Temas 1-2-7-5 del libro traducido

Recuperación da Información (Universidade da Coruña)

1

Motores de busca e recuperación de información

> "Señor. Helpmann, estou moi disposto a recuperar a información ".
>
> SamLowry, *Brasil*

1.1 Que é a recuperación de información?

Este libro está deseñado para axudar á xente a comprender os motores de busca, avalialos e comparalos e modificalos para aplicacións específicas. Buscar información na web é, para a maioría da xente, unha actividade diaria. A busca e a comunicación son de lonxe os usos máis populares do ordenador. Non en balde, moita xente en empresas e universidades está a tratar de mellorar a busca ofrecendo xeitos máis sinxelos e rápidos de atopar a información correcta. Estas persoas, xa se chaman informáticas, enxeñeiros de software, científicos da información, optimizadores de motores de busca ou algo máis, están a traballar no campo da *Recuperación de información.*[1] Entón, antes de iniciar unha viaxe detallada a través dos internos dos motores de busca, levaremos algunhas páxinas para proporcionar un contexto para o resto do libro.

GerardSalton, pioneiro na recuperación de información e unha das figuras máis importantes desde os anos 60 ata os 90, propuxo a seguinte definición no seu clásico libro de texto de 1968 (Salton, 1968):

> A recuperación de información é un campo relacionado coa estrutura, análise, organización, almacenamento, busca e recuperación de información.

A pesar dos enormes avances na comprensión e tecnoloxía da busca nos últimos 40 anos, esta definición segue sendo axeitada e precisa. O termo “informa-

[1] A recuperación de información adoita abreviarse como IR. Neste libro, usamos principalmente o termo completo. Isto non ten nada que ver co feito de que moita xente pensa que o IR significa "infravermello" ou outra cousa.

ción ”é moi xeral e a recuperación de información inclúe traballos sobre unha ampla gama de tipos de información e unha variedade de aplicacións relacionadas coa busca.

O foco principal do campo desde a década de 1950 foi o texto e o texto *documentos.* Páxinas web, correo electrónico, artigos académicos, libros e noticias son só algúns dos moitos exemplos de documentos. Todos estes documentos teñen certa estrutura, como o título, o autor, a data e a información abstracta asociada ao contido dos traballos que aparecen en revistas científicas. Os elementos desta estrutura chámanse atributos ou campos cando se refiren a rexistros de bases de datos. A importante distinción entre un documento e un rexistro de base de datos típico, como un rexistro de conta bancaria ou unha reserva de voo, é que a maior parte da información do documento está en forma de texto, que está relativamente desestruturado.

Para ilustrar esta diferenza, considere a información contida en dous atributos típicos dun rexistro de conta, o número de conta e o saldo actual. Ambos están moi ben definidos, tanto no seu formato (por exemplo, un número enteiro de seis díxitos para un número de conta e un número real con dúas cifras decimais para o saldo) como no seu significado. É moi doado comparar os valores destes atributos e, en consecuencia, é sinxelo implementar algoritmos para identificar os rexistros que satisfagan consultas como "Buscar número de conta 321456" ou "Buscar contas con saldos superiores a $ 50.000,00".

Agora considere unha noticia sobre a fusión de dous bancos. A historia terá algúns atributos, como o título e a fonte da historia, pero o contido principal é a propia historia. Nun sistema de base de datos, esta información crítica normalmente gardaríase como un único atributo grande sen estrutura interna. A maioría das consultas enviadas a un buscador web como Google [2] que se relacionan con esta historia terán a forma "fusión bancaria" ou "adquisición bancaria". Para facer esta busca, debemos deseñar algoritmos que poidan comparar o texto das consultas co texto da historia e decidir se a historia contén a información que se busca. Definir o significado dunha palabra, unha frase, un parágrafo ou toda unha noticia é moito máis difícil que definir un número de conta e, en consecuencia, non é doado comparar o texto. Comprender e modelar como as persoas comparan textos e deseñar algoritmos informáticos para realizar con exactitude esta comparación está no núcleo da recuperación de información.

Cada vez máis, as aplicacións de recuperación de información implican documentos multimedia con estrutura, contido de texto significativo e outros soportes. Os medios de información populares inclúen imaxes, vídeo e audio, incluíndo música e voz. En

[2] http://www.google.com

algunhas aplicacións, como en soporte legal, tamén son importantes as imaxes de documentos escaneados. Estes soportes teñen contido que, como o texto, é difícil de describir e comparar. A tecnoloxía actual para buscar documentos que non son de texto depende das descricións de texto do seu contido e non do propio contido, pero estase a avanzar nas técnicas de comparación directa de imaxes, por exemplo.

Ademais dunha variedade de soportes, a recuperación de información implica unha serie de tarefas e aplicacións. O escenario de busca habitual consiste en que alguén escriba unha consulta nun motor de busca e reciba respostas na forma dunha lista de documentos por orde clasificada. Aínda que busca na rede mundial (*busca na web)* é de lonxe a aplicación máis común que implica a recuperación de información, a busca tamén é unha parte crucial das aplicacións en corporacións, goberno e moitos outros dominios. *Busca vertical* é unha forma especializada de busca na web onde o dominio da busca se restrinxe a un tema concreto. *Busca de empresas* implica atopar a información requirida na enorme variedade de ficheiros informáticos espallados por unha intranet corporativa. As páxinas web son sen dúbida unha parte dese almacén de información distribuída, pero a maior parte da información atoparémola en fontes como correo electrónico, informes, presentacións, follas de cálculo e datos estruturados en bases de datos corporativas. *Busca no escritorio* é a versión persoal da busca empresarial, onde as fontes de información son os ficheiros almacenados nun ordenador individual, incluídas as mensaxes de correo electrónico e as páxinas web que recentemente se exploraron. *Busca entre iguais* implica atopar redes de información de nodos ou computadoras sen ningún control centralizado. Este tipo de busca comezou como unha ferramenta para compartir arquivos para a música, pero pode usarse en calquera comunidade baseada en intereses compartidos, ou incluso na localidade compartida no caso de dispositivos móbiles. As técnicas de busca e recuperación de información relacionadas úsanse para publicidade, análise de intelixencia, descubrimento científico, asistencia sanitaria, atención ao cliente, propiedades inmobiliarias, etc. Calquera aplicación que implique un *colección*[3] de texto ou outra información non estruturada deberá organizar e buscar esa información.

Busca baseada nunha consulta do usuario (ás veces chamada *busca ad hoc* porque o abano de posibles consultas é enorme e non está preespecificado) non é a única tarefa baseada en texto que se estuda na recuperación de información. Outras tarefas inclúen *filtrado, clasificación,* e *resposta a preguntas.* O filtrado ou seguimento implica detectar historias de interese en función dos intereses dunha persoa e proporcionar unha alerta mediante correo electrónico ou algún outro mecanismo. A clasificación ou categorización emprega un conxunto definido de etiquetas ou clases

[3] O termo *base de datos* úsase a miúdo para referirse a unha colección de datos estruturados ou non estruturados. Para evitar confusións, usamos principalmente o termo *colección de documentos (* ou simplemente *colección)* para texto. Non obstante, os termos *base de datos web* e *base de datos de motores de busca* son así habitual que os usemos ocasionalmente neste libro.

(como as categorías listadas no directorio Yahoo! [4]) e asigna automaticamente esas etiquetas aos documentos. A resposta ás preguntas é similar á busca pero está dirixida a preguntas máis específicas, como "Cal é a altura do monte. Everest? ". O obxectivo da resposta ás preguntas é devolver unha resposta específica que se atopa no texto, en lugar dunha lista de documentos. A táboa 1.1 resume algúns destes aspectos ou dimensións do campo da recuperación de información.

Exemplos de Contido	Exemplos de Aplicacións	Exemplos de Tarefas
Texto	Busca na web	Busca ad hoc
Imaxes	Busca vertical	Filtrado
Vídeo	Busca de empresas	Clasificación
Documentos dixitalizados	Busca no escritorio	Resposta de preguntas
Audio	Busca entre iguais	
Música		

Táboa 1.1. Algunhas dimensións da recuperación de información

1.2 Os grandes problemas

Os investigadores de recuperación de información centráronse en algúns temas clave que seguen a ser tan importantes na era dos buscadores comerciais que traballaban con miles de millóns de páxinas web como cando se fixeron probas nos anos 60 en coleccións de documentos que contiñan aproximadamente 1,5 megabytes de texto. Unha destas cuestións é *relevancia.*

A relevancia é un concepto fundamental na recuperación de información. Falando libremente, un documento relevante contén a información que unha persoa buscaba cando presentou unha consulta ao motor de busca. Aínda que isto pareza sinxelo, hai moitos factores que entran na decisión dunha persoa de se un determinado documento é relevante. Estes factores deben terse en conta á hora de deseñar algoritmos para comparar textos e clasificar documentos. Simplemente compare o texto dunha consulta co texto dun documento e busca unha coincidencia exacta, como se podería facer nun sistema de base de datos ou empregando o grep utilidade en Unix, produce resultados moi pobres en termos de relevancia. Unha razón obvia disto é que a linguaxe pode usarse para

[4] http://dir.yahoo.com/

prema os mesmos conceptos de moitas maneiras diferentes, a miúdo con palabras moi diferentes. A isto chámaselle o *problema de desaxuste do vocabulario* na recuperación de información.

Tamén é importante distinguir entre *relevancia tópica* e *relevancia do usuario.* Un documento de texto é tópicamente relevante para unha consulta se se trata do mesmo tema. Por exemplo, unha noticia sobre un tornado en Kansas sería tópicamente relevante para a consulta "eventos meteorolóxicos severos". Non obstante, é posible que a persoa que fixo a pregunta (a miúdo chamada usuario) non considere a historia relevante, se xa viu esa historia ou se a historia ten cinco anos ou se a historia está en chinés dun chinés. axencia de noticias. A relevancia do usuario ten en conta estas características adicionais da historia.

Para tratar o tema da relevancia, os investigadores propoñen *modelos de recuperación* e proba como funcionan. Un modelo de recuperación é unha representación formal do proceso de correspondencia dunha consulta e un documento. É a base do *algoritmo de clasificación* que se usa nun motor de busca para producir a lista clasificada de documentos. Un bo modelo de recuperación atopará documentos que probablemente sexan considerados relevantes pola persoa que presentou a consulta. Algúns modelos de recuperación céntranse na relevancia tópica, pero un motor de busca despregado nun ambiente real debe empregar algoritmos de clasificación que incorporen relevancia de usuario.

Unha característica interesante dos modelos de recuperación empregados na recuperación de información é que normalmente modelan as propiedades estatísticas do texto en lugar da estrutura lingüística. Isto significa, por exemplo, que os algoritmos de clasificación normalmente están moito máis preocupados cos recontos de aparicións de palabras que se a palabra é un substantivo ou un adxectivo. Os modelos máis avanzados incorporan características lingüísticas, pero adoitan ter unha importancia secundaria. O uso da información de frecuencia de palabras para representar texto comezou con outro pioneiro na recuperación de información, HP Luhn, na década de 1950. Esta visión do texto non se popularizou noutros campos da informática, como o procesamento da linguaxe natural, ata os anos noventa.

Outro tema fundamental para a recuperación de información é *avaliación.* Dado que a calidade do ranking dun documento depende do ben que se axuste ás expectativas dunha persoa, foi preciso desenvolver desde cedo medidas de avaliación e procedementos experimentais para adquirir estes datos e usalos para comparar algoritmos de ranking. Cyril Cleverdon foi o líder no desenvolvemento de métodos de avaliación a principios dos anos sesenta, e dúas das medidas que utilizou, *precisión* e *recordo,* aínda son populares. A precisión é unha medida moi intuitiva e é a proporción de documentos recuperados que son relevantes. Recall é a proporción de documentos relevantes que se recuperan. Cando se usa a medida de retirada, suponse que se coñecen todos os documentos relevantes para unha consulta determinada. Tal suposición é claramente problemática nunha rede

ambiente de busca, pero con pequenos *coleccións de probas* de documentos, esta medida pode ser útil. Unha colección de probas [5] para experimentos de recuperación de información consiste nunha colección de documentos de texto, unha mostra de consultas típicas e unha lista de documentos relevantes para cada consulta (o *xuízos de relevancia).* As coleccións de proba máis coñecidas son as asociadas ao TREC [6] foro de avaliación.

A avaliación de modelos de recuperación e motores de busca é unha área moi activa, con gran parte do foco actual no uso de grandes volumes de datos de rexistro das interaccións dos usuarios, como *clickthrough* datos, que rexistran os documentos nos que se fixo clic durante unha sesión de busca. O clic e outros datos de rexistro están fortemente correlacionados coa relevancia polo que se poden usar para avaliar a busca, pero as compañías de motores de busca aínda usan xuízos de relevancia ademais dos datos de rexistro para garantir a validez dos seus resultados.

O terceiro tema fundamental para a recuperación de información é a énfase nos usuarios e os seus *necesidades de información.* Isto debería quedar claro tendo en conta que a avaliación da busca está centrada no usuario. É dicir, os usuarios dun motor de busca son os xuíces finais da calidade. Isto levou a numerosos estudos sobre como as persoas interactúan cos motores de busca e, en particular, ao desenvolvemento de técnicas para axudar ás persoas a expresar as súas necesidades de información. A necesidade de información é a causa subxacente da consulta que unha persoa envía a un motor de busca. A diferenza dunha solicitude a un sistema de base de datos, como o saldo dunha conta bancaria, as consultas de texto adoitan ser descricións pobres do que realmente quere o usuario. Unha soa consulta como "gatos" pode ser unha solicitude de información sobre onde mercar gatos ou unha descrición da música de Broadway. Non obstante, a pesar da súa falta de especificidade, as consultas cunha soa palabra son moi comúns
busca inweb. Técnicas como *suxestión de consulta, ampliación de consulta,* e *relevancia comentarios* use a interacción e o contexto para refinar a consulta inicial para producir listas mellor clasificadas.

