведем матрицата на преходната вероятност P за верига на Марков от матрицата $A(N \times N)$:

- 1 Ако ред от A не съдържа нито веднъж 1, замени всеки елемент с $1/N$ и за всички останали редове направи същото
- 2 Раздели всяка 1-ца от A на броя на 1-ците в реда в който се намира. Така ако на даден ред има три 1-ци, всяка една от тях е заменена с $1/3$.
- 3 Умножи получената матрица с $1 - \alpha$.
- 4 Добави α/N към всеки елемент на получената матрица за да се получи P.

Можем да изобразим вероятностното разпределение за страницата, на която се намира потребителят по всяко време с помощта на вероятностния вектор x^{\rightarrow} . При $t=0$, потребителят може да започне обхождането от позиция (състояние), чиято съответна стойност във вектора x^{\rightarrow} е 1, а всички останали - 0. При $t=1$, по дефиниция, разпределението на потребителя се определя от вероятностния вектор $x^{\rightarrow}P$. За $t=3 \Rightarrow (x^{\rightarrow}P)P = x^{\rightarrow}P^2$ и т.н. По този начин можем да изчислим вероятностното разпределение за всяка една позиция, която потребителят заема (местоположението му в уеб пространството при всяко едно преминаване от страница в страница), като разполагаме само с първоначалното разпределение и матрицата на вероятностните преходи P.

Ако пуснем в действие веригата на Марков за многообразни стъпки, през всяко състояние се преминава с различна честота, която зависи от структурата на веригата. За осъществяването на нашия алгоритъм, нека предположим, че потребителят посещава определени страници по-

често, отколкото други (началните страници на новинарски сайтове примерно). Сега можем да направим това предположение точно, въвеждайки условия, при които честотата на посещенията клони към фиксирано, установено състояние. Имайки това предвид, можем да зададем PageRank на всеки възел V , основавайки се на това установено състояние на посещаемост и да покажем как тази оценка се пресмята.

Определение: Една верига на Марков се нарича *ergodic*, ако съществува положително число T_0 , такова че за всяка двойка състояния i, j от веригата на Марков, ако тя започва от време 0 на позиция i , тогава за всяко $t > T_0$, вероятността да се намираме в състояние j по време t , е по-голяма от 0.

Оригиналният модел на Сергей Брин и Лари Пейдж за PageRank използва структурата от хипервръзки на мрежата, за да построи верига на Марков с *примитивна* (виж по-долу) вероятностна матрица на преходите P . Несъкратимостта на матрицата (никой ред/колона от матрицата не е линейна комбинация на друг неин ред/колона) гарантира, че тя е сходима (след известен брой итерации) към вектора π^T , известен като PageRank., т.е. че той съществува.

Една матрица е несъкратима, ако от всеки връх на граф построен от нея има връзка към всеки друг връх. Неотрицателна, несъкратима матрица е *примитивна*, ако има само една собствена стойност в рамките на спектралния си радиус (радиусът на най-малкия централен кръг в комплексната равнина, който съдържа собствените стойности на A). Несъкратима верига на Марков с примитивна матрица на преходите се нарича непериодична верига.

Две са основните технически условия, за да бъде една верига на Марков *ergodic*: тя трябва да е *несъкратима и непериодична*.

Теорема: За всяка *ergodic* Марковска верига, съществува единствено установено състояние с вероятностен вектор $\pi \rightarrow$, който е собствения (eigen) вектор на P , така че ако $\eta(i, t)$ е броя посещения в състояние i на стъпка t , тогава:

$$\lim_{t \rightarrow \infty} \frac{\eta(i, t)}{t} = \pi(i),$$

Като, $\pi(i) > 0$ е установленото състояние на вероятността за стъпка i .

21.2.2 Пресмятане на PageRank (оценяване на страницата)

Нека имаме следното равенство:

$$\vec{\pi} P = \lambda \vec{\pi}.$$

(**) С други думи, уравнение (**) може да се представи така: ако $\pi \rightarrow$ е вероятностното разпределение за обхождането на потребителя на уеб страниците, той ще остане в установлено състояние на разпределение $\pi \rightarrow$. Знаеши, че $\pi \rightarrow$ е установлено състояние на разпределение, получаваме $\pi P = \pi$, следователно 1 е собствена стойност на P . Като по този начин, ако трябва да изчислим останалата част от собствения вектор на P – тази със собствена стойност 1, това щеше да е стойността на PageRank.

Нека разгледаме следният прост алгоритъм наричан още *power iteration*. Ако $x \rightarrow$ е първоначалното разпределение върху състоянията, тогава разпределението при итерация t е $x \rightarrow P^t$. С увеличаването на t бихме очаквали, че $x \rightarrow P^t$ е много сходно на разпределението $x \rightarrow P^{t+1}$, понеже

за голяма стойност на t бихме очаквали веригата на Марков да постигне своето стабилно състояние. Методът на *power iteration* симулира уеб „разходката“ на потребителя: започвайки от стъпка a и започвайки обхождане за голям брой стъпки t , като запазваме статистика за честотата на посещаване на всяка една от тях. След голям брой стъпки t , тези честоти на посещаемост се „ успокояват“ така, че варирането на пресметнатите честоти се запазва под някакъв предварително установен prag. Можем да твърдим, че този prag на честотата на посещаемост е стойността на PageRank.

Пример: Нека имаме следният уеб граф с три върха 1, 2 и 3. Връзките са както следва: $1 \rightarrow 2, 3 \rightarrow 2, 2 \rightarrow 1, 2 \rightarrow 3$. Напишете вероятностната матрица на преходите за потребител извършващ уеб „разходка“ с телепортиране, при условие че $\alpha = 0.5$.

$$P = \begin{pmatrix} 1/6 & 2/3 & 1/6 \\ 5/12 & 1/6 & 5/12 \\ 1/6 & 2/3 & 1/6 \end{pmatrix}. \quad (***)$$

Нека си представим, че потребителя започне от позиция 1, отговаряща на първоначалния вероятностен вектор на разпределението $X_0^{\rightarrow} = (1 \ 0 \ 0)$. Тогава след 1 стъпка разпределението ще бъде:

$$\vec{x}_0 P = (1/6 \ 2/3 \ 1/6) = \vec{x}_1. \quad (****)$$

След 2 стъпки :

$$\vec{x}_1 P = (1/6 \ 2/3 \ 1/6) \begin{pmatrix} 1/6 & 2/3 & 1/6 \\ 5/12 & 1/6 & 5/12 \\ 1/6 & 2/3 & 1/6 \end{pmatrix} = (1/3 \ 1/3 \ 1/3) = \vec{x}_2. \quad (*****)$$

Продължавайки по този начин, получаваме последователност от вероятностни вектори показани на фигура 4:

\vec{x}_0	1	0	0
\vec{x}_1	1/6	2/3	1/6
\vec{x}_2	1/3	1/3	1/3
\vec{x}_3	1/4	1/2	1/4
\vec{x}_4	7/24	5/12	7/24
...
\vec{x}	5/18	4/9	5/18

Фиг. 4

Продължавайки с пресмятанията от (****) и (*****), забелязваме, че разпределението клони към равновесно състояние на $x^{\rightarrow} = (5/18 \ 4/9 \ 5/18)$:

Стойността на PageRank за дадени страници (и косвената подредба сред тях) е независима за всяка една заявка, която потребителят може да направи. От където следва, че стойността на PageRank е независим критерий за статичното качество на всяка уеб страница. От друга страна относителната подредба на страниците, интуитивно трябва да зависи от направените заявки. Поради тази причина, уеб търсачките използват статични показатели за качеството на

страниците (като PageRank), само като един от многото показатели по които оценяват дадена уеб страница по време на заявка.

21.2.3 Тематично оценяване на страница (Topic-specific PageRank)

До тук обсъждахме изчисленията, чрез които потребител прескача от страница в страница. Сега ще разгледаме пренасянето до произволна страница. По този начин ние сме в състоянието да извлечем PageRank стойности, съобразени с конкретните интереси.

Да предположим, че интернет страниците за спорт се намират близо една до друга в уеб графиката. Тогава произволен потребител сърфиращ в мрежата, който често разглежда случайни спортни страници е вероятно да прекарва по-голямата част от времето си в спортните страници, така че равновесното състояние и разпределение на спортните страници да се увеличава.

Да предположим, че нашият сърфист се прехвърля на произволно избрана страница, а не в точно определена. Ние няма да се фокусираме върху събирането на всички спортни страници, достатъчно е само да имаме ненулево подмножество от S спортни страници и тогава прехвърлянето е възможно. Това е възможно чрез ръчно направена директория с такива страници.

При условие, че множеството S е непразно, където $Y \supseteq S$, върху което случайното търсене има определено разпределение, нека означим това PageRank разпределение като π^S . За уеб страници, които не са в Y , ние задаваме на PageRank стойности до нула. Ние наричаме π^S - специфична тема на PageRank. Ние неискаме това прехвърляне да взема всеки сърфист в мрежата и да го доведе до точно определена страница. Прехвърлянето до определените S цели може да бъде произволно.