Estas cuestións xurdirán ao longo deste libro e serán discutidas con moito máis detalle. Agora dispoñemos de antecedentes suficientes para comezar a falar do principal produto da investigación na recuperación de información, é dicir, os motores de busca.

1.3 Motores de busca

Un motor de busca é a aplicación práctica de técnicas de recuperación de información a coleccións de texto a gran escala. O motor de busca Aweb é o exemplo evidente, pero como

[5] Tamén se coñece como avaliación *corpus (* plural *corpus).*

[6] Conferencia de recuperación de texto: http: //trec.nist.gov/

mencionouse, os motores de busca pódense atopar en moitas aplicacións diferentes, como a busca por escritorio ou a empresa. Os motores de busca existen desde hai moitos anos. Por exemplo, MEDLINE, o sistema de busca de literatura médica en liña, comezou nos anos setenta. O termo "motor de busca" empregouse orixinalmente para referirse a hardware especializado para a busca de texto. Con todo, a partir de mediados dos anos oitenta pasouse a empregar gradualmente en preferencia ao "sistema de recuperación de información" como o nome do sistema de software que compara consultas con documentos e produce listas de resultados clasificadas de documentos. Un motor de busca ten moito máis que o algoritmo de clasificación, por suposto, e discutiremos a arquitectura xeral destes sistemas no seguinte capítulo.

Os motores de busca inclúen unha serie de configuracións que reflicten as aplicacións para as que están deseñados. Motores de busca web, como Google e Yahoo !, [7] debe ser capaz de capturar ou *rastrexar,* moitos terabytes de datos e despois proporcionan tempos de resposta inferiores a miles de millóns de consultas enviadas todos os días desde todo o mundo. Motores de busca empresariais, por exemplo, Autonomía [8] —Deber ser capaz de procesar a gran variedade de fontes de información nunha empresa e utilizar o coñecemento específico da empresa como parte da busca e tarefas relacionadas, como *minería de datos.* A minería de datos refírese ao descubrimento automático de datos de estrutura interesantes e inclúe técnicas como *agrupación.* Os motores de busca de escritorio, como a función de busca de Microsoft Vista ™, deben poder incorporar rapidamente novos documentos, páxinas web e correo electrónico a medida que a persoa os crea ou os mira, así como proporcionar unha interface intuitiva para buscar nesta mestura moi heteroxénea de información. Hai un exceso entre estas categorías con sistemas como Google, por exemplo, que está dispoñible en configuracións para a busca empresarial e de escritorio.

Código aberto os motores de busca son outra clase importante de sistemas que teñen obxectivos de deseño algo diferentes aos dos motores de busca comerciais. Hai varios destes sistemas e a páxina de Wikipedia para a recuperación de información [9] ofrece ligazóns a moitos deles. Tres sistemas de especial interese son Lucene, [10]

Lémur, [11] e o sistema fornecido con este libro, Galago. [12] Lucene é un popular motor de busca baseado en Java que se usou para unha ampla gama de aplicacións comerciais. As técnicas de recuperación de información que emprega son relativamente sinxelas.

---

7 http://www.yahoo.com
8 http://www.autonomy.com
9 http://en.wikipedia.org/wiki/Information_retrieval
10 http://lucene.apache.org
11 http://www.lemurproject.org
12 http://www.search-engines-book.com

Lemur é un xogo de ferramentas de código aberto que inclúe o motor de busca baseado en Indri C ++. O lemur foi utilizado principalmente por investigadores de recuperación de información para comparar técnicas avanzadas de busca. Galago é un motor de busca baseado en Java baseado nos proxectos Lemur e Indri. As tarefas deste libro fan un uso extensivo de Galago. Está deseñado para ser rápido, adaptable e fácil de comprender e incorpora técnicas de recuperación de información moi eficaces.

Os "grandes problemas" no deseño dos motores de busca inclúen os identificados para a recuperación de información: algoritmos de clasificación eficaces, avaliación e interacción cos usuarios. Non obstante, hai unha serie de características críticas adicionais dos motores de busca que resultan da súa implantación en contornos operativos a grande escala. O máis importante entre estas características é o *actuación* do motor de busca en termos de sures como *tempo de resposta, rendemento de consulta,* e *velocidade de indexación.* Tempo de resposta é o atraso entre enviar unha consulta e recibir a lista de resultados, o rendemento mide o número de consultas que se poden procesar nun tempo determinado e a velocidade de indexación é a velocidade coa que os documentos de texto poden transformarse en índices para a busca. An *índice* é unha estrutura de datos que mellora a velocidade de busca. O deseño de índices para os motores de busca é un dos principais temas deste libro.

Outra medida importante de rendemento é a rapidez con que se poden incorporar novos datos nos índices. As aplicacións de busca normalmente tratan información dinámica e en constante cambio. *Cobertura* mide cantas das informacións existentes nun, digamos, un contorno de información corporativa foron indexadas e almacenadas no motor de busca e *recencia* ou *frescura* mide a "idade" da información almacenada.

Os motores de busca pódense usar con pequenas coleccións, como algúns centos de correos electrónicos e documentos nun escritorio ou coleccións moi grandes, como toda a web. Pode haber só algúns usuarios dunha determinada aplicación ou moitos miles. *Escalabilidade* é claramente un problema importante para o deseño de motores de busca. Os deseños que funcionen para unha determinada aplicación deben seguir funcionando a medida que medra a cantidade de datos e o número de usuarios. Na sección 1.1, describimos como os motores de busca se usan en moitas aplicacións e para moitas tarefas. Para iso, teñen que selo *personalizable* ou *adaptable.* Isto significa que moitos aspectos diferentes do motor de busca, como o algoritmo de clasificación, a interface ou a estratexia de indexación, deben ser capaces de axustarse e adaptarse aos requirimentos da aplicación.

Tamén se producen problemas prácticos que afectan ao deseño dos motores de busca para aplicacións específicas. O mellor exemplo disto é *Correo non desexado* busca inweb. Xeralmente considérase que o correo non desexado é o correo lixo, pero máis xeralmente pódese definir como información enganosa, inapropiada ou non relevante nun documento deseñado para algúns

beneficio comercial. Hai moitos tipos de spam, pero un dos tipos que deben tratar os motores de busca son os spam que se colocan nun documento para que se recupere en resposta a consultas populares. A práctica do "spamdexing" pode degradar significativamente a calidade do ranking dun motor de busca e os deseñadores de motores de busca web teñen que desenvolver técnicas para identificar o spam e eliminar eses documentos. A figura 1.1 resume os principais problemas implicados no deseño de motores de busca.

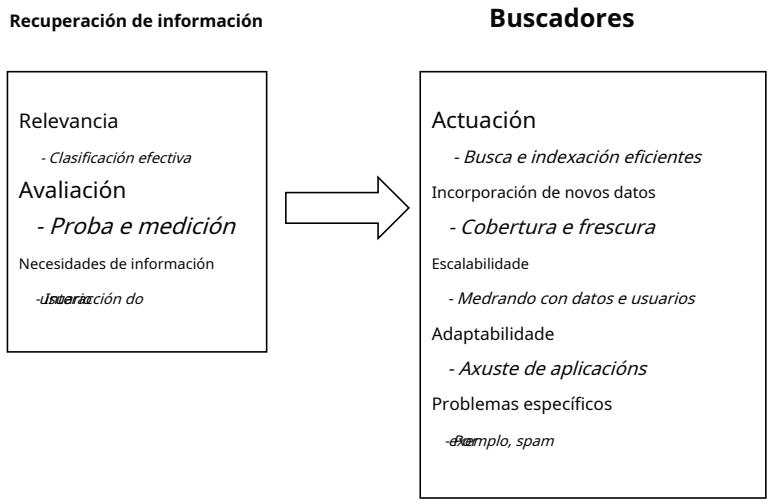

Fig 1.1. Deseño de motores de busca e problemas básicos de recuperación de información

Baseándonos nesta discusión sobre a relación entre a recuperación de información e os motores de busca, agora consideramos que funcións desempeñan os informáticos e outros no deseño e uso dos motores de busca.

1.4 Buscadores

A investigación de recuperación de información implica o desenvolvemento de modelos matemáticos de texto e linguaxe, experimentos a gran escala con coleccións de probas ou usuarios e moita escritura en papel académico. Por estas razóns, adoita facelo académicos ou persoas en laboratorios de investigación. Estas persoas están formadas principalmente en informática, aínda que tamén están representadas ciencias da información, matemáticas e, ocasionalmente, ciencias sociais e lingüística computacional. Entón, quen traballa

con motores de busca? En boa medida, é o mesmo tipo de xente pero con énfase máis práctica. A industria informática comezou a usar o termo *enxeñeiro de busca* para describir este tipo de persoas. Os enxeñeiros de busca son principalmente persoas formadas en informática, principalmente con antecedentes de sistemas ou bases de datos. Sorprendentemente, poucos deles teñen formación na recuperación de información, que é unha das principais motivacións deste libro.

Cal é o papel dun enxeñeiro de busca? Certamente as persoas que traballan nas principais empresas de busca na web deseñando e implementando novos motores de busca son enxeñeiros de busca, pero a maioría dos enxeñeiros de busca son as persoas que modifican, amplían, manteñen ou axustan os motores de busca existentes para unha ampla gama de aplicacións comerciais. As persoas que deseñan ou "optimizan" contido para os motores de busca tamén son enxeñeiros de busca, do mesmo xeito que as persoas que implementan técnicas para tratar o spam. Os motores de busca cos que traballan os enxeñeiros de busca abarcan todo o abano mencionado na última sección: usan principalmente motores de busca de código aberto e empresariais para o desenvolvemento de aplicacións, pero tamén sacan o máximo proveito dos buscadores de escritorio e web.

A importancia e omnipresencia da busca de aplicacións informáticas modernas fixo que a enxeñaría de busca se converta nunha profesión crucial na industria informática. Non obstante, hai moi poucos cursos que se imparten nos departamentos de ciencias da computación que permitan aos estudantes apreciar a variedade de cuestións relacionadas, especialmente desde a perspectiva da recuperación de información. Este libro preténdese dar aos potenciais enxeñeiros de busca a comprensión e as ferramentas que precisan.

Referencias e lecturas posteriores

En cada capítulo, ofrecemos algunhas indicacións sobre artigos e libros que ofrecen máis detalles sobre os temas tratados. Esta lectura adicional non debería ser necesaria para comprender o material que se presentou, senón que dará máis antecedentes, máis profundidade nalgúns casos e, para temas avanzados, describirá técnicas e resultados da investigación que non se tratan neste libro.

Na nosa opinión, as referencias clásicas sobre a recuperación de información son os libros de Salton (1968; 1983) e vanRijsbergen (1979). O libro de VanRijsbergen segue a ser popular, xa que está dispoñible na web. [13] Os tres libros ofrecen excelentes descricións da investigación realizada nos primeiros anos de recuperación de información, ata finais dos setenta. O primeiro libro de Salton foi particularmente importante en termos de definición

[13] http://www.dcs.gla.ac.uk/Keith/Preface.html

the field of information retrieval for computer science. Os libros máis recentes inclúen Baeza-Yates e Ribeiro-Neto (1999) e Manning et al. (2008).

Os traballos de investigación sobre todos os temas tratados neste libro pódense atopar na Conferencia do Grupo de interese especial sobre a recuperación de información (SIGIR) da Proceedings of the Association for Computing Machinery (ACM). Estes trámites están dispoñibles na web como parte da Biblioteca dixital de ACM. [14] Os bos traballos sobre recuperación e busca de información tamén aparecen na Conferencia Europea sobre Recuperación de Información (ECIR), na Conferencia sobre Información e Xestión do Coñecemento (CIKM) e na Conferencia de Busca e Minería de Datos (WSDM). TheWSDMconference é un spin-o ff da WorldWideWebConference (WWW), que incluíu algúns traballos importantes sobre a busca na web. Os actos dos talleres TREC están dispoñibles en liña e conteñen descricións útiles de novas técnicas de investigación de moitos grupos académicos e industriais diferentes. Pódese atopar unha visión xeral dos experimentos TREC en Voorhees e Harman (2005). Un número crecente de artigos relacionados coa busca comezan a aparecer en conferencias de bases de datos, como VLDB e SIGMOD. Tamén aparecen artigos ocasionais en congresos de tecnoloxía lingüística,

Exercicios

1.1. Pensa e escribe un pequeno número de consultas para un motor de busca web. Asegúrese de que as consultas varían na lonxitude (é dicir, non son todas unha palabra). Tente especificar exactamente a información que está a buscar nalgunhas das consultas. Executa estas consultas en dous buscadores comerciais na web e compara os 10 mellores resultados para cada consulta facendo xuízos de relevancia. Escribe un informe que responda polo menos ás seguintes preguntas: Cal é a precisión dos resultados? Cal é a superposición entre os resultados dos dous motores de busca? Un motor de busca é claramente mellor que o outro? Se é así, por cantas? Como funcionan as consultas curtas en comparación coas longas?

1.2. *Busca do sitio* é outra aplicación común dos motores de busca. Neste caso, a busca está restrinxida ás páxinas web dun determinado sitio web. Compare a busca no sitio coa busca web, a busca vertical e a busca empresarial.