По този начин, може да се вземат предвид и други теми в областта на науката, политиката и т.н. Всяко едно такова разпределение възлага на всяка уеб страница PageRank стойност в интервала $[0,1]$. Един потребител се интересува от точно определена тема. Това ни дава възможност да имаме предвид различните настройки, за които търсачката знае от каква тема потребителя се интересува. Това може да бъде реалистично, защото търсещия регистрира своите интереси посредством анализ на неговите навици в мрежата.

По същия начин можем да си представим специфични PageRank дистрибуции за всяка една от няколко теми, като например наука, религия, политика и така нататък. Всяка от тези дистрибуции, възлага на всяка уеб страница PageRank стойност в интервала $[0,1]$. За един потребител, който се интересува само от една единствена тема между тези теми, ние можем да се позовем на съответното разпределение на PageRank, когато вкараме и класиране на резултатите от търсения. Това ни дава възможността за разглеждане на настройките, в която търсачката знае от какви теми се интересува даден потребител. Това може да се случи, тъй като потребителите пряко регистрират своите интереси, или защото системата се учи, наблюдавайки поведението на всеки потребител с течение на времето.

Но какво ще стане ако потребителя има смесени интереси? Например, ако се интересува и от спорт, и от политика, ще можем ли да персонализираме PageRank специално за него? Как бихме могли евентуално да изчислим различното разпределение на PageRank за всеки потребителски профил? Въпреки, ние можем да го адресираме, при условие че приемем

индивидуалните интереси на потребителя като линейна комбинация от малък брой специфични разпределения.

Потребител с различни интереси може да избере дали да се прехвърли към S множество със спортни страници или към политическите страници. Това се определя по случаен начин спрямо предпочитанията му в точно определен момент. Ако изборът води до спортните страници, потребителят се прехвърля към множеството S .

Изпълнението на тази идея изглежда първоначално тромаво – иска се изпълнението ѝ за всеки потребител, изиска се изчисляването на матрица на вероятностите и на фиксираното разпределение. Имайки предвид факта, че еволюцията на вероятностното разпределение чрез състоянието на веригата на Марков може да се разглежда като линейна система. Персоналният вектор на PageRank за потребителя, чийто интереси са 60% спорт и 40% политика, може да бъде изчислен като $0.6 \sim \pi_s + 0.4 \sim p$, където π_s и p са съответно базовите PageRank вектори за спорт и за политика.

21.3 Хъбове и авторитети (Hubs and Authorities)

Сега ще разработим схема, при която, всяка уеб страница, отговаряща на дадена заявка, се идентифицира с две оценки. Едната се нарича хъб оценка, а другата – оценка авторитет. За всяко запитване се изчисляват два класирани списъка на резултатите. Класификацията на единния списък се осъществява от хъб оценки, а от другия от авторитетните оценки. Този подход произтича от конкретен поглед върху създаването на уеб страници. В него има два основни вида страници, които са полезни като резултати за широка тема търсения. Под широка тема търсения имаме предвид информационна заявка от вида на „Иска ми се да научавам повече за левкемията“. Има авторитетни източници на информация по темата – в този случай, страницата на Националния институт е такъв. Ние ще наричаме тези страници авторитети. От друга страна, има много страници в мрежата, които са ръчно съставени списъци на връзки към авторитетни уеб страници върху определена тема. Тези хъб страници не са сами по себе си авторитетни източници на информация по съответната тема, а са по-скоро компилация на човек с интерес към темата, който е прекарал известно време събирайки страниците. Въпреки това, ние ще вземем тези хъб страници и чрез тях ще се опитаме да намерим авторитетни такива. При изчисленията, ние развиваме тези хъб страници, които ще се появят с най-високи резултати при търсене.

Една добра хъб страница е онази, която има множество допирни точки с много добри авторитетни страници. Добра авторитетна страница е онази, която има много допирни точки с много хъб страници. Нека предположим, че имаме множество от добри хъб и авторитетни страници заедно с хипер връзките между тях. Ние интерактивно ще изчисляваме хъб и авторитет резултатите за всяка интернет страница в тази група. За интернет страница v в нашето подмножество ще използваме $h(v)$ за да обозначим хъб резултата, $a(v)$ за авторитета. Първоначално имаме $h(v) = a(v) = 1$ за всички възли v . Също така обозначаваме $i \rightarrow j$ ако съществуват хипер връзки от v до j . В основата на интерактивния алгоритъм са новите хъб и авторитет резултати за всички страници получени от уравнението.

$$h(v) \leftarrow J^a(y)$$

$$v^y$$

$b(y)$, което доказва твърдението, че добрите хъб допирни точки водят до добри авторитетни страници и обратното. Първият ред от уравнението показва, че хъб резултът от страницата v е сума от авторитетните резултати за страниците. С други думи ако v води до страници с авторитетни резултати, то хъб резултата расте. За втория ред е обратното – ако v води до добри хъбове, то авторитетните резултати се повишават.

Какво ще се случи ако извършим тази актуализация итеративно? Ако пресметнем отново хъб резултатите тогава ще бъдат ли новите авторитети базирани на съвместимите хъбове? Нека да преобразуваме формулата на вектор – матрица. Нека X и A са вектори от всички хъб и авторитети на страниците в нашия граф.

A е квадратна матрица с един ред и една колона в подмножеството. Входът A_{ij} е един, това е хипервръзка от страница i до страница j и 0 при всеки друг случай. Уравнението вече има вида

$$b \leftarrow A \sim a$$

$$a \leftarrow A^T \sim b,$$

Където A^T е транспонирана матрица на A . Сега дясната страна е вектор и можем да преобразуваме уравнението.

$$b \leftarrow AA^T \sim b$$

$$a \leftarrow A^T a.$$

Ако заменим \leftarrow със $=$ и да въведем стойност на първата част на уравнението се превръща във вектор докато AA^T докато втората част се превръща в A^TA .

$$b = (1/\lambda_b)AA^T \sim b$$

$$a = (1/\lambda_a)A^T A \sim a.$$

Тук използваме λ_b за означаване на стойността AA^T и λ_a за да се обозначи собствена стойност A^TA .

Това води до някои основни последици:

1. Интерактивните актуализации на уравнението са намалени от собствените стойности и са еквивалентни на метода за пресмятане на векторите AA^T и A^TA . Като се вземе предвид, че стойността на AA^T е уникална, интерактивното пресмятане на b и a се определя от уникални стойности, определени при входа на A и от тук структурата на връзките на графа.
2. При изчисляването на тези входни вектори, ние не сме ограничени да използваме метод за итерация и наистина бихме могли да използваме всеки бърз метод за пресмятане на собствения вектор на матрицата.

В резултат на изчисления по този начин се получава следната форма:

- 1 Събираме подмножество от уеб страници, под формата на графиката, включително и техните хипервръзки и изчисления на AAT и AA .
- 2 Изчисляване на основните собствени вектори на AAT за да формират вектор на хъб резултати b , а на авторитетни резултати U .

3 Изходът от анализа на хъбове и авторитети.

HITS Този метод на анализ на връзката е известен като Хитове, което е акроним за хипервръзка - индуцирана на тема търсене.

Пример 21.2: Ако приемем, че заявка за ягуари двойно предаване на връзки, чиито основни съдържат дума, матрицата е А на фигураната 21.4:

0	0	1	0	0	0	0
0	1	1	0	0	0	0
1	0	1	2	0	0	0
0	0	0	1	1	0	0
0	0	0	0	0	1	1
0	0	0	0	0	1	1
0	0	0	2	1	0	1

Хъб и авторитетните вектори са:

$$b = (0.03 \ 0.04 \ 0.33 \ 0.18 \ 0.04 \ 0.04 \ 0.35)$$

$$a = (0.10 \ 0.01 \ 0.12 \ 0.47 \ 0.16 \ 0.01 \ 0.13)$$

Тук, q3 е главния авторитет – два хъба са базирани на тежестта на ягуар – връзки.

От итеративните актуализации интуицията на добрите хъбове и добрити авторитети са високоточувани – страниците би трябвало да ни дават добри хъбове и от подмножеството на уеб страниците.

21.3.1 Изборът на подмножество на уеб

При сглобяване подмножество на уеб страници, касаещо една тема като левкемия, ние трябва да вземем предвид факта, че добрити авторитетни страници не винаги съдържат специфичния термин от заявката. Например, страниците на IBM са авторитетни източници на информация за хардуера на компютъра, въпреки че тези страници може и да не съдържат термина компютър или хардуер. Въпреки това, хъбът може да съдържа компютърни и хардуерни ресурси и е вероятно те да се използват при подобни условия, като в резултат ще ни свържат към съответните страници към уеб сайта на IBM.

Въз основа на тези наблюдения, следната процедура е предложена за съставянето на подмножество на уеб, за които да се изчислят хъб и авторитет резултатите.

1. При заявка (да речем левкемия), използваме индекс – текст, за да получим всички страници, които съдържат левкемия. Наричаме го – базов набор от страници.
2. Изграждане на базов набор от страници, които да включват корена, както и всяка страница или линкове към страница, или връзките на една страница към корена.

След това трябва да използваме определена база от хъб и авторитет резултати. Основния набор от данни е изграден по този начин, по три причини:

1. Хубавата авторитетна страница може да не съдържа текст на заявката (като компютърен хардуер).
2. Ако текстовата заявка успява да улови добри хъб страници на νb в корена, след включването на всички страници, свързани с всяка страница в главния набор от всички добри авторитети, свързани с νb в основния набор.

3. Обратно, ако текстът от заявката успява да улови една добра авторитетна страница *ia* в главния набор, след включването на страниците, които сочат към *ia* ще донесе други добри хъбове в основния набор от данни. С други думи, „разширяване“ на корена в основния набор от обогатяващите хъбове и авторитети.

Изпълняване на НITS през различни запитвания разкрива някои интересни прозрения за анализ. Често документите, които се появяват като топ хъбове и авторитети включват езици, различни от езика на заявката. Тези страници са вероятно въведени в базовата съкупност, след сглобяването на набора от корени. По този начин, някои елементи на извлечане от езика (където заявка на един език извлича документи от друг) са очевидни; интересното е че не се изисква езиков превод.

Коренът се състои от всички страници, съвпадащи с текста от заявката в действителност. За изпълнението се предполага, че е достатъчно да използвате 200 или по-малко уеб страници, за корена, а не всички страници от заявката. Всеки алгоритъм за компютърни вектори може да бъде използван за изчисляване на хъб/авторитет за оценка на вектор. В действителност, не трябва да се изчислят точните стойности на тези резултати. Достатъчно е да се знаят, относителните стойности на оценките, така че да можеда се идентифицират най-добрите хъбове и авторитети. За тази цел е възможно малък брой итерации на метода, да дава относително най-добрите от тях. Експериментите показват, че на практика около пет повторения на уравнението (21.8) дават доста добри резултати. Освен това, тъй като структурата на връзките на уеб графиката е доста рядка, ние не ги изпълняваме като векторна матрица.

Hubs	Authorities
schools	The American School in Japan
LINK Page-13	The Link Page
ú-íšw-Z	%d2e2-§'a'c-Šw=Zfz-[f=fy-[fW
¤a%o-íšw-Zfz-[f=fy-[fW	Kids' Space
100 Schools Home Pages (English)	'Aéé-s-§'Aéé%u-íšw-Z
K-12 from Japan 10...met and Education)	(ééç ášw-@-íšw-Z
http://www...iglobe.ne.jp/~IKESAN	KEIMEI GAKUEN Home Page (Japanese)
J,fj-íšw-ZU'N,Pg'-Œé	Shiranuma Home Page
¤ÖS-—§-ÖS-—Œ-íšw-Z	fuzoku-es.fukui-u.ac.jp
Koulutus ja oppilaitokset	welcome to Misa E&J school
TOYODA HOMEPAGE	=_þ2CE\$E%q n-e-§'t2e-íšw-Z,fy
Education	http://www...p~m_maru/index.html
Cay's Homepage(Japanese)	fukui haruyama-es HomePage
-yí-íšw-Z,fz-[f=fy-[fW	Torisu primary school
UNIVERSITY	goo
%J-—íšw-Z DRAGON97-TOP	Yakumo Elementary,Hokkaido,Japan
¤A%o-íšw-Z,T,N,Pg,fz-[f=fy-[fW	FUZOKU Home Page
¶µ·éÅÅ© ¥å*È*å½¥å*È*å½	Kamishibun Elementary School...

Фигура 21.6

Фигура 21.6 показва резултатите от прегледа на заявката за Япония и началните училища. Цифрата показва най-добрите хъбове и авторитети, всеки ред изброява заглавието – маркер от съответната HTML страница. Тъй като резултата е низ, не е задължително да е на латиница и при печат (в много случаи) е низ от безсмыслици. Всеки от тях отговаря на една уеб страница, която не използва латиница, в този случай е много вероятно страниците да са на японски. Там също изглежда, че страниците не са на английски език, което е неочаквано като се има предвид, че заявката е на английски език. В действителност, този резултат е пример за функциониране на попаденията - след използването на корен, низа на заявката се игнорира. Основният набор от данни е вероятно да съдържа страници на други езици. Например, ако центърът е на английски език, а хъб страницата на японски език, резултатите са страници на японски начални училища. Тъй като последващо изчисляване на най-добрите хъбове и авторитети е изцяло

основана на връзка с някои от тези не-английски страници, те ще се появят сред най-добрите хъбове и авторитети.

УПРАЖНЕНИЕ 21,19

Ако всички хъб и авторитет резултати се инициализират с 1, какво е хъб/авторитет резултат на възел, след едно повторение?

УПРАЖНЕНИЕ 21,20

Как ще тълкувате вписванията на матрици AA^T и A^TA ? Каква е връзката на съвместното наличие на матрица CCT в глава 18?

УПРАЖНЕНИЕ 21,21

Какви са основните собствени стойности на AA^T и на A^TA ?

22 Анализ на мнения

22.1 УВОД

Текстовата информация може да бъде класифицирана на две големи групи: факти и мнения. Фактите са обективни изрази за лица, събития и техните свойства. Мненията обикновенно са субективни изрази, които описват настроения, оценки или чувства към лица, събития и техните свойства. Понятието мнение е много широко. Тук ще се съсредоточим върху общественото мнение, което предава положителното или отрицателното настроение на хората спрямо даден обект.

Голяма част от изследователската дейност върху текстовата обработка на информация е фокусирана върху намирането и извличането на фактическа информация: например уебтърсенето, класификация и кълстерилизация на текст, както и други дейности по обработката на естествен език. Доскоро беше направено много малко по обработката на мнения. И все пак, мненията изглежда са важни, тъй като всеки път, когато трябва да вземем решение, ние искаем да чуем мнението на другите. Това е вярно не само за физическите лица, но също така и за организацията.

Една от причините за липсата на проучвания върху мненията е факта, че имаше малко текстове с мнения преди World Wide Web. Преди популяризирането на Web, когато дадено лице трябваше да вземе решение, обикновенно се налагаше да иска мнения от приятелите и семейството си. Когато една организация искаше да разбере становището или мнението на широката общественост за своите продукти и услуги, трябваше да направи проучване на общественото мнение. Но с появата на Web светът се промени. Мрежата промени коренно начина, по който хората изразяват своите възгледи и мнения. Днес те могат да публикуват мнения за продуктите, предлагани в търговските обекти и да изразят становището си за почти всичко в интернет форумите, дискусионните групи и блоговете. Сега, ако някой иска да закупи даден продукт, вече не е ограничен с мненията само на своите приятели и семейство, тъй като има много мнения за продуктите в интернет. За една компания, вече не е необходимо да провежда изследвания и проучвания върху целеви групи, да наема външни консултанти, за да разбере мнението на потребителите за своите продукти, както и становището на своите конкуренти, защото генерираното от потребителите съдържание в интернет вече може да им даде тази информация.

Въпреки това, намирането на източници на мнения и техния мониторинг в интернет, все още е доста трудна задача, защото има голям брой различни източници и всеки източник може да съдържа огромен брой мнения. В много случаи мненията са скрити в дълги текстове, в големи блогове и форуми. Трудно е за един читател да намери подходящите източници, да извлече съдържанието се в тях мнения, да ги прочете и обобщи и да ги организира в използваеми форми.

Така се появява необходимостта от автоматизирано откриване на мнения и системи за обобщение. Появява се необходимост от Sentiment analysis, известен също и като извличане на мнения. Това може да се приеме за едно предизвикателство към обработката на естествен език или за един проблем в извлечането на текстова информация. Поради огромната стойност за

практическите приложения, налице е значително нарастване на изследванията в тези две области в академичните среди, както и в приложението им в индустрията. Днес има най-малко 20-30 компании, които предлагат услуги за извлечане на мнения. Само в САЩ.

Основните аспекти в изследването на Sentiment analysis:

- **Проблеми на Sentiment analysis** - Както е за всеки научен проблем, преди да бъде решен, той трябва да бъде определен и формализиран. Формулировката ще въведе основните определения и понятия, основните въпроси, подпроблеми и цели. Тя също така служи като обща рамка за обединението на различните изследователски направления. От гледна точка на потребителите, тя казва какви са основните задачи, техните входове и възможни изходи, и как резултатът може да се използва в практиката.
- **Класификация на мнения и субективностна класификация** - Това е област, която се проучва основно в академичните среди. Тя третира sentiment analysis като проблем на класификацията на текст. Две подтеми, които са били широко проучени са: (1) Класификация на текст, съдържащ мнения. Изразяване на положително или отрицателно мнение; (2) Класификация на едно изречение като обективно или субективно. За обективните изречения (мнения) се извършва класификация на положителни, отрицателни или неутрални. Например, при преглед на даден продукт, се определя дали преглеждащия е положително или отрицателно настроен към продукта. Втората част се отнася за отделни изречения, за това да се определи дали едно изречение изразява мнение или не (често се нарича субективностна класификация), и ако е така, дали мнението е положително или отрицателно (ниво на sentiment analysis).
- **Базиран на характеристики анализ на мнения** - Този анализ първоначално цели откриване на мнението, които са изразени в едно изречение, а след това определя дали мнението са положителни, отрицателни или неутрални. Целите са обектите, техните компоненти, атрибути и функции. Един обект може да бъде продукт, услуга, лице, организация, събитие, тема и т.н. Например в едно изречение, отнасящо се за даден продукт, идентифицираме характеристиките на продукта, които са били коментирани и определяме дали коментарите са били положителни или отрицателни. Например, в изречението „Животът на батерията на тази камера е твърде кратък.”, коментарът е за „живота на батерията”, а обекта е камерата и становището е отрицателно. Много реални приложения изискват това ниво на подробен анализ, защото, за да се направят продуктови подобрения, трябва да се знае кои компоненти и характеристики на продуктите са харесвани и кои – не, от потребителите. Тази информация не е част от откриването на мнения или класификацията за субективност.
- **Sentiment analysis на изречения със сравнение** - Оценката на даден обект може да се направи по два основни начина – пряка оценка и сравнение. Директната оценка, наречена директно мнение, дава положително или отрицателно мнение за обекта, без да се споменават други подобни обекти. Сравнението означава да се сравни обекта с някои други подобни обекти (напр. конкурентни продукти). Например, изречението „Качеството на картината на тази камера е лошо” изразява пряко мнение, докато изречението „Качеството на картината на тази камера е по-добро от това на камера X”

– изразява сравнение. Ясно е, че е полезно да се идентифицират такива изречения, да се извлекат сравнителните мнения, изразени в тях и да се определи кои обекти са предпочитани от автора на изречението (в горния пример Камера X не се предпочита по отношение на качеството на картина).