[14] http://www.acm.org/dl

1.3. Use a web para atopar cantos exemplos poida ser de motores de busca de código aberto, sistemas de recuperación de información ou tecnoloxía relacionada. Fai unha breve descrición de cada motor de busca e resume as similitudes e diferenzas entre eles.

1.4. Enumere os servizos ou sitios de cinco webs que utilizan que aparecen para buscar, *non* incluíndo motores de busca web. Describe o papel da busca dese servizo. Describa tamén se a busca está baseada nunha base de datos ou `grep` estilo de coincidencia ou se a busca está a empregar algún tipo de clasificación.

2

Arquitectura dun buscador

"Aínda que a súa primeira pregunta pode ser a máis permanente, pode ou non entender que tamén é a máis irrelevante".

O Arquitecto, *Matrix Reloaded*

2.1 Que é unha arquitectura?

Neste capítulo, describimos a arquitectura de software básica dun motor de busca. Aínda que non existe un acordo universal sobre a definición, unha arquitectura de software xeralmente consiste en compoñentes de software, as interfaces proporcionadas por estes compoñentes e as relacións entre eles. Unha arquitectura úsase para describir un sistema cun nivel particular de abstracción. Un exemplo de arquitectura usada para proporcionar un estándar para integrar compoñentes de tecnoloxía de busca e linguaxe relacionados é UIMA (Unstructured InformationManagement Architecture). [1] UIMA define interfaces para compoñentes para simplificar a adición de novas tecnoloxías a sistemas que manexan texto e outros datos non estruturados.

A nosa arquitectura de motores de busca úsase para presentar descricións de alto nivel dos compoñentes importantes do sistema e as relacións entre eles. Non é unha descrición a nivel de código, aínda que algúns dos compoñentes corresponden a módulos de software no buscador Galago e outros sistemas. Usamos esta arquitectura neste capítulo e ao longo do libro para proporcionar contexto á discusión de técnicas específicas.

Unha arquitectura está deseñada para garantir que un sistema satisfaga os requisitos ou obxectivos da aplicación. Os dous obxectivos principais dun motor de busca son:

- Efectividade (calidade): queremos poder recuperar o conxunto de documentos máis relevante posible para unha consulta.
- Eficiencia (velocidade): queremos procesar as consultas dos usuarios o máis rápido posible.

[1] http://www.research.ibm.com/UIMA

Tamén podemos ter obxectivos máis específicos, pero normalmente estes entran nas categorías de eficacia ou eficacia (ou ambas). Por exemplo, a colección de documentos que queremos buscar pode estar cambiando; asegurarse de que o motor de busca reacciona inmediatamente aos cambios de documentos é un problema de eficacia e de eficacia.

A arquitectura dun motor de busca vén determinada por estes dous requisitos. Por ser un sistema eficiente, os motores de busca empregan estruturas de datos especializadas que están optimizadas para unha rápida recuperación. Debido a que queremos resultados de alta calidade, os motores de busca procesan coidadosamente o texto e almacenan estatísticas de texto que axudan a mellorar a relevancia dos resultados.

Moitos dos compoñentes que comentamos nas seguintes seccións empregáronse durante décadas e este deseño xeral demostrou ser un compromiso útil entre os obxectivos competentes de recuperación eficaz e eficiente. En capítulos posteriores, discutiremos estes compoñentes con máis detalle.

2.2 Elementos básicos

Os compoñentes do motor de busca admiten dúas funcións maiores, ás que chamamos *proceso de indexación* e o *proceso de consulta.* O proceso de indexación constrúe as estruturas que permiten a busca e o proceso de consulta utiliza esas estruturas e a consulta dunha persoa para producir unha lista clasificada de documentos. A figura 2.1 mostra os "bloques de construción" de alto nivel do proceso de indexación. Estes compoñentes principais son *adquisición de texto, texto transformación,* e *creación de índices.*

A tarefa do compoñente de adquisición de texto é identificar e poñer a disposición os documentos que se buscarán. Aínda que nalgúns casos isto implicará o uso sinxelo dunha colección existente, a adquisición de texto requirirá con máis frecuencia construír unha colección por *rastreando* ou escanear a web, unha intranet corporativa, un escritorio ou outras fontes de información. Ademais de pasar documentos ao seguinte compoñente do proceso de indexación, o compoñente de adquisición de texto crea un almacén de datos de documentos, que contén o texto e *metadatos* para todos os documentos. Os metadatos son información sobre un documento que non forma parte do contido do texto, como o tipo de documento (por exemplo, correo electrónico ou páxina web), a estrutura do documento e outras funcións, como a lonxitude do documento.

O compoñente de transformación de texto transforma os documentos en *termos índice* ou *características.* Os termos do índice, como o nome indica, son as partes dun documento que se gardan no índice e se utilizan na busca. O termo índice máis sinxelo é unha palabra, pero non todas as palabras poden usarse para buscar. Unha "característica" úsase máis a miúdo en

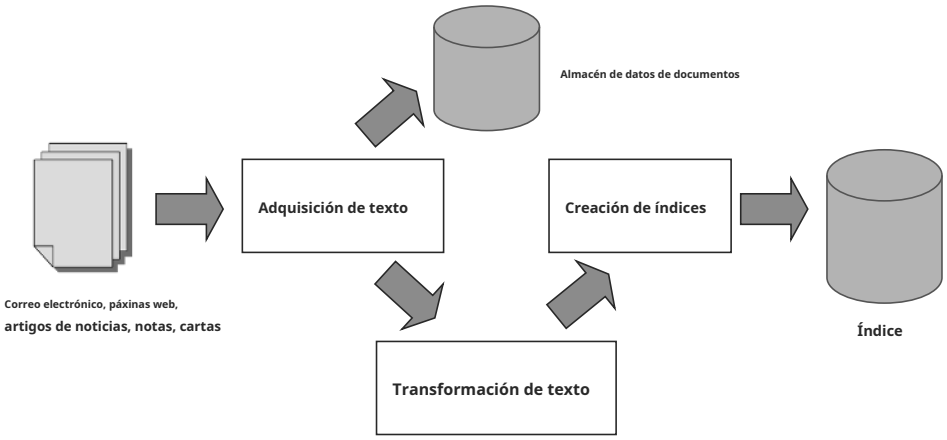

Fig. 2.1. O proceso de indexación

o campo da aprendizaxe automática para referirse a unha parte dun documento de texto que se usa para representar o seu contido, que tamén describe un termo índice. Exemplos doutros tipos de termos ou características do índice son frases, nomes de persoas, datas e ligazóns nunha páxina web. Ás veces, os termos do índice denomínanse simplemente "termos". O conxunto de todos os termos indexados para unha colección de documentos chámase *vocabulario índice.*

O compoñente de creación de índices toma a saída do compoñente de transformación de texto e crea os índices ou as estruturas de datos que permiten a busca rápida. Dado o gran número de documentos en moitas aplicacións de busca, a creación de índices debe ser eficiente, tanto en termos como en espazo. Os índices tamén deben ser capaces de ser eficientes *actualizado* cando se adquiren novos documentos. *Índices invertidos,* ou ás veces *ficheiros invertidos,* son de lonxe a forma máis común de índice utilizada polos motores de busca. Un índice invertido, moi sinxelamente, contén unha lista para cada termo índice dos documentos que conteñen ese termo índice. Invertese no sentido de ser o contrario a un ficheiro de documento que lista, para cada documento, os termos de índice que conteñen. Hai moitas variacións de índices invertidos, e a forma particular do índice empregado é un dos aspectos máis importantes dun motor de busca.

A figura 2.2 mostra os elementos básicos do proceso de consulta. A principal compo-

nentes son *interacción do usuario, clasificación,* e *avaliación.*

O compoñente de interacción do usuario proporciona a interface entre a persoa que realiza a busca e o motor de busca. Unha tarefa deste compoñente é aceptar a consulta do usuario e transformala en termos de índice. Outra tarefa é coller a lista clasificada de documentos do motor de busca e organizala na re-

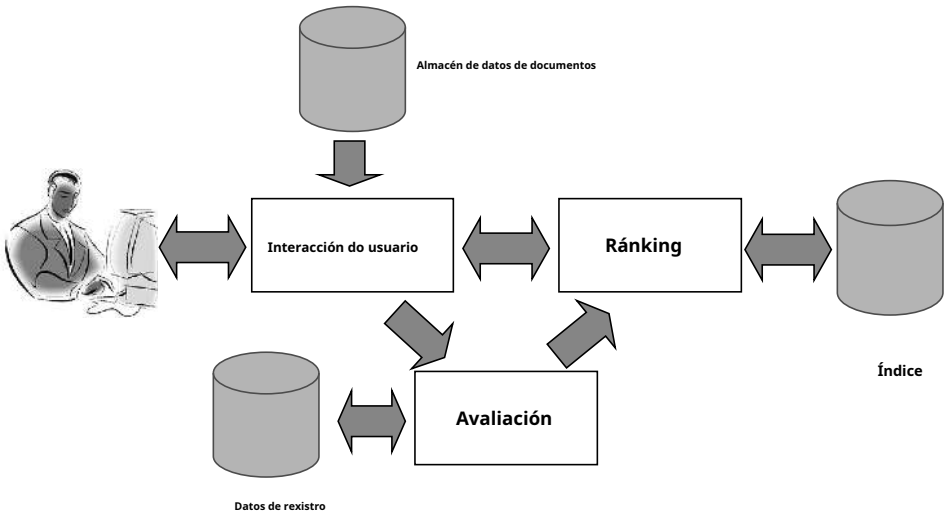

Fig. 2.2. O proceso de consulta

resultados mostrados ao usuario. Isto inclúe, por exemplo, xerar o *fragmentos* usado para resumir documentos. O almacén de datos do documento é unha das fontes de información empregadas para xerar os resultados. Finalmente, este compoñente tamén ofrece unha serie de técnicas para refinar a consulta de xeito que represente mellor a necesidade de información.

O compoñente da clasificación é o núcleo do motor de busca. Toma a consulta transformada do compoñente de interacción do usuario e xera unha lista clasificada de documentos usando puntuacións baseadas nun modelo de recuperación. A clasificación debe ser á vez eficiente, xa que é posible que moitas consultas teñan que ser procesadas en pouco tempo e eficaz, xa que a calidade da clasificación determina se o motor de busca cumpre o obxectivo de atopar información relevante. A eficacia da clasificación depende dos índices e a eficacia depende do modelo de recuperación.

A tarefa do compoñente de avaliación é medir e controlar a eficacia e a eficiencia. Unha parte importante diso é rexistrar e analizar o comportamento do usuario usando *datos de rexistro.* Os resultados da avaliación úsanse para sintonizar e mellorar o compoñente da clasificación. A maior parte do compoñente de avaliación non forma parte do motor de busca en liña, ademais de rexistrar os datos do usuario e do sistema. A avaliación é principalmente unha actividade normal, pero é unha parte crítica de calquera aplicación de busca.

2.3 Descompoñéndoo

Vemos agora con máis detalle os compoñentes de cada un dos bloques básicos. Non todos estes compoñentes formarán parte de todos os motores de busca, pero xuntamente abarcan o que consideramos que son as funcións máis importantes para unha ampla gama de aplicacións de busca.

2.3.1 Adquisición de texto

Rastreador

En moitas aplicacións, o *rastreador* compoñente ten a principal responsabilidade de identificar e adquirir documentos para o motor de busca. Existen varios tipos de rastrexadores diferentes, pero o máis común é o rastrexador web xeral. O rastrexador Aweb está deseñado para seguir as ligazóns en páxinas web para descubrir e descargar novas páxinas. Aínda que isto pareza enganosamente sinxelo, hai retos significativos no deseño dun rastrexador web que poida manexar eficientemente o enorme volume de novas páxinas na web, ao mesmo tempo que garante que as páxinas que poden cambiar desde a última vez que un rastrexador visitou un sitio mantéñense "frescos" para o motor de busca. Un rastrexador web pode restrinxirse a un único sitio, como unha universidade, como base para *busca de sitios. Enfocado,* ou *tópico,* os rastrexadores web usan técnicas de clasificación para restrinxir as páxinas que se visitan a aquelas que son susceptibles de tratar dun tema específico. Este tipo de rastrexo pode ser usado por un *vertical* ou *tópico* aplicación de busca, como un motor de busca que proporciona acceso á información médica nas páxinas web.

Para a busca de empresas, o rastrexador está adaptado para descubrir e actualizar todos os documentos e páxinas web relacionados co funcionamento dunha empresa. Unha empresa *rastrexador de documentos* segue ligazóns para descubrir páxinas externas e internas (é dicir, restrinxidas á intranet corporativa), pero tamén debe escanear directorios tanto corporativos como persoais para identificar correo electrónico, documentos de procesamento de textos, presentacións, rexistros de bases de datos e outra información da empresa. Os rastrexadores de documentos tamén se usan para a busca no escritorio, aínda que neste caso só hai que escanear os directorios persoais do usuario.

Feeds

Fontes de documentos son un mecanismo para acceder a un fluxo de documentos en tempo real. Por exemplo, unha fonte de noticias é un fluxo constante de noticias e actualizacións. Ao contraste cun rastrexador, que debe descubrir novos documentos, adquire un motor de busca

novos documentos dun feed simplemente controlándoo. *RSS*[2] é un estándar común utilizado para fontes web de contido como noticias, blogs ou vídeos. Un "lector" RSS úsase para subscribirse a fontes RSS, que se formatean usando *XML.*[3] XML é unha linguaxe para describir formatos de datos, similar ao HTML.[4] O lector supervisa eses feeds e proporciona contido novo cando chega. As fontes de radio e televisión tamén se usan nalgunhas aplicacións de busca, onde os "documentos" conteñen fluxos de audio e vídeo segmentados automáticamente, xunto con texto asociado de subtítulos ou recoñecemento de voz.

Conversión

Os documentos atopados por un rastrexador ou proporcionados por un feed raramente están en texto plano. Pola contra, veñen nunha variedade de formatos, como HTML, XML, Adobe PDF, Microsoft Word ™, Microsoft PowerPoint®, etc. A maioría dos motores de busca requiren que estes documentos se convertan nun texto consistente máis un formato de metadatos. Nesta conversión, as secuencias de control e os datos non contidos asociados a un formato concreto elimínanse ou rexístranse como metadatos. No caso de HTML e XML, gran parte deste proceso pode describirse como parte do compoñente de transformación de texto. Para outros formatos, o proceso de conversión é un paso básico que prepara o documento para o seu posterior procesamento. Os documentos PDF, por exemplo, deben converterse en texto. Hai varias utilidades dispoñibles que realizan esta conversión, con diferentes graos de precisión. Do mesmo xeito,

Outro problema de conversión común é o texto *codificado* nun documento. ASCII[5] é un esquema común de codificación de caracteres dun byte estándar usado para o texto. ASCII usa 7 ou 8 bits (ASCII ampliado) para representar 128 ou 256 caracteres posibles. Non obstante, algúns idiomas, como o chinés, teñen moitos máis caracteres que o inglés e usan outros esquemas de codificación. Unicode é un esquema de codificación estándar que usa 16 bits (normalmente) para representar a maioría das linguas do mundo. Calquera aplicación que trate de documentos en diferentes idiomas debe asegurarse de que se converten nun esquema de codificación consistente antes de procesalo.

2 RSS en realidade refírese a unha familia de estándares con nomes similares (e as mesmas iniciais), como a Really Simple Syndication ou Rich Site Summary.

3 Linguaxe de marcado extensible

4 Linguaxe de marcado de hipertexto

5 Código estándar americano para o intercambio de información

Almacén de datos de documentos

O almacén de datos de documentos é unha base de datos empregada para xestionar un gran número de documentos e os datos estruturados que están asociados a eles. Os contidos do documento normalmente almacénanse en forma comprimida para maior eficacia. Os datos estruturados constan de metadatos de documentos e outra información extraída dos documentos, como ligazóns e *texto áncora (* o texto asociado a unha ligazón). A *sistema de base de datos relacional* pódese usar para almacenar os documentos e metadatos. Non obstante, algunhas aplicacións utilizan un sistema de almacenamento máis sinxelo e eficaz para proporcionar tempos de recuperación moi rápidos para tendas de documentos moi grandes.

Aínda que os documentos orixinais están dispoñibles na web, na base de datos empresarial, o almacén de datos do documento é necesario para proporcionar un acceso rápido ao contido do documento para unha serie de compoñentes do motor de busca. A xeración de resumos dos documentos recuperados, por exemplo, tardaría demasiado se o motor de busca tivese que acceder aos documentos orixinais e reprocesalos.

2.3.2 Transformación de texto

Analizador

O compoñente de análise é o responsable do procesamento da secuencia de texto *fichas* no documento para recoñecer elementos estruturais como títulos, figuras, ligazóns e cabeceiras. *Tokenizante* o texto é un primeiro paso importante neste proceso. En moitos casos, as fichas son as mesmas que as palabras. Tanto o documento como o texto de consulta deben transformarse en fichas no mesmo xeito para que se poidan comparar facilmente. Hai unha serie de decisións que potencialmente supoñen unha recuperación que fan que a tokenización non sexa trivial. Por exemplo, unha definición sinxela para os tokens podería ser cadeas de caracteres alfanuméricos separados por espazos. Non obstante, isto non nos indica como tratar con caracteres especiais como maiúsculas, guións e apostrofos. ¿Debemos tratar a "mazá" igual que "Apple"? Hai "en liña" dúas palabras ou unha palabra? ¿Debería tratarse o apóstrofo de “O'Connor” como o de “propietario”? Nalgúns idiomas, a tokenización faise aínda máis interesante. O chinés, por exemplo, non ten un separador de palabras evidente como un espazo en inglés.

A estrutura do documento adoita especificarse mediante unha linguaxe de marcado como HTML ou XML. HTML é a linguaxe predeterminada empregada para especificar a estrutura das páxinas web. XML ten moita máis flexibilidade e úsase como formato de intercambio de datos para moitas aplicacións. O analizador de documentos utiliza o coñecemento do *sintaxe* da linguaxe de marcas para identificar a estrutura.

Utilízanse tanto HTML como XML *etiquetas* para definir documento *elementos.* Por exemplo, <h2> Buscar </h2> define "Buscar" como un título de segundo nivel en HTML. As etiquetas e outras secuencias de control deben tratarse adecuadamente cando se tokeniza. Outros tipos de documentos, como o correo electrónico e as presentacións, teñen unha sintaxe e métodos específicos para especificar a estrutura, pero gran parte disto pode ser eliminado ou simplificado polo compoñente de conversión.

Parando

O compoñente de parada ten a tarefa sinxela de eliminar palabras comúns do fluxo de fichas que se converten en termos de índice. As palabras máis comúns son normalmente *función* palabras que axudan a formar a estrutura das frases pero pouco contribúen por si soas á descrición dos temas tratados polo texto. Exemplos son "o", "de", "a" e "para". Debido a que son tan comúns, eliminalos pode reducir considerablemente o tamaño dos índices. Dependendo do modelo de recuperación que se use como base do ranking, a eliminación destas palabras normalmente non ten ningún impacto na eficacia do motor de busca e incluso pode melloralo un pouco. A pesar destas vantaxes potenciais, pode ser difícil decidir cantas palabras se inclúen no *lista de palabras de parada.* Algunhas listas de palabras clave empregadas na investigación conteñen centos de palabras. O problema co uso destas listas é que se fai imposible buscar con consultas como "estar ou non ser" ou "abaixo". Para evitalo, as aplicacións de busca poden empregar listas de palabras clave moi pequenas (quizais só conteñen "o") ao procesar o texto do documento, pero despois usan listas máis longas para o procesamento predeterminado do texto da consulta.

Descendente

Descender é outra transformación a nivel de palabra. A tarefa do compoñente orixinario (ou *stemmer)* é agrupar palabras que se derivan dun común *talo.* Agrupar "peixes", "peixes" e "pescar" é un exemplo. Ao substituír cada membro dun grupo por unha palabra designada (por exemplo, a máis curta, que neste caso é "peixe"), aumentamos a probabilidade de que coincidan as palabras empregadas en consultas e documentos. De feito, a derivación xeralmente produce pequenas melloras na eficacia da clasificación. Do mesmo xeito que parar, a derivación pódese facer de forma agresiva, conservativa ou en absoluto. A derivación agresiva pode causar problemas de busca. Pode que non sexa apropiado, por exemplo, recuperar documentos sobre diferentes variedades de peixes en resposta á consulta de "pesca". Algunhas aplicacións de busca usan un tallo máis conservador, como simplemente identificar formas plurais usando a letra "s", ou poden

non procure cando procese o texto do documento e céntrese en engadir variantes de palabras adecuadas á consulta.

Algúns idiomas, como o árabe, son máis complicados *morfoloxía* que o inglés e, en consecuencia, a derivación é máis importante. Un compoñente eficaz en árabe ten un enorme impacto na eficacia da busca. Pola contra, hai pouca variación de palabras noutras linguas, como o chinés, e para estas linguas a derivación non é efectiva.

Extracción e análise de enlaces

As ligazóns e o texto de referencia correspondente en páxinas web poden identificarse e extraerse facilmente durante a análise de documentos. A extracción significa que esta información se rexistra no almacén de datos do documento e pode indexarse por separado do contido xeral do texto. Os motores de busca web fan un uso extensivo desta información a través de *análise de ligazóns* algoritmos como PageRank (Brin & Page, 1998). A análise de ligazóns proporciona ao motor de busca unha clasificación da popularidade e, en certa medida, do *autoridade* dunha páxina (noutras palabras, o importante que é). *Texto de referencia,* que é o texto que se pode facer clic nunha ligazón web, pode usarse para mellorar o contido de texto dunha páxina á que apunta a ligazón. Estes dous factores poden mellorar significativamente a eficacia da busca na web para algúns tipos de consultas.

Extracción de información

A extracción de información úsase para identificar termos de índice que son máis complexos que as palabras simples. Isto pode ser tan sinxelo coma palabras en negra ou palabras en títulos, pero en xeral pode requirir un cálculo adicional significativo. A extracción de características sintácticas como frases nominais, por exemplo, require algunha forma de análise sintáctica ou *etiquetado de parte do discurso.* A investigación nesta área centrouse en técnicas para extraer características con contido semántico específico, como *entidade nomeada* recoñecedores, que poden identificar de forma fiable información como nomes de persoas, nomes de empresas, datas e localizacións.

Clasificador

O compoñente clasificador identifica os metadatos relacionados coa clase para documentos ou partes de documentos. Isto abrangue unha serie de funcións que a miúdo se describen por separado. As técnicas de clasificación asignan etiquetas de clase predefinidas aos documentos. Estas etiquetas normalmente representan categorías de actualidade como "deportes", "política" ou "negocios". Dúas

exemplos importantes doutros tipos de clasificación son a identificación de documentos como spam e a identificación de partes non contidas de documentos, como a publicidade. As técnicas de agrupación úsanse para agrupar documentos relacionados sen categorías predefinidas. Estes grupos de documentos pódense empregar de varias maneiras durante a clasificación ou a interacción do usuario.

Creación de índices

Estatísticas de documentos
Ponderación
Inversión
Distribución

2.3.3 Creación de índices

Estatísticas de documentos

A tarefa do compoñente de estatísticas de documentos é simplemente recompilar e rexistrar información estatística sobre palabras, funcións e documentos. Esta información é utilizada polo compoñente da clasificación para calcular puntuacións para documentos. Os tipos de datos que se requiren xeralmente son os recontos de aparicións de termos de índice (palabras e características máis complexas) en documentos individuais, as posicións nos documentos onde se produciron os termos de índice, as contas de aparicións en grupos de documentos (como todos documentos etiquetados como "deportes" ou toda a colección de documentos) e as extensións dos documentos en canto ao número de fichas. Os datos reais requiridos están determinados polo modelo de recuperación e o algoritmo de clasificación asociado. As estatísticas do documento almacénanse en *táboas de busca,* que son estruturas de datos deseñadas para a recuperación rápida.

Ponderación

Termo índice *pesos* reflicten a importancia relativa das palabras nos documentos e úsanse na computación das puntuacións para a clasificación. O modelo de recuperación determina a forma específica dun peso. O compoñente de ponderación calcula os pesos usando as estatísticas do documento e almacénalos en táboas de busca. Os pesos poderían calcularse como parte do proceso de consulta e algúns tipos de pesos requiren información sobre a consulta, pero facendo o maior cálculo posible durante a proceso de indexación, mellorarase a eficacia do proceso de consulta.

Coñécese como un dos tipos máis comúns empregados en modelos de recuperación de antigos *tf.idf* ponderación. Hai moitas variacións destes pesos, pero todos están baseados nunha combinación da frecuencia ou reconto de aparicións de termos de índice nun documento (o *frecuencia de prazo,* ou *tf)* e a frecuencia de aparición do termo índice ao longo de toda a colección de documentos (*frecuencia inversa do documento,* ou *idf).* O *idf* o peso chámase frecuencia inversa do documento porque dá pesos elevados aos termos que se producen en moi poucos documentos. Unha fórmula típica para *idf* é rexistro *N / n,* onde *N* é o

número total de documentos indexados polo motor de busca e *n* é o número de documentos que conteñen un termo concreto.

Inversión

O *inversión* o compoñente é o núcleo do proceso de indexación. A súa tarefa é cambiar o fluxo de información de termos de documentos procedentes do compoñente de transformación de texto en información de termos de documentos para a creación de índices invertidos. O reto é facelo con eficacia, non só para un gran número de documentos cando se crean inicialmente os índices invertidos, senón tamén cando os índices se actualizan con novos documentos de feeds ou rastrexos. O formato dos índices invertidos está deseñado para un procesamento rápido de consultas e depende en certa medida do algoritmo de clasificación empregado. Os índices tamén se comprimen para mellorar aínda máis a eficiencia.

Distribución do índice

O compoñente de distribución de índices distribúe índices en varios ordenadores e potencialmente en varios sitios dunha rede. A distribución é esencial para un rendemento eficiente cos motores de busca web. Distribuíndo os índices dun subconxunto dos documentos (*distribución de documentos),* tanto a indexación como o proceso de consulta pódense facer en *paralelo.* Distribución dos índices para un subconxunto de termos (*distribución a prazo)* tamén pode soportar o procesamento en paralelo de consultas. *Replicación* é unha forma de distribución onde as copias de índices ou partes de índices almacénanse en varios sitios para que o procesamento de consultas se faga máis eficiente reducindo os atrasos na comunicación. A busca de pares a pares implica unha forma de distribución menos organizada onde cada nodo dunha rede mantén os seus propios índices e colección de documentos.