- **Търсене и извлечане на мнения** - След като стандартното търсене в мрежата е толкова успешно в много аспекти, не е трудно да предположим, че търсенето на мнения също ще бъде много полезно и успешно. Например, при дадена ключова дума за заявка „аборти”, ние искаме да намерим положителни и отрицателни мнения по въпроса от търсачката на мнения. За такава заявка трябва да бъдат решени 2 задачи: (1) Извличане на документи или изречения, които имат значение за заявката, (2) Идентифициране и класифициране на документите или изреченията с мнения от тези извадки. Така търсенето на мнения е комбинация от извлечане на информация и sentiment analysis.
- **Спам – мнения и полезност на мненията** - Тъй като мненията в мрежата имат голямо практическо приложение, не е изненада, че хората са започнали да злоупотребяват с тях. Спам-мненията се отнасят до фалшиви мнения, които се опитват умислено да заблудят читателите или автоматизираните системи. Това става чрез даване на незаслужени положителни мнения за някои целеви обекти с цел подкрепа за обекта и/или чрез даване на злонамерени отрицателни мнения за други обекти с цел да навреди на репутацията им. Откриването на такъв спам е много важно за приложенията. Програмите за мнения имат и приложения, които оценяват полезността или качеството на мненията. Автоматичното задаване на полезни, оценяващи стойности за мненията е полезно, тъй като мненията могат да бъдат класирани въз основа на стойностите на тяхната полезност. С класирането читателят може да се съсредоточи само върху качествените мнения. Трябва да се отбележи обаче, че спам и полезност са много различни понятия.

22.2 Проблеми в анализа на чувства (Sentiment analysis)

Sentiment analysis или извлечането на мнения е компютърно изследване на мнения, чувства и емоции, изразени в текста. Нека да използваме следния сегмент от преглед на iPhone, за да въведем проблема (асоциираме на всяко изречение номер за по-лесна справка):

- (1) Купих си iPhone преди няколко дни.
- (2) Това е много хубав телефон.
- (3) Сензорният еcran наистина е страховчен.
- (4) Качеството на звука също е много добро.
- (5) Въпреки, че животът на батерията не е дълъг, все пак съм доволен от телефона.
- (6) Въпреки това, майка ми беше много ядосана, тъй като не ѝ казах, преди да го купя.
- (7) Тя каза също, че телефонът е твърде скъп и искаше да го върна в магазина.

Въпросът е: какво искаме да извлечем от тези изречения? Първото нещо, което може да забележите е, че има няколко мнения в тях. Изречения (2),(3),(4) изразяват положителни мнения, а изречения (5),(6),(7) изразяват отрицателни мнения и емоции. Освен това, може да се забележи, че в изреченията има няколко обекта или предмета, за които са изразени мнения.

Мнението в изречение (2) е за iPhone като цяло, а мнениета в изречения (3),(4),(5) са за „сензорния еcran”, „качество на звука” „животът на батерията” на iPhone. Мнението в изречение (7) е за цената на iPhone, а мнението, емоцията в изречение (6) няма връзка с iPhone. Това е важно уточнение. В дадено приложение потребителят може да се интересува от мнения за точно определени предмети или обекти, а не за всички предмети в текста. И накрая, да обърнем внимание на притежателите на мнениета. Притежателят на мнениета в изречения (2),(3),(4),(5) е авторът на самия преглед, а в изречения (6),(7) притежател на мнениета е „майка ми”.

По принцип мнение може да се изрази за нещо, например за даден продукт, услуга, физическо лице, организация, събитие или тема. Ние използваме термина „обект”, за да обозначим целта на коментара. Обектът може да има набор от компоненти (части) и набор от атрибути (свойства).

Всеки компонент може да има свои подкомпоненти и комплекс от атрибути и т.н. По този начин обекта може да се разглежда йерархично, въз основа на връзката между частите му. Формално имаме следното:

ДЕФИНИЦИЯ (обект): Един обект е единица, която може да бъде даден продукт, лице, събитие, организация, или тема. Той е свързан с двойката $o(T,A)$, където T е йерархията на компонентите (или частите), подкомпонентите и т.н., а A е набора от атрибути на o . Всеки компонент има свой собствен набор от подкомпоненти и атрибути.

Пример: Определена марка мобилен телефон е обект. Той разполага с набор от компоненти – например батерия, еcran ..., а също и набор от атрибути като качество на звука, размер, тегло. Компонента батерия също има набор от атрибути, като например „живот на батерията”, както и размер на батерията.

Въз основа на това определение, обектът може да бъде представен като дърво, йерархия или таксономия. Коренът на дървото е самият обект. Всеки некоренов възел е негов компонент или подкомпонент. Всяко ребро изразява връзка между компоненти или подкомпоненти. Всеки възел се асоциира също с набор от атрибути или свойства. Мнение може да бъде изразено за всеки възел и всеки атрибут на възел.

Пример 2: В пример 1 може да се изрази мнение за клетъчния телефон като цяло (коренът), например “Аз не харесвам този телефон” или за някой от неговите атрибути “Качеството на звука на този телефон е гадно”. Също така може да се изрази мнение за всеки един от компонентите на телефона или за някой атрибут на компонент.

22.3 Класификация на мнения и субективностна класификация

Ще се насочим към някои основни теми в научните изследвания в областта на Sentiment analysis. Една от тях е класификацията на документи, съдържащи мнения (например преглед на някой продукт), които изразяват положително или отрицателно мнение. Задачата е известна като класификация на мнения на ниво документи, тъй като се счита, че целия документ е една базова единица за информация. В съществуващите научни изследвания се приема по презумция, че документът съдържа мнения. Естествено същата класификация на мнения може да се приложи върху отделни изречения. Но в този случай, не се приема, че отделните

изречения съдържат мнения. Задачата за определяне на това, дали дадено изречение изразява мнение, се нарича субективностна класификация. Резултатните изречения, за които вече се знае, че съдържат мнения, се подлагат на класификация и се класифицират като изразявани положителни или отрицателни становища. Това се нарича класификация на мнения на ниво изречения.

22.3.1 Класификация на мнения на ниво документи

При дадено множество от документи с мнения D , той определя дали документа $d \in D$ изразява положително или отрицателно становище за даден обект. Формално погледнато:

Задача: Даден е документ с мнения d , който коментира обект o . Трябва да се определи ориентацията oo на мнението, изразено за o , т.е. да се определи ориентацията oo за функцията в петорката (o, f, so, h, t) , където $f=o$, а h, t, so са известни или са без значение. Съществуващите изследвания в областа на анализа на мнения правят следното предположение:

Предположение: Даден документ с мнения (например потребителски преглед на продукт) изразява мнения само за един обект o и мнението са от един точно определен притежател h .

Това предположение се отнася и за потребителските мнения за продукти и услуги. Въпреки това предположението не може да се отнесе за форуми или постове в блогове, защото в такива постове авторите могат да изразяват мнения за няколко продукта и да ги сравняват, да използват сравнителни и превъзходни изречения. Повечето от съществуващите техники за класификация на мнения на ниво документи се основават на обучението с учител, въпреки че има и други допустими техники.

Класификацията на мнения очевидно може да бъде формулирана като проблем на обучението с учител, което се състои от 2 класа (положително и отрицателно мнение). Обучението и тестовите данни, използвани в съществуващите изследвания, са извлечени най-вече от потребителски прегледи за даден продукт, което не е изненадващо. Тъй като всеки преглед в един типичен рекламен сайт вече има целеви рейтинг (например 1-5 звезди), обучението и тестовите данни са лесно достъпни. Обикновено, когато се гласува с 4-5 звезди, се счита за положителна оценка, а когато се гласува с 1-2 звезди, се счита за отрицателна оценка (или съответно палец нагоре или надолу).