Interacción do usuario

Entrada de consulta
Transformación da consulta
Saída de resultados

2.3.4 Interacción do usuario

Entrada de consulta

O compoñente de entrada de consulta proporciona unha interface e un analizador para un *lingua de consulta.* As linguas de consulta máis sinxelas, como as que se usan na maioría das interfaces de busca na web, só teñen un pequeno número de *operadores.* Un operador é un comando no idioma de consulta que se usa para indicar texto que debería ser tratado dun xeito especial. En xeral, os operadores axudan a aclarar o significado da consulta limitando como

o texto do documento pode coincidir co texto da consulta. Un exemplo de operador nunha linguaxe de consulta simple é o uso de comiñas para indicar que as palabras incluídas deben aparecer como unha frase no documento, en lugar de como palabras individuais sen relación. Non obstante, unha consulta web típica consiste nun pequeno número de *palabras clave* sen operadores. Unha palabra clave é simplemente unha palabra importante para especificar o tema dunha consulta. Debido a que os algoritmos de clasificación para a maioría dos motores de busca web están deseñados para consultas de palabras clave, as consultas máis longas que poden conter unha proporción máis baixa de palabras clave normalmente non funcionan ben. Por exemplo, a consulta "motores de busca" pode producir un mellor resultado cun motor de busca web que a consulta "cales son as técnicas de implementación típicas e as estruturas de datos utilizadas nos motores de busca". Un dos retos para o deseño de motores de busca é dar bos resultados para unha serie de consultas e mellores resultados para consultas máis específicas.

Hai idiomas de consulta máis complexos dispoñibles, xa sexa para persoas que desexan ter moito control sobre os resultados da busca ou para aplicacións que utilizan un motor de busca. Do mesmo xeito que a linguaxe de consulta SQL (Elmasri e Navathe, 2006) non está deseñada para o usuario típico dunha aplicación de base de datos (o *usuario final),* estes idiomas de consulta non están deseñados para os usuarios finais de aplicacións de busca. *Booleano* as linguas de consulta teñen unha longa historia na recuperación de información. Os operadores neste idioma inclúen Boolean E, OU, e NON, e algunha forma *proximidade* operador que especifica que as palabras deben producirse xuntas a unha distancia específica (normalmente en termos de reconto de palabras). Outras linguaxes de consulta inclúen estes e outros operadores nun marco probabilístico deseñado para permitir a especificación de funcións relacionadas tanto coa estrutura como co contido do documento.

Transformación da consulta

O compoñente de transformación de consulta inclúe unha serie de técnicas deseñadas para mellorar a consulta inicial, tanto antes como despois de producir unha clasificación de documentos. O procesamento máis sinxelo implica algunhas das mesmas técnicas de transformación de texto empregadas no texto do documento. A tokenización, detención e derivación deben facerse no texto da consulta para producir termos de índice que sexan comparables aos termos do documento.

Corrección ortográfica e *suxestión de consulta* son técnicas de transformación de consultas que producen resultados similares. En ámbolos dous casos, preséntaselle ao usuario alternativas á consulta inicial que son susceptibles de corrixir erros ortográficos ou ser descricións máis específicas das súas necesidades de información. Estas técnicas adoitan aproveitar o extenso *rexistros de consulta* recollidos para aplicacións web. *Ampliación de consulta* técnicas

tamén suxire ou engade termos adicionais á consulta, pero normalmente baseado nunha análise de aparición de termos nos documentos. Esta análise pode empregar diferentes fontes de información, como toda a colección de documentos, os documentos recuperados ou documentos no ordenador do usuario. *Comentarios de relevancia* é unha técnica que amplía as consultas baseadas en termos ocorridos en documentos que o usuario identifica como relevantes.

Saída de resultados

O compoñente de saída de resultados é o responsable de construír a visualización dos documentos clasificados procedentes do compoñente de clasificación. Isto pode incluír tarefas como xerar *fragmentos* para resumir os documentos recuperados, *destacando* palabras e pasaxes importantes en documentos, agrupando a saída para identificar grupos de documentos relacionados e atopando publicidade adecuada para engadir á exhibición de resultados. Nas aplicacións que inclúan documentos en varios idiomas, os resultados poden traducirse a un idioma común.

Ránking

Puntuación
Optimización
Distribución

2.3.5 Clasificación

Puntuación

O compoñente de puntuación, tamén chamado *procesamento de consultas,* calcula as puntuacións dos documentos usando o algoritmo de clasificación, que se basea nun modelo de recuperación. Os deseñadores dalgúns motores de busca indican explícitamente o modelo de recuperación que utilizan. Para outros motores de busca, só se discute o algoritmo de clasificación (se se revelan detalles), pero todos os algoritmos de clasificación están baseados implicitamente nun modelo de recuperación. As funcións e pesos empregados nun algoritmo de clasificación, que pode que se deriven *empiricamente (* por probas e avaliación), debe estar relacionado coa actualidade e a relevancia do usuario, ou o motor de busca non funcionaría.

Propuxéronse moitos modelos e métodos de recuperación de algoritmos de clasificación diferentes. A forma básica da puntuación do documento calculada por moitos destes modelos é

$$\sum_{eu} q_{eu.}\, d_{eu}$$

onde a suma é superior a todos os termos do vocabulario da colección, q_{eu} é o peso do termo de consulta do *eu* th trimestre e d_{eu} é o peso do termo do documento. O termo pesas depende do modelo de recuperación particular que se está a empregar, pero son xeralmente semellante a *tf.idf* pesos. No capítulo 7, comentamos os algoritmos de clasificación

baseado no *BM25* e *probabilidade de consulta* modelos de recuperación (así como outros) con máis detalle.

A puntuación do documento debe calcularse e compararse moi rapidamente para determinar a orde clasificada dos documentos que se dan ao compoñente de saída de resultados. Esta é a tarefa do compoñente de optimización do rendemento.

Optimización do rendemento

A optimización do rendemento implica o deseño de algoritmos de clasificación e os índices asociados para diminuír o tempo de resposta e aumentar o rendemento da consulta. Dada unha forma particular de puntuación de documentos, hai varias formas de calcular esas puntuacións e producir a saída do documento clasificado. Por exemplo, as puntuacións pódense calcular accedendo ao índice dun termo de consulta, calculando a contribución dese termo á puntuación dun documento, engadindo esta contribución a un acumulador de puntuacións e accedendo ao seguinte índice. A isto chámaselle *termo á vez* puntuación. Outra alternativa é acceder simultaneamente a todos os índices dos termos de consulta e calcular puntuacións movendo punteiros a través dos índices para atopar os termos presentes nun documento. Neste *documento á vez* puntuación, a puntuación final do documento calcúlase inmediatamente en vez de acumularse cada trimestre. En ambos os casos, son posibles outras optimizacións que diminúen significativamente o tempo necesario para computar os documentos de mellor clasificación. *Seguro* as optimizacións garanten que as puntuacións calculadas serán as mesmas que as puntuacións sen optimización.
Inseguro as optimizacións, que non teñen esta propiedade, nalgúns casos poden ser máis rápidas, polo que é importante avaliar detidamente o impacto da optimización.

Distribución

Dada algunha forma de distribución de índices, a clasificación tamén se pode distribuír. A *corredor de consultas* decide como asignar as consultas aos procesadores dunha rede e é o responsable de montar a lista final clasificada para a consulta. O funcionamento do corredor depende da forma de distribución do índice. *Caché* é outra forma de distribución onde os índices ou incluso listas de documentos clasificados das consultas anteriores quedan na memoria local. Se a consulta ou o termo índice é popular, hai unha probabilidade significativa de que esta información poida ser reutilizada cun aforro substancial de tempo.

2.3.6 Avaliación

Rexistro

Os rexistros das consultas dos usuarios e as súas interaccións co motor de busca son unha das fontes de información máis valiosas para axustar e mellorar a eficacia e a eficacia da busca. Os rexistros de consultas pódense empregar para a corrección ortográfica, suxestións de consulta, almacenamento en caché de consultas e outras tarefas, como axudar a facer coincidir a publicidade coas buscas. Os documentos dunha lista de resultados nos que se fai clic e se navega adoitan ser relevantes. Isto significa que os rexistros de clics do usuario en documentos (datos de clics) e información como o *tempo de permanencia (* tempo dedicado a mirar un documento) pódese usar para avaliar e adestrar algoritmos de clasificación.

Análise de clasificación

Tendo en conta datos de rexistro ou xuízos de relevancia explícitos para un gran número de pares (consulta, documento), a eficacia dun algoritmo de clasificación pódese medir e comparar con alternativas. Esta é unha parte fundamental para mellorar un motor de busca e seleccionar valores para os parámetros axeitados para a aplicación. Adoitan empregarse unha variedade de medidas de avaliación, que tamén deben seleccionarse para medir os resultados que teñen sentido para a aplicación. As medidas que enfatizan a calidade dos documentos mellor clasificados, en lugar de toda a lista, por exemplo, son axeitadas para moitos tipos de consultas web.

Análise de rendemento

O compoñente de análise de rendemento implica supervisar e mellorar o rendemento global do sistema, do mesmo xeito que o compoñente de análise de clasificación supervisa a eficacia. Empréganse unha variedade de medidas de rendemento, como o tempo de resposta e o rendemento, pero as medidas empregadas tamén dependen da aplicación. Por exemplo, unha aplicación de busca distribuída debe supervisar o uso e a eficiencia da rede ademais doutras medidas. Para a análise de clasificación, as coleccións de probas adoitan empregarse para proporcionar un ambiente experimental controlado. O equivalente para a análise de rendemento é *simulacións,* onde as redes reais, procesadores, dispositivos de almacenamento e datos son substituídos por modelos matemáticos que se poden axustar mediante parámetros.

2.4 Como funciona *De verdade* Traballo?

Agora xa coñeces os nomes e as funcións básicas dos compoñentes dun motor de busca, pero aínda non dixemos moito sobre como estes compoñentes realmente cumpren estas funcións. Diso trata o resto do libro. Cada capítulo describe, profundidade, como máis ou máis compoñentes traballan. Se aínda non entendes un compoñente despois de rematar o capítulo apropiado, podes estudar o código Galago, que é unha implementación das ideas presentadas ou as referencias descritas ao final de cada capítulo.

Referencias e lecturas posteriores

Referencias detalladas sobre as técnicas e modelos mencionados nas descricións dos compoñentes daranse nos capítulos correspondentes. Hai algunhas referencias xerais para arquitecturas de busca. Un libro de base de datos, como Elmasri e Navathe (2006), ofrece descricións da arquitectura do sistema de bases de datos e das linguaxes de consulta asociadas que son interesantes para comparar coa arquitectura do motor de busca aquí comentada. Hai algunhas similitudes no alto nivel, pero os sistemas de bases de datos céntranse en datos estruturados e coincidencias exactas en lugar de en texto e clasificación, polo que a maioría dos compoñentes son moi diferentes.

O clásico traballo de investigación sobre arquitectura de motores de busca web, que ofrece unha visión xeral dunha versión inicial de Google, é Brin e Page (1998). Outra visión xeral do sistema para un motor de busca de propósito xeral anterior (Inquery) atópase en Callan et al. (1992). Pódese atopar unha descrición completa da arquitectura e compoñentes de Lucene en Hatcher e Gospodnetic (2004).

Exercicios

2.1. Busque algúns exemplos dos compoñentes do motor de busca descritos neste capítulo no código Galago.

2.2. A *máis-coma-isto* a consulta prodúcese cando o usuario pode facer clic nun documento concreto da lista de resultados e indicarlle ao motor de busca que atope documentos similares a este. Describe que compoñentes de baixo nivel se utilizan para responder a este tipo de consulta e a secuencia na que se usan.

2.3. O filtrado de documentos é unha aplicación que almacena un gran número de consultas ou perfís de usuario e compara estes perfís con todos os documentos entrantes nun feed. Os documentos que son suficientemente similares ao perfil remítense a esa persoa por correo electrónico ou algún outro mecanismo. Describe a arquitectura dun motor de filtrado e como pode diferir dun motor de busca.

5

Clasificación con índices

"Debe ir máis rápido".

David Levinson, *Día da Independencia*

5.1 Visión xeral

Como se trata dun libro bastante técnico, se leu ata aquí, probablemente comprenderá algo *estruturas de datos* e como se usan nos programas. Se desexa almacenar unha lista de elementos, as listas ligadas e as matrices son boas opcións. Se desexa atopar rapidamente un elemento baseado nun atributo, unha táboa de hash é a mellor opción. As tarefas máis complicadas requiren estruturas máis complicadas, como árbores B ou colas prioritarias.

Por que son necesarias todas estas estruturas de datos? En rigor, non o son. A maioría das cousas que desexa facer cun ordenador pódense facer só con matrices. Non obstante, as matrices teñen inconvenientes: as matrices sen clasificar son lentas na busca e as matrices ordenadas son lentas na inserción. Pola contra, as táboas de hash e as árbores son rápidas tanto para a busca como para a inserción. Estas estruturas son máis complicadas que as matrices, pero a diferenza de velocidade é convincente.

A busca de texto é moi diferente das tarefas de computación tradicionais, polo que require un tipo de estrutura de datos, o *índice invertido.* O nome "índice invertido" é realmente un termo común para moitos tipos diferentes de estruturas que comparten a mesma filosofía xeral. Como verás en breve, o tipo específico de estrutura de datos empregada depende da función de clasificación. Non obstante, dado que as funcións de clasificación que clasifican ben os documentos teñen unha forma similar, os tipos máis útiles de índices invertidos atópanse en case todos os motores de busca.

Este capítulo trata sobre como as consultas dos motores de busca son realmente procesadas por un ordenador, polo que se podería chamar todo este capítulo *procesamento de consultas.* A última sección deste capítulo chámase así e os algoritmos de procesamento de consultas presentados baséanse nas estruturas de datos presentadas anteriormente no capítulo.

O procesamento de consultas é un problema especialmente importante na busca na web, xa que alcanzou unha escala que sería difícil de imaxinar hai só 10 anos. Persoas de todo o mundo escriben máis de medio billón de consultas todos os días, buscando índices que conteñen miles de millóns de páxinas web. Os índices invertidos están no núcleo de todos os motores de busca web modernos.