Класификацията на мнения е подобна, но също се и различава от класическата базирана на теми класификация на текст, която класифицира документите в предварително дефинирани класове по теми, например политика, наука, спорт и др. В класификацията по теми са важни свързаните с определена тема думи. В класификацията по мнения думите, свързани с темата не са важни. Вместо това са важни думите, които показват мнение, отношение, положително или отрицателно становище, например: чудесно, невероятно, отлично, ужасно, лошо, най-лошото и т.н.

Съществуващите методи за обучение с учител могат лесно да се приложат в анализа на мнения, например Наивен Бейсов класификатор, Support Vector Machines (SVM) и др. Панг използва този подход, за да класифицира филмови рецензии в два класа – положителни и отрицателни. Показано бе, че чрез използване на unigrams (торба с индивидуални думи) като функция за класификация, те се представят добре както с Наивен Бейсов класификатор, така и с SVM.

Неутралните мнения не се включват в изследванията, което до известна степен улеснява проблема.

22.3.2 Базиран на характеристики анализ на мнения

Въпреки че класифицирането на текстове с мнения на ниво документи или на ниво изречения е полезно, в много от случаите то не предоставя необходимата детайлност, от която се нуждаят някои приложения. Положително мнение в документ за даден обект не означава, че авторът има положително становище за всички характеристики и аспекти на обекта. По същия начин отрицателно мнение, изразено в документ не означава, че авторът не харесва нищо в обекта. В един типичен текст с мнения авторът описва положителните и отрицателните аспекти на обекта, въпреки че общата нагласа към определения обект може да е положителна или отрицателна. На ниво документи и на ниво изречения класификацията не предоставя такава информация. За да получим такива детайли, трябва да отидем на ниво характеристики на обекта. Ще се фокусираме върху две основни задачи:

Определяне на обектните характеристики, които са били коментирани. Например, в изречението „Качеството на картина е невероятно“, характеристиката на обекта е „качество на картина“.

Определя се дали мнението за характеристики на обекта са положителни, отрицателни или неутрални. В горния пример мнението за характеристиката „качество на картина“ е положително.

Определяне на притежател на мнение, обект и време:

В някои приложения е полезно да се идентифицират и да се определят притежателите на мнението, т.е. лицата или организациите, които изразяват определени становища.

Притежателите на мнения са по – лесно определими в статиите с новини или други видове официални документи, в които лицата или организациите, изразяващи становищата, се посочват изрично в текста. Тези притежатели трябва да бъдат идентифицирани от системата. В случаите на потребителски генерирано съдържание в интернет, притежателите на мнения често са авторите на дискусионните постове, блогъри или рецензенти, при чието влизане в системата, самоличността се установява, въпреки че истинската им идентичност в реалния свят може да бъде неизвестна.

Извличането на името на обекта е по – лесно в дискусионни форуми, блогове и потребителски прегледи. Имайте предвид, че въпреки фокусирането на мнението и оценката върху даден обект, той също така може да бъде сравнен с други конкурентни обекти.

Извличането на време също е лесно в контекста на уеб, тий като всеки уеб сайт обикновено показва времето, когато е подадена всяка публикация, така че извлечането му е лесно. В новини и други видове документи извлечането на време също не е проблем. Общо тези три задачи за извлечение са известни като „разпознаване на имена на обекти“. Те са били широко изследвани.

Кореферентностна резолюция: В отзивите за продукти преглежданите обекти обикновено са известни. Но това не е така за мнението, изразени в блогове или дискусионни постове.

Например в поста „Аз имам Canon S50, закупен от Amazon. Той прави страхотни снимки.“ Могат да бъдат зададени два въпроса:

- (1) Кой обект се коментира в поста?
- (2) Кой е обекта, означен с „той“ във второто изречение?

За нас като хора е ясно, че поста се отнася за Canon S50, който е обект на проблема за извлечане, обсъден по-горе. Ние също знаем, че „той“ означава Canon S50, което всъщност изразява проблема за кореферентностна резолюция. Кореферентностната резолюция е проучена подробно в NLP (Обработка на естествен език). Въпреки това си остава едно голямо предизвикателство.

22.4 Sentiment analysis на сравнителни изречения

Директното изразяване на положителни или отрецателни становища за обекти или някакви техни характеристики е само една от формите за оценка. Сравнението на даден обект с някои други подобни обекти е друга форма за израз на нагласа или оценка. Сравненията са близки, но също така и доста се различават от преките становища и мнения. Те не само имат различно семантично значение, но също така имат и различни синтактични форми. Например, един типичен пряк израз на мнение е „Качеството на картиината на тази камера е страхотно“, а едно съответно сравнително изречение би било „Качеството на картиината на камера X е по-добро от това на камера Y“.

22.4.1 Дефиниция на проблема

Като цяло, сравнителното изречение изразява отношение, на базата на прилики и разлики между два или повече обекти. Сравнението обикновено се изразява с помощта на сравнителна или превъзходна степен под формата на прилагателно или наречие. Сравнителната степен се използва, за да се заяви, че един обект има повече в определено количество от друг обект. А превъзходната степен се използва, за да се заяви, че даден обект има най-много или най-малко в определено количество.

Като цяло, сравнението може да бъде между два или повече обекти, групи обекти, както и между един обект и останалите от групата. Сравнението може да бъде също между по-новите или по-старите версии на един и същ обект.

22.4.2 Тип на сравнителните релации

Сравнителните релации и отношения могат да бъдат групирани в 4 основни типа. Първите три типа се наричат ранжирани, а последния е неранжирано сравнение.

1. Неравностойни ранжирани сравнения: Отношения от типа „по-голямо“ или „по-малко от“, изрично подреждане на някакви обекти по отношение на определени техни характеристики. Например „Чипът на Intel е по-бърз от този на AMD“. Този тип сравнения могат да включват и предпочитания на потребителите, например „Предпочитам Intel пред AMD“.
2. Равностойни сравнения: Отношения от типа „равни по“. Тук двета обекта се изравняват по отношение на определени техни характеристики, например „Качеството на картиината на камера X е толкова добро, колкото това на камера Y“.

3. Превъзходни сравнения: Отношения от типа „по-голямо“ или „по-малко от всички“, като се сравнява дадения обект спрямо всички други. Например „Чипа на Intel е най-бърз“.
4. Неранжирани сравнения: Отношения, които сравняват характеристиките на два или повече обекта, но не ги подреждат по ранг. Има три основни подтипа:
 - Обекта А е близък или различен от обекта Б по отношение на някои характеристики. Например „Кафето е различно от „Пепси“.
 - Обекта А има характеристика f1, а обекта Б има характеристика f2 (f1 и f2 обикновено са заменими). Например „Настолните персонални компютри използват външни високоговорители, а лаптопите имат вградени високоговорители“.
 - Обекта А има характеристика f1, която обекта Б не притежава. Например „Мобилните телефони X имат слушалки, а мобилните телефони Y – нямат“.

Въпреки че повечето сравнителни изречения съдържат сравнителни прилагателни и срвнителни наречия, например по-добре, по –голям, по-бърз и т.н., много изречения, които съдържат тези думи не са сравнителни, например „Не мога да продължа по-нагоре“. По същия начин много изречения, които не съдържат такива показатели, са сравнителни изречения (обикновено с ранжиране), например „Мобилният телефон X има Bluethooth, а мобилният телефон Y – няма“.

Интересно явление при сравнителните изречения е, че такова изречение обикновено има ключова дума или ключова фраза показваща сравнение. Ключови думи и ключови фрази са :

1. Сравнителни прилагателни и сравнителни наречия. Например: по – малко, по – добри и други, започващи с „по-“,
2. Превъзходни прилагателни и превъзходни наречия. Например: най – добър, най – точен и други, започващи с „най-“.
3. Други показателни думи като: подобни, различават се, превъзхожда, предпочитат, номер едно, надхвърля, побеждава и т.н.

Тъй като ключовите думи сами по себе си могат да постигнат добър recall (добро извлечане) може да се използва филтър за този набор от ключови думи, който е по – малко вероятно да участва в сравнителни изречения. След тази операция се подобрява precision (точността) на останалите изречения.

22.4.3 Извличане на обекти и характеристики на обекти в сравнителни изречения

За да се извлекат обектите и характеристиките на обектите, които се сравняват, могат да бъдат приложени много методи за извлечане на информация, например Условни случаенни полета, скрити модели на Марков и др. Jindal и Liu използват етикети с последователни правила (LSP) и CRF за извършване на извлечането. Алгоритъмът прави следните предположения:

- Има само една сравнителна връзка в едно изречение. На практика това се нарушава само в много малък брой случай.
- Обектите и техните свойства са съществителни и местоимения. Те обхващат повечето случаи.

- Идентифициране на предпочитаните обекти в сравнителни изречения

Подобно на обикновенния Sentiment analysis, при анализа на сравнителни изречения също е необходимо да се определи, дали изречението изразява лично мнение или не. Въпреки това, за разлика от нормалните изречения, не е за предпочитане да се прилага класификация на мнения върху сравнителни изречения, защото сравнителното изречение не изразява пряко положително или отрицателно мнение. Вместо това в него се сравняват множество обекти, като те се ранжират според общите им характеристики, за да се даде сравнително мнение. С други думи, предпочитаните обекти се представят, въз основа на сравнение на някои от общите им характеристики. Тъй като повечето сравнителни изречения сравняват само два обекта, анализа на сравнителното изречение се състои в идентифициране на предпочитания обект.