Existen fortes dependencias entre os compoñentes separados dun motor de busca. O algoritmo de procesamento de consultas depende do modelo de recuperación e di o contido do índice. Isto tamén funciona á inversa, xa que é improbable que elixamos un modelo de recuperación que non teña algoritmo de procesamento de consultas eficiente. Dado que non discutiremos detalladamente os modelos de recuperación ata o capítulo 7, comezamos este capítulo describindo un modelo abstracto de clasificación que motiva a nosa elección de índices. Despois diso, hai catro partes principais no capítulo. Na primeira parte, discutimos os diferentes tipos de índice invertido e que información sobre os documentos se captura en cada índice. A segunda parte ofrece unha visión xeral das técnicas de compresión, que son unha parte crítica da implementación eficiente de índices inversos para a recuperación de texto. A terceira parte do capítulo describe como se constrúen os índices, incluíndo unha discusión sobre o marco MapReduce que se pode empregar para coleccións de documentos moi grandes. A parte final do capítulo céntrase en como se usan os índices para xerar clasificacións de documentos en resposta a consultas.

5.2 Modelo abstracto de clasificación

Antes de comezar a ver como construír índices para un sistema de busca, comezaremos por considerar un modelo abstracto de clasificación. Todas as técnicas que consideraremos neste capítulo pódense ver como implementacións deste modelo.

A figura 5.1 mostra os compoñentes básicos do noso modelo. Na parte esquerda da figura hai un documento de mostra. Os documentos están escritos en linguaxes humanas naturais, que son difíciles de analizar directamente polos ordenadores. Así, como vimos no capítulo 4, o texto transfórmase en *termos índice* ou *características do documento.* Para os efectos deste capítulo, unha característica do documento é un atributo do documento que podemos expresar numéricamente. Na figura, amosamos dous tipos de características. Por riba, temos *tópico* características, que estiman o grao en que o documento trata dun asunto concreto. Na parte inferior da figura vemos dous posibles documentos *calidade* características. Unha característica é o número de páxinas web que ligan a este documento e outra é o número de días desde a última actualización desta páxina. Estas funcións non resolven se o documento é unha boa coincidencia tópica para unha consulta, pero si

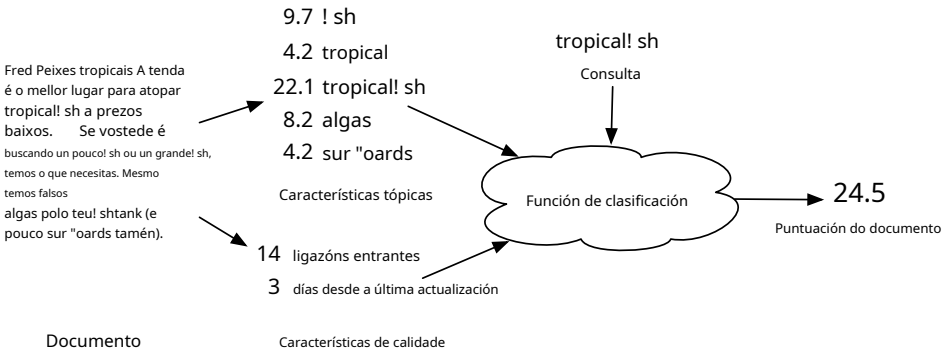

Fig. 5.1. Os compoñentes do modelo abstracto de clasificación: documentos, características, consultas, a función de recuperación e puntuacións de documentos

aborda a súa calidade: unha páxina sen ligazóns entrantes que non se editou durante anos é probablemente unha mala coincidencia para calquera consulta. Cada un destes valores de característica xérase usando un *función de función,* que é só unha expresión matemática que xera números a partir do texto do documento. No capítulo 4 comentamos algunhas das principais funcións tópicas e de calidade e no capítulo 7 aprenderás sobre as técnicas para crear boas funcións de funcións. Neste capítulo supoñemos que xa se crearon valores de características razoables.

Na parte dereita da figura, vemos unha nube que representa o *función de clasificación.* A función de clasificación toma datos das funcións do documento combinadas coa consulta e produce unha puntuación. Polo de agora, o contido desa nube non ten importancia, agás o feito de que as clasificacións máis razoables funcionan ignorando as características do documento e céntranse só no pequeno subconxunto relacionado coa consulta. Este feito fai do índice invertido unha estrutura de datos atractiva para a busca.

A saída final da función de clasificación é unha puntuación, que supoñemos que é un número real. Se un documento obtén unha puntuación alta, isto significa que o sistema pensa que o documento é un bo xogo para a consulta, mentres que un número máis baixo significa que o sistema pensa que o documento é un bo resultado para a consulta. Para crear unha lista clasificada de resultados, os documentos están ordenados por puntuación de xeito que os documentos con maior puntuación sexan os primeiros.

Supoña que é un motor de busca humano, que intenta clasificar os documentos nunha orde adecuada en resposta a unha consulta do usuario. Quizais colocarías os documentos en pilas, como "bo", "non tan bo" e "malo". O ordenador está facendo esencialmente o mesmo coa puntuación. Non obstante, tamén pode romper os lazos mirando con atención cada documento para decidir cal é o máis relevante. Desafortunadamente,

atopar un significado profundo nos documentos é difícil para os computadores, polo que os motores de busca céntranse en identificar boas funcións e puntuar en función desas características.

Un modelo de clasificación máis concreto

Máis adiante no capítulo veremos as técnicas de avaliación de consultas que supoñen algo máis forte sobre o que acontece na función de clasificación. En concreto, asumimos que a función de clasificación R adopta a seguinte forma:

$$R(Q, D) = \sum_{eu} g_{eu}(Q) f_{eu}(D)$$

Aquí, f_{eu} é algunha función que extrae un número do documento texto. g_{eu} é unha función de función similar que extrae un valor da consulta. Estas dúas funcións forman un par de funcións de características. Cada par de funcións multiplícase xuntos e engádense os resultados de todos os pares para crear unha puntuación final do documento.

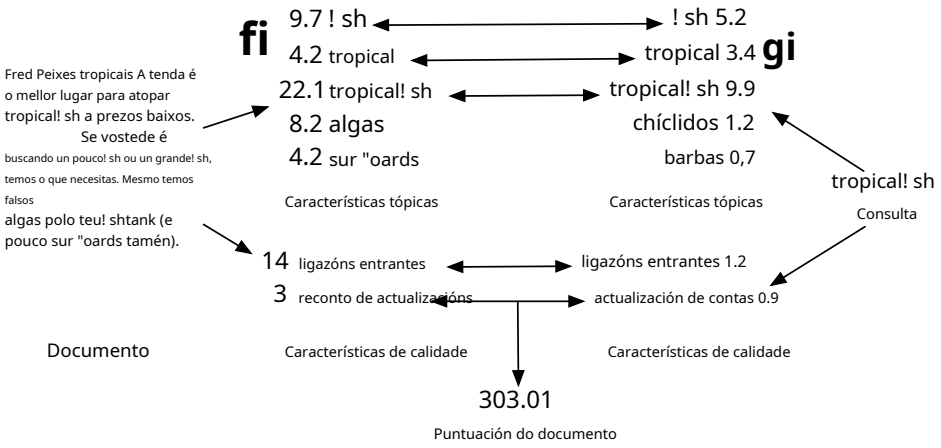

Fig. 5.2. Un modelo de clasificación máis concreto. Observe como a consulta e o documento teñen funcións de características neste modelo.

A figura 5.2 mostra un exemplo deste modelo. Do mesmo xeito que no modelo abstracto de clasificación, extráense varias características do documento. Esta imaxe mostra só algunhas características, pero en realidade haberá moitas máis. Estes corresponden ao $f_{eu}(D)$ funcións na ecuación que se acaba de amosar. Poderiamos nomealos facilmente $f_{\text{tropical}}(D)$ ou $f_{\text{peixe}}(D)$; estes valores serán maiores para os documentos que conteñan as palabras "tropical" ou "peixe" con máis frecuencia ou con máis importancia.

O documento ten algunhas características que non son de actualidade. Para este documento de exemplo, vemos que o motor de busca nota que este documento foi actualizado tres veces e que ten 14 ligazóns entrantes. Aínda que estas funcións non nos din nada sobre se este documento coincidiría co asunto dunha consulta, si que nos dan algúns consellos sobre a calidade do documento. Sabemos que non só foi publicado na web e logo abandonado, xa que se actualiza ocasionalmente. Tamén sabemos que hai outras 14 páxinas con ligazóns que o apuntan, o que pode significar que ten algunha información útil sobre el.

Teña en conta que tamén hai funcións de funcións que actúan na consulta. A característica función $g_{\text{tropical}}(Q)$ avalía un gran valor porque "tropical" está na consulta. Non obstante, $g_{\text{barbas}}(Q)$ tamén ten un pequeno valor diferente de cero porque está relacionado con outros termos da consulta. Estes valores das funcións de función de consulta multiplícanse polas funcións do recurso do documento, entón resumidas para crear unha puntuación do documento.

A consulta tamén ten algúns valores de funcións que non son tópicos, como a función de reconto de actualizacións. Por suposto, isto non significa que a consulta se actualizou. O valor desta función indica a importancia que teñen as actualizacións de documentos para esta consulta. Por exemplo, se a consulta fose "o tempo de hoxe en Londres", preferiríamos os documentos que se actualizan con frecuencia, xa que é improbable que un documento que non se actualice polo menos a diario diga algo interesante sobre o tempo de hoxe. Esta consulta debería ter un valor elevado para a función de reconto de actualizacións. Pola contra, un documento que nunca cambiou podería ser moi relevante para a consulta "texto completo de moby dick". Esta consulta pode ter un valor de función baixo para o reconto de actualizacións.

Se un sistema de recuperación tivese que realizar unha suma de millóns de funcións por cada documento, os sistemas de busca de texto non serían prácticos. Na práctica, a consulta tenturas ($g_{eu}(Q)$) son maioritariamente cero. Isto significa que a suma de cada documento é só superior a cero $g_{eu}(Q)$ valores.

5.3 Índices invertidos

Todos os índices de motores de busca modernos están baseados en *índices invertidos.* No pasado empregáronse outras estruturas de índice, sobre todo *ficheiros de sinatura,* [1] pero os índices invertidos considéranse a estrutura de índice máis eficiente e flexible.

[1] Unha sinatura é unha representación concisa dun bloque de texto (ou documento) como unha secuencia de bits, similar ás pegadas dixitais comentadas no capítulo 3. Utilízase unha función hash para cada palabra do bloque de texto para establecer bits en posicións específicas no sinatura a un.

Un índice invertido é o equivalente computacional do índice que se atopa na parte traseira deste libro de texto. Pode que desexe ver o índice deste libro como exemplo. O índice do libro está ordenado alfabéticamente por *termo índice.* Cada termo índice vai seguido dunha lista de páxinas sobre esa palabra. Por exemplo, se queres saber máis sobre a derivación, buscarías o índice ata atopar palabras que comecen por "s". Despois, escanearías as entradas ata chegar á palabra "stemming". A lista de números de páxina conduciríache ao capítulo 4.

Do mesmo xeito, un índice invertido organízase por termo do índice. O índice é *invertido* porque normalmente pensamos en que as palabras forman parte dos documentos, pero se invertimos esta idea, os documentos asócianse ás palabras. Os termos do índice adoitan alfabeticarse coma un índice de libro tradicional, pero non é necesario que os atopen, xa que adoitan atoparse directamente usando unha táboa de hash. Cada termo índice ten o seu *lista invertida* que conteña os datos relevantes para ese termo. Nun índice dun libro, os datos relevantes son unha lista de números de páxina. Nun motor de busca, os datos poden ser unha lista de documentos ou unha lista de aparicións de palabras. Cada entrada de lista chámase a *publicación,* e a parte da publicación que fai referencia a un documento ou situación específica chámase a miúdo a *punteiro.* Cada documento da colección recibe un número único para facelo eficiente para almacenar os indicadores de documentos.

Os índices nos libros almacenan algo máis que información de localización. Para palabras importantes, a miúdo un dos números de páxina está marcado en negrita, o que indica que esta páxina contén unha definición ou discusión ampliada sobre o termo. Os ficheiros invertidos tamén poden ter información ampliada, onde as publicacións poden conter unha serie de información que non sexan só localizacións. Ao almacenar a información correcta xunto con cada publicación, as funcións de funcións que vimos na última sección pódense calcular de xeito eficiente.

Finalmente, por convención, os números de páxinas dun índice de libros imprímense en orde crecente, de xeito que os números de páxinas máis pequenos sexan os primeiros. Tradicionalmente, as listas inversas almacénanse do mesmo xeito. Estes *ordenado por documento* as listas están ordenadas por número de documento, o que fai que algúns tipos de procesamento de consultas sexan máis eficientes e tamén mellore a compresión de listas. Non obstante, algúns ficheiros invertidos que consideraremos teñen outro tipo de pedidos.

As alternativas aos ficheiros invertidos xeralmente teñen unha ou máis desvantaxes. O ficheiro sig- nature, por exemplo, representa cada documento da colección como un pequeno conxunto de bits. Para buscar un ficheiro de sinatura, a consulta convértese nunha sinatura e compáranse os patróns de bits. En xeral, todas as sinaturas deben ser dixitalizadas para cada busca. Mesmo se o índice está codificado de forma compacta, isto é moito procesado. A vantaxe do ficheiro invertido é que só hai que considerar unha pequena fracción do índice

para procesar a maioría das consultas. Ademais, as coincidencias nos ficheiros de sinatura son ruidosas, polo que non se garante que a coincidencia de sinaturas coincida no texto do documento. O máis importante, é difícil xeneralizar as técnicas de ficheiros de sinatura para a busca clasificada (Zobel et al., 1998).

Outro enfoque é utilizar estruturas de datos espaciais, como *kd árbores.* Neste enfoque, cada documento codifícase como un punto nalgún espazo de alta dimensión, e a consulta tamén. A estrutura de datos espaciais pode entón usarse para atopar os documentos máis próximos á consulta. Aínda que moitos enfoques de clasificación son fundamentalmente espaciais, a maioría das estruturas de datos espaciais non están deseñadas para o número de dimensións asociadas ás aplicacións de texto.[2] Como resultado, tende a ser moito máis rápido usar unha lista invertida para clasificar os documentos que usar unha estrutura de datos espaciais típica.

Nas seguintes seccións, veremos algúns tipos de ficheiros invertidos. En cada caso, a organización de ficheiros invertida está ditada pola función de clasificación. As funcións de clasificación máis complexas requiren máis información no índice. Estes índices máis complicados requiren espazo adicional e poder computacional para procesalos, pero pódense empregar para xerar unha clasificación efectiva dos documentos. A organización de índices non é de ningún xeito un problema resolto e continúanse investigando sobre o mellor xeito de crear índices que poidan producir de xeito máis eficiente clasificacións de documentos efectivas.

5.3.1 Documentos

A forma máis sinxela de lista invertida almacena só os documentos que conteñen cada palabra e sen información adicional. Este tipo de lista é semellante ao tipo de índice que atoparías ao final deste libro de texto.

A figura 5.3 mostra un índice deste tipo construído a partir das catro frases da táboa 5.1 (polo que neste caso, os "documentos" son frases). O índice contén todas as palabras que se atopan nas catro frases. Xunto a cada palabra, hai unha lista de cadros e cada un contén o número dunha frase. Cada unha destas caixas é unha publicación. Por exemplo, mire a palabra "peixe". Podes ver rapidamente que esta palabra aparece nas catro frases, porque aparecen os números 1, 2, 3 e 4. Tamén podes determinar rapidamente que "peixe" é a única palabra que aparece en todas as frases.

Dúas palabras achéganse: "tropical" aparece en cada frase pero S_4, e "auga" non está S_3.

[2] Cada termo dun documento corresponde a unha dimensión, polo que hai decenas de miles de dimensións en efecto. Isto é en comparación cunha aplicación de base de datos típica con decenas de dimensións como máximo.

S_1 Os peixes tropicais inclúen os peixes que se atopan en ambientes tropicais arredor no mundo, incluíndo especies de auga doce e de auga salgada.

S_2 Os pescadores adoitan empregar o termo peixes tropicais para referirse só a aqueles que require auga doce, con peixes tropicais de auga salgada chamados peixes mariños.

S_3 Os peixes tropicais son os peixes de acuario máis populares, debido a que son a miúdo brillantes coloración.

S_4 Nos peixes de auga doce, esta coloración normalmente deriva de iridescencia mentres que os peixes de auga salgada adoitan pigmentarse.

Táboa 5.1. Catro frases da entrada de Wikipedia para *peixes tropicais*

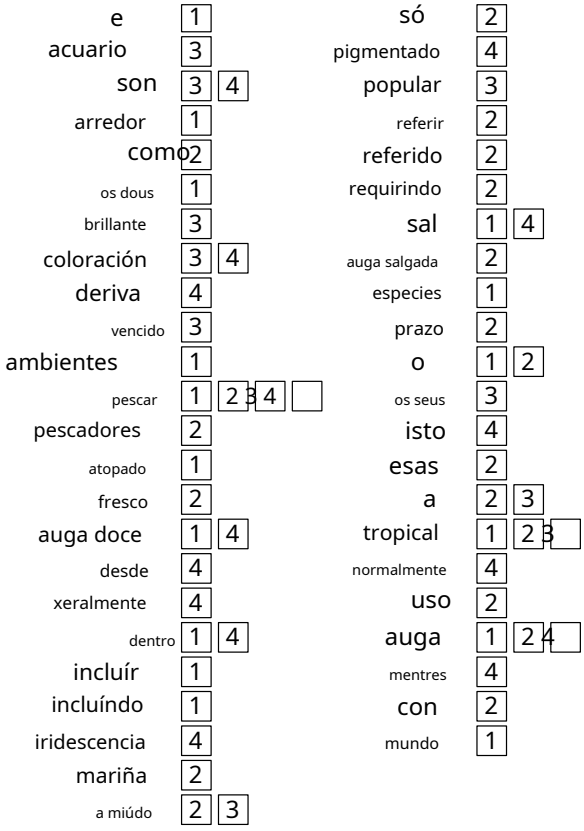

Fig. 5.3. Un índice invertido para os documentos (frases) da táboa 5.1

Teña en conta que este índice non rexistra o número de veces que aparece cada palabra; só rexistra os documentos nos que aparece cada palabra. Por exemplo, S_2 conten a palabra "peixe" dúas veces, mentres que S_1 contén "peixes" só unha vez. A lista invertida para "peixes" non mostra distinción entre as frases 1 e 2; ambos están listados no do mesmo xeito. Nas próximas seccións veremos índices que inclúen información sobre as frecuencias de palabras.

As listas invertidas son máis interesantes cando consideramos a súa intersección. Supoñamos que queremos atopar a frase que conteña as palabras "coloración" e "Auga doce". O índice invertido indícanos que aparece "coloración" en S_3 e S_4, mentres que aparece "auga doce" en S_1 e S_4. Isto só o podemos dicir rapidamente S_4 contén tanto "coloración" como "auga doce". Dado que cada lista está ordenada polo número da frase, atopar a intersección destas listas leva $O(\text{máximo}(m, n))$ tempo, onde m e n son as lonxitudes das dúas listas. O algoritmo é o mesmo que inmerge sort. Con lista *saltando,* que veremos máis adiante no capítulo, este custo cae $O(\text{min}(m, n))$.

5.3.2 Contas

Lembre que o noso modelo abstracto de clasificación considera que cada documento está composto por características. Cun índice invertido, cada palabra do índice corresponde a unha función de documento. Unha función de clasificación pode procesar os datos desta función nunha puntuación de documento. Nun índice invertido que só contén información do documento, as funcións son binarias, é dicir, son 1 se o documento contén un termo, 0 doutro xeito. Esta é unha información importante, pero é demasiado groseiro atopar os mellores documentos cando hai moitas coincidencias posibles.

Por exemplo, considere a consulta "peixes tropicais". Tres frases coinciden con isto consulta: S_1, S_2, e S_3. Os datos do índice baseado en documentos (Figura 5.3) non nos dan ningunha razón para preferir ningunha destas frases sobre ningunha outra.

Mire agora o índice da figura 5.4. Este índice ten un aspecto similar ao anterior. Aínda temos as mesmas palabras e o mesmo número de publicacións e o primeiro número de cada publicación é o mesmo que no índice anterior. Non obstante, cada publicación ten agora un segundo número. Este segundo número é o número de veces que aparece a palabra no documento. Esta pequena cantidade de datos adicionais permítenos prefire S_2 máis S_1 e S_3 para a consulta "peixes tropicais", xa que S_2 contén "tropical" dúas veces e "peixe" tres veces.

Neste exemplo, pode non ser obvio que S_2 é moito mellor que S_1 ou S_3, pero en xeral, o reconto de palabras pode ser un poderoso predictor de relevancia do documento. En en particular, o reconto de palabras pode axudar a distinguir documentos que se refiren a un determinado

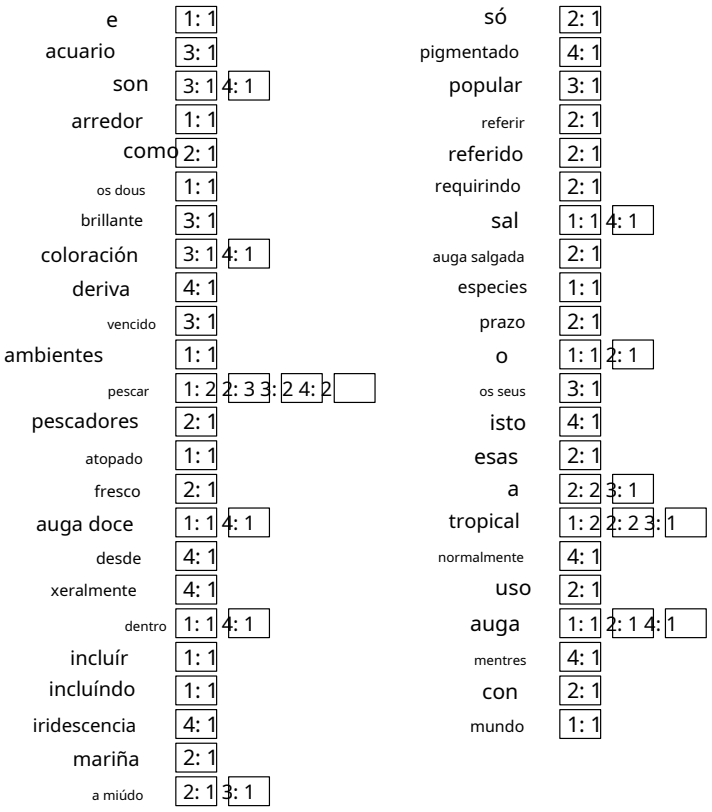

Fig. 5.4. Un índice invertido, con reconto de palabras, para os documentos da táboa 5.1

tema dos que discuten de paso ese tema. Imaxina dous documentos, un sobre peixes tropicais e outro sobre illas tropicais. O documento sobre as illas tropicais probablemente contería a palabra "peixe", pero só unhas cantas veces. Por outra banda, o documento sobre os peixes tropicais contería a palabra "peixe" moitas veces. O uso de contas de ocorrencia de palabras axúdanos a clasificar o documento máis relevante máis alto neste exemplo.

5.3.3 Posicións

Ao buscar coincidencias para unha consulta como "peixes tropicais", a localización das palabras no documento é un importante predictor de relevancia. Imaxina un documento sobre alimentos que incluía unha sección sobre froitas tropicais seguida dunha sección sobre peixes de auga salgada. Ata o de agora, ningún dos índices que consideramos contén información suficiente para indicarnos que este documento non é relevante. Aínda que un documento

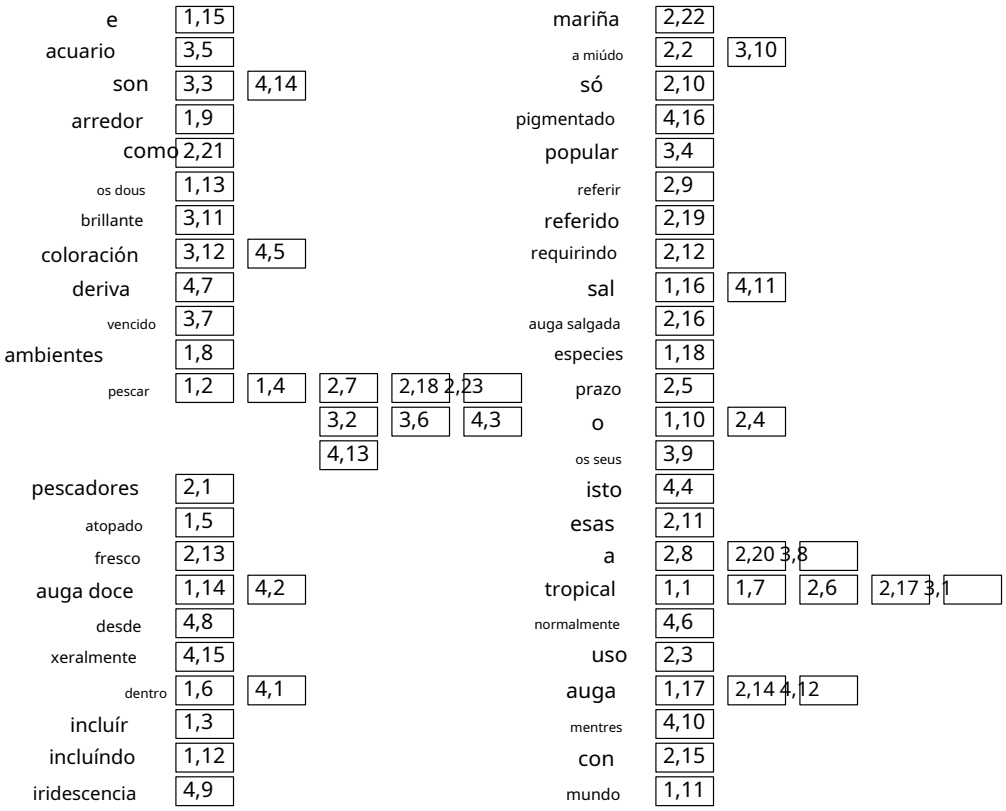

Fig. 5.5. Un índice invertido, con posicións de palabras, para os documentos da táboa 5.1

que conteña as palabras "tropical" e "peixe" é probable que sexa relevante, realmente queremos saber se o documento contén a frase exacta "peixe tropical".

Para determinalo, podemos engadir información de posición ao noso índice, como na figura 5.5. Este índice comparte algunhas características estruturais cos índices anteriores, xa que ten os mesmos termos de índice e cada lista contén algunhas publicacións. Non obstante, estas publicacións son diferentes. Cada publicación contén dous números: primeiro un número de documento, seguido dunha posición de palabra. Nos índices anteriores, só había unha publicación por documento. Agora hai unha publicación por palabra.