22.5 Търсене и извличане на мнения

Както търсенето в уеб се оказава изключително важно и полезно, не е трудно да предположим, че търсенето на мнения също ще е важно. Човек има възможност да обхожда потребителски генерираното съдържание в интернет и да търси мнения за всеки един продукт или марка. Могат да бъдат използвани два типични вида заявки за търсенето на мнения:

- Търсене на общественото мнение за даден обект или характеристика на обект. Например, да намерите мнения на клиенти относно някаква марка камери или за качеството на картината на камерите, или общественото мнение по някаква политическа тема. Напомням, че един „обект“ може да бъде продукт, марка, лице, организация, събитие или тема.
- Намиране на мнение на лице или организация (титуляр на мнението) за даден обект или характеристика на обект. Например мнението на Барак Обама относно абпорта. Този тип търсене е от особено значение за новините, където са изрично посочени лицата или организацията, които изразяват мнението.

За първия тип заявки потребителят може просто да даде името на обекта или името на характеристиката и името на обекта. За втория тип заявки потребителят може да посочи притежателят на мнението (титуляра, от чието мнение се интересува) и обекта (за какво или за кого е мнението).

Подобно на традиционното уеб търсене, търсенето на мнение също има две основни задачи:

- (1) Извличане на съответните на заявката на потребителя документи/изречения.
- (2) Класиране на извлечените документи и изречения.

Въпреки това съществуват и големи разлики. При търсенето и извличането на мнения трябва да се изпълняват две подзадачи:

- Търсене на документи/изречения, които имат отношение към темата на заявката. Това е единствената задача, която се изпълнява в традиционното уеб търсене и извличането на информация.
- Определяне дали документите/изреченията изразяват мнения и дали те са положителни или отрицателни. Това е задача от Sentiment analysis. Традиционното

търсене не изпълнява тази задача. Това е и подзадачата, която прави търсенето на мнения по-сложно в сравнение с традиционното търсене.

Що се отнася до класирането, традиционните уеб търсачки класират уеб страниците по авторитет и релевантност. Основната предпоставка е, че най-високо класираните страници (в идеалния случай първата страница) съдържа достатъчно информация, за да задоволи нуждата от информация на потребителя. Тази парадигма е подходяща за търсенето на факти, защото един факт се равнява на произволен брой описания на този факт. Така, че ако първата страница съдържа необходимата информация, няма нужда да гледате останалите страници. При търсенето на мнения, тази парадигма е добра при втория тип заявки, защото даден притежател на мнение обикновено има само едно становище за даден обект или тема и становището се съдържа в един единствен документ или страница. За първия тип заявки тази парадигма трябва да се промени, тъй като класирането в търсенето на мнения има две цели.

Първо, трябва да се подредят документите или изреченията с висока степен на информация в съдържанието, най-отгоре.

Второ, трябва да се отрази естественото разпределение на положителни и отрицателни становища.

Тази втора цел е важна, защото в повечето практически приложения реалните пропорции на положителни и отрицателни мнения са най-важните части от информацията, както е и в традиционното проучване на мнения. Четенето само на най – високо класираните резултати, както при традиционното търсене, би създало проблеми, тъй като едно мнение не е равно на множество мнения. Горния (най - високия) резултат представлява становището само на едно лице или организация. Така, класирането в търсенето на мнения трябва да улови естественото разпределение на положителните и отрицателните настроения на цялото население. Едно просто решение е да се съзладат две класации, по една за положителните и отрицателните становища. Броят на положителните и отрицателните мнения показва разпределението.

23 СЪЗДАВАНЕ НА КОМПЮТЪРНИ РЕЧНИЦИ.

(ИЗВЛИЧАНЕ НА ИНФОРМАЦИЯ ОТ КОРПУСИ)

23.1 Увод

В настоящата работа ще разгледаме речниците като част от компютърната обработка на естествения език. Техния процес на изграждане и осъвременяване, както и връзката им с корпусната и компютърна лингвистика. Ще се даде оглед на това как съвременният дигитален свят променя традиционната лексикографска практика и я обвързва с новите тенденции в лингвистиката и информационните технологии.

23.2 Кратко развитие на речниците през годините. Автоматизиране на лексикографските практики.

Речникът, както всяка една книга или текст, до началото на дигиталната епоха „пребивава“ в човешкия свят само на хартиен носител. Големите речници от минали векове, като например Оксфордския речник на английски език - Oxford English Dictionary (Murray et.al.1928), са създавани от лексикографите на ръка, като са използвали фишове, след което са набирани и отпечатвани. Практиката е даден речник да се основава на данните от действителния език, като се събира автентичен материал под формата на примери от реални текстове. Така са се формирали „корпуси“ от ръкописни фишове с илюстративни примери от реалната употреба на езика. Това е изключително трудоемка задача, която е противчала десетилетия наред и се е реализирала от хиляди доброволци.

Имайки предвид, че това е само една част от процеса на създаването на речник, търсенето на подходи и начини за улеснение винаги е било налице. Така, постепенно, се забелязва потенциала на компютрите за улесняване и рационализиране на събирането, съхранението и обработката на речниковите материали. Те постепенно започват да се конструират като бази от данни, където всеки компонент има свое поле. Компютрите също така спомагат и за по-систематичната проверка и обработка на препратките в речниците.

С времето все повече навлиза и се прилага корпусно базираната методология при създаването на речници, като с хода на все по-голямата достъпност на компютрите и информационните технологии нарастват и самите корпуси, като също се подобряват и системите за търсене в тях, които позволяват на лексикографите да извлечат подходящи данни по-ефикасно. Корпусите биват анотирани посредством токънизация, лематизация и тагиране на частите на речта. Това позволява по-прецизни и фокусирани заявки за търсене.

В сравнение с ръчното въвеждане на речникови статии новите системи за създаване на речници улесняват значително работата. Когато се използва такъв вид софтуер, цялостният процес на въвеждане на речникова статия се контролира от абстрактния модел – „граматиката“ на речника. По този начин на практика се прилагат решението за структурата на речника – как да бъде класифицирано и представено разнообразието от лексикографски факти. Някои типове данни като бележки за стила, принадлежност към част на речта и др. имат крайно

множество от възможни стойности, които могат да бъдат представени във формата на списъци за избор. По този начин лексикографите не трябва да помнят дали определена стойност трябва да е написана получерно или в курсив, какво е приетото съкращение за разговорна употреба – „раз.”, „разг.”, „разгов.” и т.н. Добрата система за създаване на речници също така улеснява редакторската работа. Например редакторът често може да иска да преструктурира дълги речникови статии, променяйки реда на значенията и другите компоненти. Конфигурирането на системите за създаване на речници по подходящ начин е сложна задача, но в резултат работата се улеснява в значителна степен. Някои съществени, но рутинни проверки – за препратките в двете посоки, за речниковата съвместимост и т.н. – са напълно автоматизирани, осъществяват се в процеса на създаване на речника с малка или без човешка намеса.

Процесът на автоматизиране на лексикографската работа е развиващ се и с годините се постигат все по-големи резултати. В днешно време компютърните технологии и интернет оказват силно влияние, а компютърната обработка на естествения език спомага както работата на специалистите, така и нуждите на останалите потребители, погледнато от два различни ъгъла.

23.3 Взаимодействието корпус – речник

От споменатото дотук стана ясно, че употребата на анотиран корпус е от основно значение за създаването на речници. От корпусите могат да се извлекат не само примери, които да бъдат достоверни за реалната употреба на думите, но също така и важни лингвистични данни. Такива, които онагледяват семантичните признаци на лексикалните единици, тяхната синтактична структура (аргументна структура), участието им в словосъчетания, различната употреба в разнообразни контекстови рамки, и т.н. Също така информацията, която се извлича спомага и за формулирането на подходящи дефиниции, които са важна част при превод от един език на друг и намирането на преводни еквиваленти. Освен това все повече се работи и върху лингвистично-правилното извлечане на лемите, т.е. основната форма на думите, от употребените словоформи в дадените текстови комплекси. Това е т. нар. процес на лематизация, който е важен при създаването на речници, тъй като в тях всяка дума е представена на първо място в своята основна форма (лема). В компютърната обработка на корпуси за автоматичен процес на лематизация се използват правила, които обаче не винаги, а и особено при по-богати и специфични езици, дават точни резултати, т.е. възможно е в списъка от леми, който се извлича да има голям набор от грешки. Тези правила се конструират от специалисти и са строго индивидуални за всеки език. Докато при един от най-широко разпространените езици, английския, лематизацията е сравнително лесна и опростена, при много други езици богатият набор от словоформи е предпоставка за по-дълбоко и детайлно анализиране.

Например повечето глаголи на английски език имат само четири форми, от които свеждането им до лемата би могло да стане с правила от следния тип:

лема – hope
форми на –ed / –s (hoped, hopes) → правило: изтрий –d / –s
форма на –ing (hoping) → правило: изтрий –ing, добави –e

При испанския език например алгоритъмът е съвсем друг с оглед на глаголите, например, които се представят в речника в своята инфинитивна форма, и следователно е нужно обработката на думата да доведе до нейния корен, към който да се прибави едно от трите окончания за инфинитив (за съответните три спрежения). Тук списъкът с правила би бил значително по-дълъг, тъй като испанският език, подобно на българския, е „богато“ флексивен (т.e. голям брой окончания за лице и число при всяко глаголно време, при глаголите).

Но връщайки се към идеята за взаимодействието корпус – речник, трябва да споменем и обратния път, т.e. връзката речник → корпус. След като се създава един компютърен (електронен) речник, и в частност такъв с граматическа насоченост, носещ морфологична, синтактична и/или семантична информация), той впоследствие се превръща в един потенциален участник в процеса на анотирането на нови текстове. Важна роля играе при „стъпката“ на тагиране, където подава достоверна информация за типа част на речта и съответните граматически характеристики.

23.4 Изготвяне на електронен речник с XML

XML (*eXtensible Markup Language*) е един от основните типове маркиращи езици, който се утвърди и зае своето място в съвременния оглед на информационните и компютърни технологии. Това е един от предпочитаните средства за съставяне на електронни речници. Тук ще покажем извадка от едно примерно представяне на думи от българския език в XML формат със съответните нужни граматични характеристики. Ще се спрем на глаголите, тъй като те са основополагащо ядро в езика и вербалната система е богата с лингвистични данни.

Глаголите се характеризират със своята предикатно-аргументна структура. Тя представя какви аргументи притежава всеки предикат, т.e. каква е задължителната експлицитна или имплицитна съчетаемост с други езикови единици и каква е тяхната съответна семантична роля.

По-горе споменахме ролята на т. нар. абстрактен модел („граматика“ на речника), който дефинира и контролира структурата и съдържанието на речниковата статия. Тя спомага за изграждането на обща рамка за съставянето на речниците, която спомага за ограничаване на възможни грешки.

При въвеждането на речниковата статия в XML формат една такава роля на абстрактен модел играе DTD-то (*Document Type Definition*) свързано с дадения XML файл. В него се описват всички възможни елементи и атрибути и съответните им характеристики, които могат да фигурират в XML файла. По този начин се стандартизира модела за всеки, който участва в изграждането на речника (тъй като почти винаги това е екипна работа) и се сигнализира при въвеждането на недефинирани данни или такива с различно означение (като гореспоменатия пример с вариантите за изписване на „разговорна употреба“).

Примерно DTD за описание на предикати. Лема – характеристики. Аргументна структура.

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <!ELEMENT dictionary (word)+ >
3  <!ELEMENT word (synt)+ >
4  <!ATTLIST word
5      lemma CDATA #REQUIRED
6      pos (V|N|A) #FIXED "V"
7      aspect (cs|hecs|nby) #REQUIRED
8      tr (tr|intr) #REQUIRED
9      refi CDATA #IMPLIED
10     >
11   <!ELEMENT synt (ArgStr+) >
12   <!ELEMENT ArgStr (def, frame+) >
13   <!ELEMENT def (#PCDATA) >
14   <!ATTLIST def
15       type (state|activity|achievement|accomplishment) #REQUIRED>
16   <!ELEMENT frame (((NP|CP|pro)+,(NP|AP|CP|PP)*, (AdvP|CP)*), example+) >
17   <!ELEMENT NP EMPTY>
18   <!ATTLIST NP
19       function (s|do|io|p|a) #REQUIRED
20       SemRole (agent|force|instr|exp|loc|theme|patient) #REQUIRED
21       expl (y|n) #REQUIRED>
22   <!ELEMENT CP EMPTY>
23   <!-- zakusvan prepecheni filiyki -> not a direct object-->
24   <!ATTLIST CP
25       function (s|do|io|p|a) #REQUIRED
26       SemRole (agent|force|instr|exp|loc|theme|patient) #REQUIRED
27       expl (y|n) #REQUIRED>
28   <!ELEMENT pro EMPTY>
29   <!ELEMENT AP EMPTY>
30   <!ATTLIST AP
31       function (p) #REQUIRED
32       expl (y|n) #REQUIRED>
33   <!ELEMENT PP EMPTY>
34   <!ATTLIST PP
35       function (io|p|a) #REQUIRED
36       SemRole (agent|force|instr|exp|loc|theme|patient) #REQUIRED
37       expl (y|n) #REQUIRED>
38   <!ELEMENT AdvP EMPTY>
39   <!ATTLIST AdvP
40       function (a) #REQUIRED
41       SemRole (loc) #REQUIRED
42       expl (y|n) #REQUIRED>
43   <!ELEMENT example (#PCDATA|a|p)*>
44   <!ELEMENT a (#PCDATA) >
45   <!ATTLIST a
46       id (agent|force|instr|exp|loc|theme|patient|pro) #REQUIRED>
47   <!ELEMENT p (#PCDATA) >

```

Глаголна парадигма. Морфологични характеристики.

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <!ELEMENT dictionary (entry)+>
3  <!ELEMENT entry (pf+,if+,ger?,noun?)>
4  <!ATTLIST entry lemma CDATA #REQUIRED>
5  <!ELEMENT pf (#PCDATA)>
6  <!ATTLIST pf
7      t (p|a|i) #REQUIRED
8      p (1|2|3) #REQUIRED
9      n (sg|pl) #REQUIRED>
10 <!ELEMENT if (#PCDATA) >
11 <!ATTLIST if
12     type (сд|мсд|мнсд|мстп|дееп) #REQUIRED
13     n (sg|pl) #REQUIRED
14     g (m|f|n|all) #REQUIRED
15     d (y|n|yy) #IMPLIED
16     >
17 <!ELEMENT ger (#PCDATA) >
18 <!ELEMENT noun (#PCDATA) >
19

```

Извадка от примерно описание на предикати в XML. Лема – характеристики. Аргументна структура.

```

<word lemma="вярвам" pos="V" aspect="нечв" tr="intr"
refl="се(варва ми се - датива рефлексива тантум; варва се - безлично); си (варва си - рефлексива тантум си)">
<synt>
<ArgStr>
<def type="state">убеден съм, уверен съм, имам вара; вземам за истина</def>
<frame>
<NP function="s" SemRole="exp" expl="n"/>
<CP function="do" SemRole="theme" expl="n"/>
<example><a id="exp">A3</a>
<p>варвам</p>
<a id="theme">че е така</a></example>
</frame>
<frame>
<NP function="s" SemRole="exp" expl="n"/>
<PP function="io" SemRole="theme" expl="n"/>
<example><a id="exp">A3</a>
<p>варвам</p>
<a id="theme">на думите му</a>
</example>
</frame>
<frame>
<CP function="s" SemRole="exp" expl="y"/>
<example><a id="exp">Този, който иска</a>
<p>варва</p></example>
</frame>
</ArgStr>
<ArgStr>
<def type="state">имам доверие</def>
<frame>
<NP function="s" SemRole="exp" expl="n"/>
<PP function="io" expl="n" SemRole="theme"/>
<example><a id="exp">A3</a>
<a id="theme">му</a>
<p>варвам</p></example>
</frame>
</ArgStr>
</synt>
</word>

```

а. Глаголна парадигма. Морфологични характеристики.

```

<entry lemma="вървам">
  <pf t="p" p="1" n="sg">вървам</pf>
  <pf t="p" p="2" n="sg">върваш</pf>
  <pf t="p" p="3" n="sg">върва</pf>
  <pf t="p" p="1" n="pl">върваме</pf>
  <pf t="p" p="2" n="pl">вървате</pf>
  <pf t="p" p="3" n="pl">върват</pf>
  <pf t="a" p="1" n="sg">вървах</pf>
  <pf t="a" p="2" n="sg">върва</pf>
  <pf t="a" p="3" n="sg">върва</pf>
  <pf t="a" p="1" n="pl">вървахме</pf>
  <pf t="a" p="2" n="pl">вървахте</pf>
  <pf t="a" p="3" n="pl">върваха</pf>
  <pf t="i" p="1" n="sg">вървах</pf>
  <pf t="i" p="2" n="sg">върваше</pf>
  <pf t="i" p="3" n="sg">върваше</pf>
  <pf t="i" p="1" n="pl">вървахме</pf>
  <pf t="i" p="2" n="pl">вървахте</pf>
  <pf t="i" p="3" n="pl">върваха</pf>
  <if type="cd" n="sg" g="m" d="n">върваш</if>
  <if type="cd" n="sg" g="m" d="y">вървашия</if>
  <if type="cd" n="sg" g="m" d="yy">вървашият</if>
  <if type="cd" n="sg" g="f" d="n">върваша</if>
  <if type="cd" n="sg" g="f" d="y">вървашата</if>
  <if type="cd" n="sg" g="n" d="n">вървашо</if>
  <if type="cd" n="sg" g="n" d="y">вървашото</if>
  <if type="cd" n="pl" d="n" g="all">вървашис</if>
  <if type="cd" n="pl" d="y" g="all">вървашите</if>
  <if type="mcd" n="sg" g="m" d="n">вървал</if>
  <if type="mcd" n="sg" g="m" d="y">вървалия</if>
  <if type="mcd" n="sg" g="m" d="yy">вървалият</if>
  <if type="mcd" n="sg" g="f" d="n">вървала</if>
  <if type="mcd" n="sg" g="f" d="y">вървалата</if>
  <if type="mcd" n="sg" g="n" d="n">вървало</if>
  <if type="mcd" n="sg" g="n" d="y">вървалото</if>
  <if type="mcd" n="pl" d="n" g="all">вървалис</if>
  <if type="mcd" n="pl" d="y" g="all">вървалите</if>
  <if type="mnsd" n="sg" g="m" d="n">вървал</if>
  <if type="mnsd" n="sg" g="f" d="n">вървала</if>
  <if type="mnsd" n="sg" g="n" d="n">вървало</if>
  <if type="mnsd" n="sg" g="pl" d="n" g="all">вървали</if>
  <get>вървайки</get>
  <noun>върване</noun>
</entry>

```

23.5 Използване на корпуси. Търсене в българския Браун корпус чрез регулярни изрази

Както бе споменато по-горе лексикографската практика е да се събират автентични примери от реални текстове при създаването на речници и в днешно време се засилва и употребата на електронни корпуси за извлечение на нужна информация. Търсенето на всяка информација предоставяна от корпусите не е само практика на лексикографите, а и на много други специалисти в различни сфери и за различни нужди.

За българския език ще разгледаме българския Браун корпус.

На уеб страницата на Секцията по компютърна лингвистика към Института за български език „Професор Любомир Андрейчин“ в Българската академия на науките - http://dcl.bas.bg/Corpus/home_bg.html, от където е и достъпът до Браун корпуса, откриваме следното кратко описание:

„Българският Браун корпус е създаден съобразно методологията, разработена в университета Браун (Brown university, Providence, Rhode Island, USA) и приложена в създаването на известния Brown Corpus. Българският Браун корпус включва 500 текста, разпределени в 15 категории от 2 типа - художествени и информативни. Дължината на текстовете е приблизително фиксирана на 2 000 думи. Броят на думите варира с оглед на запазване на границите на началното и крайно изречение на всяка извадка. Големината на корпуса е 1 001 286 думи. Корпусните единици са части от текстове, създадени или публикувани като първо издание в периода 1990-2005, основната част - след 2000 година.“

Корпусът е документиран и нормализиран, направени са проверки за погрешна замяна на кирилски с латински букви и за правописни и пунктуационни грешки. Първата версия на корпуса е създадена през 2001-2002 година. При съставянето, поради невъзможност да се покрият всички категории, са пренебрегнати някои принципи на Принстънския Браун корпус (оригиналност и съвременност на текстовете и др.). Опитът от създаването на първата версия, както и значителното нарастване на електронните публикации в периода 2002-2005 г. дават възможност за съставянето на втората версия на корпуса.”

На страницата се предоставя среда за ползване на корпуса, където потребителите могат да правят своите заявки (търсене) по зададени съответни критерии. Достъпността е пригодена както за специалисти така и за други потребители с любознателни или други мотиви. Търсено то на думи, словосъчетания и др. може да се осъществява както чрез обикновено изписване на думата/-ите в полето „Търси”, така и чрез регулярен израз, които спомагат за едно по детайлно търсене и една лингвистично задълбочена заявка.

The screenshot shows the homepage of the Brown Corpus for Bulgarian Language. At the top, there is a navigation bar with links for 'Home', 'About', 'Search', 'Help', and 'Contact'. Below the navigation is a search form with fields for 'Text' and 'Type' (set to 'По шаблон'). To the left is a sidebar with a logo for 'DEPARTMENT OF COMPUTATIONAL LINGUISTICS' and a tree, and a menu with links: 'Заглавна' (Main), 'Съдържание' (Content), 'Класификация' (Classification), 'Описание' (Description), 'Употреба' (Usage), 'Авторски права' (Copyright), 'Публикации' (Publications), 'Връзки' (Links), 'Търсене' (Search), and 'EN'. The main content area has a section titled 'Заглавна' (Main) with text about the corpus's methodology and history. It also contains a section about the search function and its features. At the bottom of the page, there is a logo for 'National Science Fund'.

Например ако търсим изречения, в които фигурира името на Иван Вазов, за пример, бихме го изписали буквально в полето „Търси”, но ако искаме да направим списък от собствени имена и фамилии на лица ограждени с даден контекст, то заявката ни е по-абстрактна и би била зададена с регулярен израз от типа:

[А-Я][а-я]+ [А-Я][а-я]+

За имена с потенциални български фамилии:

[А-Я][а-я]+ [А-Я][а-я]+ов(а)?

Примерен регулярен израз за търсене на примери с парадигмата на прилагателно от типа:
върхове?н[иао]?[ят]?[гаоे]?
и др.

23.6 Заключение

В съвременния компютъризиран свят тенденцията е информацията от предигиталната епоха да бъде дигитализирана или по друг начин прехвърлена в дигиталния свят. В днешно време съвременните текстове биват дигитално създавани, търсено на информация е дигитално насочена, много сфери от живота са дигитално ориентирани,..., което е предпоставка и за дигитално създаване и „пребиваване” на ресурси. Компютърните (електронните) речници са съвременния поглед към лексикографията, а и не само. Все повече и повече лингвистиката се насочва към компютърната обработка на естествените езици, чийто цели са многолики. Лингвистиката бива вплетена в много сфери на съвременни научни интереси и стремежи, а речниците са винаги били част от нея. Работи се усърдно в сферата на машинния превод, в която тенденцията е да бъдат все повече и по-успешно комбинирани двата основни подхода, а именно този основан на лингвистични правила и този – на статистически методи, при които се използват анотирани корпуси, паралелни корпуси и други способи. В днешния свят, с навлизането на съвременната „*lingua franca*”, съществува заплахата за постепенно „потъване” на локалните диалекти и майчини езици. Много учени виждат в процеса на усъвършенстване на машинния превод едно спасение на лингвистичното богатство на народите, което ще позволи глобалното интернет общество да комуникира помежду си без да използва унифициран или общоприет комуникативен език. В този ред на мисли можем да заключим, че речникът е като един сандък, който пази словното богатство на един народ и като важен елемент с историческа стойност той успешно се внедрява в съвременните дигитални тенденции.

23.7 Използвана литература:

- .1 Български език – Списание на Института за български език „Професор Любомир Андрейчин“ при Българската академия на науките (2011 г.), кн. 3. → използван ресурс за точки 2 и 3 от настоящата работа
- .2 http://dcl.bas.bg/Corpus/home_bg.html - Браун корпус за български език

24 Речник: превод на термините

concept/term	Превод
best-merge persistent	запазва най-доброто сливане
Blocked sort-based indexing	Блоково индексиране основано на сортиране
BSBI	БИБС
buffer	буфер
bus	шина
caching	кеширане
Centroid Clustering	Центроидно клъстериране
centroid similarity	центроидно сходство
Clique	клика
clustering	клъстериране
coherent clusters	съгласувани клъстери
combination similarity	комбинирано подобие
complete-link	метод на пълното свързване
cosine similarity	косинусова мярка за подобие
Crawling	обхождане (на мрежата, на уеб сайт)
decision boundary	граница
dendrogram	дендрограма
Distributed	Разпределеност (на задачите)
divisive	разделящ
DNS resolution	превеждане на имената (превод от domain name към IP адрес)
document weighting function	функции за отегляване на документи
Domain Name Service	услуга за превеждане на домейн имена към IP адреси
Extensible	Разширяем
sorting algorithm	сортиращ алгоритъм
Flat clustering	Плоско клъстериране
Freshness	Актуалност
group-average	средно подобие

Group-average clustering/GAAC	agglomerative	Агломеративно клъстериране чрез осреднени подобия/АКОП
HAC/Hierarchical Clustering	Aggloemrative	ЙАК/Йерархично Агломеративно Клъстериране
header		хедър
Hierarchical clustering		Йерархично клъстериране
Hyperlink		връзка, хипервръзка
hyperplane		хиперравнина
Hypertext		хипертекст
inverted index		обърнат индекс
labeling clusters		етициране на клъстери
linear classifier		линеен класификатор
machine learning		машинно самообучение
margin		граница
merge		сливане
merge line		линия на сливане
Multithreaded		многонитков
outliers		крайности
Performance and efficiency		Бързодействие и ефективност
Politeness		съобразителност/учтивост
posting		срещане
postings list		списък със срещания
ranking		ранкиране
ranking function		ранкираща функция
Robots exclusion protocol		Протокол за изключване на обхождащи роботи
Robustness		устойчивост
Scalable		Машабируем
seek time		време за достъп
similarity		подобие
single-link		метод на единичното свързване
Single-pass in-memory indexing		Индексиране в паметта с едно обхождане
slack variable		слаба променлива

SPIMI	ИПЕО
structural SVM	структурна SVM
supervised machine learning	машинно самообучение с учител (с направление)
support vectors	поддържащи вектори
term	термин, терм
term	терм
text features	особености на текста
text summarization research	проучвания за обобщения на текстове (:D)
Thread	нишка
token	дума
training examples/data	множество обучаващи примери
transfer time per byte	време за трансфер на байт
URL	адрес
URL frontier	граница от адреси, гранични адреси
Web Crawler	обхождащ уеб робот, паяк, обхождащ робот