Mire a longa lista para a palabra "peixe". Nos outros índices, esta lista contiña só catro publicacións. Agora a lista contén nove publicacións. As dúas primeiras publicacións din nos que a palabra "peixe" é a segunda palabra e a cuarta palabra en S_1. As tres seguintes publicacións dinnos que "peixe" é a sétima, a décimo oitava e a vixésimo terceira palabra en S_2.

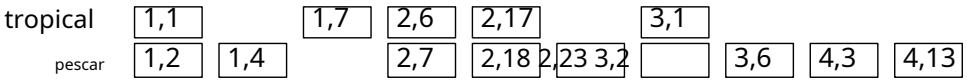

Fig. 5.6. Aliñando listas de publicación para "peixes tropicais" e "peixes" para atopar a frase "peixes tropicais"

Esta información é moi interesante cando observamos intersecciónscon outras listas de publicación. Usando unha intersección coa lista de "tropical", atopamos onde aparece a frase "peixe tropical". Na figura 5.6, as dúas listas invertidas están aliñadas a continuación entre si. Vemos que a palabra "tropical" é a primeira palabra en $S_{1,}$ e "peixe" é a segunda palabra en $S_{1,}$ o que significa iso S_1 debe comezar coa frase "tropical peixe ". A palabra "tropical" aparece de novo como a sétima palabra en $S_{1,}$ pero "peixe" non aparece como a oitava palabra, polo que non se trata dunha frase. En total, hai catro ocorrencias da frase "peixe tropical" nas catro frases. Os phrasematches son fáciles de ver na figura; ocorren nos puntos onde as publicacións están aliñadas en columnas.

Esta mesma técnica pódese estender para atopar frases máis longas ou expresións de proximidade máis xerais, como "atopar tropical dentro de 5 palabras de peixe. "Supoñamos que a palabra" tropical "aparece na posición *páx.* Despois podemos buscar na lista invertida "peixes" para calquera ocorrencia entre a posición *páx* - 5 e *p* + 5. Calquera desas ocorrencias constituiría unha coincidencia.

5.3.4 Campos e extensións

Os documentos reais non son só listas de palabras. Teñen frases e parágrafos que separan os conceptos en unidades léxicas. Algúns documentos teñen títulos e encabezados que fornecen resumos breves do resto do contido. Os tipos especiais de documentos teñen as súas propias seccións; por exemplo, cada correo electrónico contén información sobre o remitente e unha liña de asunto. Todos estes son exemplos do que chamaremos *campos de documento,* que son seccións de documentos que levan algún tipo de semántico.

Ten sentido incluír información sobre campos no índice. Por exemplo, supoña que ten un profesor chamado doutor Brown. O doutor Brown enviouche un correo electrónico sobre cando se deben presentar os proxectos do curso, pero non o atopas. Podes escribir "marrón" na caixa de busca do teu programa de correo electrónico, pero o resultado que queiras mesturarase con outros usos da palabra "marrón", como BrownUniversity ou calcetíns marróns. Unha busca de "marrón" no De: a liña do correo electrónico centrará a súa busca exactamente no que quere.

A información de campo é útil incluso cando non se usa explícitamente na consulta. Os títulos e as cabeceiras adoitan ser bos resumos do resto dun documento. Polo tanto, se un usuario busca "peixes tropicais", ten sentido preferir documentos co título "Peixes tropicais", aínda que un documento titulado "Mauricio" mencione as palabras "tropical" e "peixe" con máis frecuencia. Este tipo de preferencia por determinados campos de documentos pódese integrar na función de clasificación.

Para xestionar este tipo de buscas, o motor de busca ten que ser capaz de determinar se unha palabra está nun campo concreto. Unha opción é facer listas invertidas separadas para cada tipo de campo de documento. Esencialmente, podería crear un índice para títulos de documentos, un para cabeceiras de documentos e outro para texto corporal. Buscar palabras no título é tan sinxelo como buscar no índice do título. Non obstante, atopar unha palabra en calquera sección do documento é máis complicado, xa que é preciso buscar listas invertidas de moitos índices diferentes para tomar esa determinación.

Outra opción é almacenar información en cada publicación de palabras sobre onde se produciu a palabra. Por exemplo, poderiamos especificar que o número 0 indica un título e 1 indica o texto do corpo. Cada publicación da lista invertida contería un 0 ou un 1 ao final. Estes datos poderían usarse para determinar rapidamente se unha publicación estaba nun título e só requiriría un bit por publicación. Non obstante, se tes máis campos que un título, a representación medrará.

Estas dúas suxestións teñen problemas cando se atopan con estruturas de documentos máis complicadas. Por exemplo, supoña que é conveniente indexar os libros. Algúns libros, coma este, teñen máis dun autor. Nalgún lugar da descrición XML deste libro, podes atopar:

<autor> W. Bruce Croft </author>,
<author> Donald Metzler </author> e
<author> Trevor Strohman </author>

Supoña que lle gustaría atopar libros dun autor chamado Croft Donald. Se escribe a frase consulta "Croft donald" nun motor de busca, debería coincidir este libro? Nela aparecen as palabras "croft" e "donald" e, de feito, aparecen unhas ao lado das outras. Non obstante, están en dous campos de autor distintos. Probablemente non sexa bo para a consulta "Croft donald", pero os dous métodos anteriores para tratar con campos (bits na lista de publicación, índices separados) non poden facer este tipo de distinción.

Aquí é onde *listas de extensións* entra. An *extensión* é unha rexión contigua dun documento. Podemos representar estas extensións empregando posicións de palabras. Por exemplo, se o título dun libro comezou na quinta palabra e rematou xusto antes da novena palabra,

poderiamos codificar iso como (5,9). Para o texto do autor mostrado anteriormente, poderiamos escribir autor: (1,4), (4,6), (7,9). O (1,4) significa que as tres primeiras palabras ("W. Bruce Croft") constitúen o primeiro autor, seguido do segundo autor ("DonaldMetzler"), que son dúas palabras. A palabra "e" non está nun campo de autor, pero si as dúas seguintes palabras si, polo que a última publicación é (7,9).

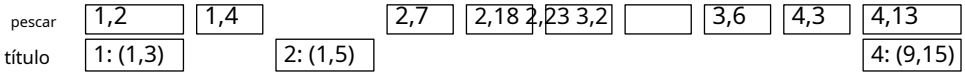

Fig. 5.7. Aliñando listas de publicación de "peixe" e título para atopar coincidencias da palabra "peixe" no campo de título dun documento.

A figura 5.7 mostra como funciona isto na práctica. Aquí temos a mesma lista de publicacións de posicións para "peixes" que usamos no exemplo anterior. Tamén temos unha lista extensa para o campo de título. Por máis claridade, hai lagoas nas listas de publicacións para que as publicacións axeitadas se alinean unha á beira da outra. Ao comezo das dúas listas, vemos que o documento 1 ten un título que contén as dúas primeiras palabras (1 e 2, que remata xusto antes da terceira palabra). Sabemos que este título inclúe a palabra "peixe", porque a lista invertida de "peixe" indícanos que "peixe" é a segunda palabra do documento 1. Se o usuario quere atopar documentos coa palabra "peixe" no título , o documento 1 é igual. O documento 2 non coincide, porque o seu título remata só antes da quinta palabra, pero "peixe" non aparece ata a sétima palabra. Ao parecer, o documento 3 non ten ningún título, polo que non son posibles coincidencias. O documento 4 ten un título que comeza na novena palabra (quizais o documento comeza cunha data ou unha declaración de autor) e contén a palabra "peixe". En total, este exemplo mostra dous documentos coincidentes: 1 e 4.

Este concepto pódese estender a todo tipo de campos, como títulos, parágrafos ou frases. Tamén se pode usar para identificar anacos de texto máis pequenos cun significado específico, como enderezos ou nomes, ou incluso só para rexistrar que palabras son verbos.

5.3.5 Puntuacións

Se as listas invertidas vanse empregar para xerar valores de funcións de funcións, por que non só almacenar o valor da función de funcións? Isto é certamente posible, e algúns motores de busca moi eficientes fan isto. Esta visión permite almacenar valores de funcións de característica que serían demasiado computacionalmente intensos para computar

durante a fase de procesamento da consulta. Tamén elimina a complexidade do motor de procesamento de consultas e do código de indexación, onde pode ser máis tolerable.

Fagamos isto máis concreto. Na última sección, houbo un exemplo sobre como se debería preferir un documento co título "Peixes tropicais" sobre un documento "Mauricio" para a consulta "peixe tropical", aínda que o documento de Mauricio conteña as palabras "tropical" e "peixe" moitas veces. O cálculo das puntuacións que reflicten esta preferencia require certa complexidade no momento da avaliación da consulta. As publicacións de "peixes tropicais" deben separarse en grupos, polo que sabemos cales están no título e cales non. Despois, temos que definir algunha puntuación para as publicacións de títulos e as publicacións que non sexan de título e mesturar eses números, e isto cómpre facelo para todos os documentos.

Un enfoque alternativo consiste en almacenar o valor final na lista invertida. Poderiamos facer unha lista para "peixes" que contén publicacións como [(1: 3.6), (3: 2.2)], o que significa que o valor total da característica para "peixe" no documento 1 é 3,6 e no documento 3 é 2,2 . É de supoñer que o número 3.6 veu tendo en conta cantas veces aparecían "peixes" no título, nas cabeceiras, en letras grandes, en negra e en ligazóns ao documento. Quizais o documento non conteña a palabra "peixe" en absoluto, pero en cambio moitos nomes de peixes, como "carpa" ou "troita". O valor 3,6 é entón un indicador de canto se trata deste documento sobre os peixes.

Almacenar puntuacións coma esta aumenta e diminúe a flexibilidade do sistema. Aumenta a flexibilidade porque se fai posíbel unha puntuación computacionalmente custosa, unha boa parte do traballo duro de puntuación de documentos insírese no índice. Non obstante, pérdese a flexibilidade, xa que xa non podemos cambiar o mecanismo de puntuación unha vez que se constrúe o índice. O que é máis importante, a información sobre a proximidade de palabras desapareceu neste modelo, o que significa que non podemos incluír información de frases na puntuación a menos que tamén creamos listas invertidas para frases. Estas listas de frases precomputadas requiren un espazo adicional considerable.

5.3.6 Ordenación

Ata o de agora, asumimos que as publicacións de cada lista invertida estarían ordenadas polo número do documento. Aínda que esta é a opción máis popular, esta non é a única forma de pedir unha lista invertida. Tamén se pode ordenar unha lista invertida por puntuación, de xeito que os documentos con maior puntuación sexan os primeiros. Isto ten sentido só cando as listas xa gardan a puntuación ou cando é probable que só se computa un tipo de puntuación da lista invertida. Ao almacenar puntuacións en lugar de documentos, o motor de procesamento de consultas pode concentrarse só na parte superior de cada lista invertida, onde o

rexístranse os documentos con maior puntuación. Isto é especialmente útil para consultas que constan dunha soa palabra. Nunha lista invertida tradicional ordenada por documento, o motor de procesamento de consultas tería que escanear toda a lista para atopar a parte superior k documentos de puntuación, mentres que só faría falta ler o primeiro k publicacións nunha lista ordenada por puntuacións.

5.4 Compresión

Existen moitas formas diferentes de almacenar información dixital. Normalmente facemos unha distinción simple entre almacenamento persistente e transitorio. Usamos un almacenamento persistente para almacenar cousas en ficheiros e directorios que queremos gardar ata que escollamos eliminalas. Disos, CD, DVD, memoria flash e cinta magnética úsanse normalmente para este propósito. A RAM dinámica (Random Access Memory), por outra banda, úsase para almacenar información transitoria, que é información que só necesitamos mentres o ordenador está en funcionamento. Agardamos que cando volvamos o ordenador, toda esa información desapareza.

Podemos facer distincións máis finas entre os tipos de almacenamento en función da velocidade e da capacidade. A cinta magnética é lenta, os discos son máis rápidos, pero a memoria dinámica é moito máis rápida. Os ordenadores modernos son tan rápidos que nin a RAM dinámica é o suficientemente rápida como para manterse, polo que os microprocesadores conteñen polo menos dous niveis de memoria caché. O tipo de memoria máis rápido compón os rexistros do procesador. Nun mundo perfecto, poderiamos usar rexistros ou memoria caché para todo o almacenamento transitorio, pero é demasiado caro para ser práctico.

A realidade, entón, é que os ordenadores modernos conteñen un *xerarquía da memoria.* Na parte superior da xerarquía temos unha memoria pequena, pero rápida. A base consiste nunha memoria enorme, pero lenta. O rendemento dun motor de busca depende en gran medida de como faga uso das propiedades de cada tipo de memoria.

As técnicas de compresión son a ferramenta máis poderosa para xestionar a xerarquía da memoria. As listas invertidas para unha gran colección son moi grandes. De feito, cando inclúe información sobre a posición das palabras e as extensións do documento, o índice pode ter un tamaño comparable[3] á colección de documentos. A compresión permite almacenar os mesmos datos da lista invertida en menos espazo. O beneficio obvio é que isto podería reducir os requirimentos de disco ou memoria, o que aforraría

[3] Como exemplo, os índices para colecciónS TREC construídos usando o motor de busca de código aberto Indri van do 25 ao 50% do tamaño da colección. A figura inferior é para unha colección de páxinas web.

cartos. Máis importante aínda, a compresión permite que os datos suban á xerarquía de memoria. Se os datos do índice están comprimidos por un factor de catro, podemos almacenar catro veces máis datos útiles na caché do procesador e podemos enviarlle datos ao procesador catro veces máis rápido. No disco, a compresión tamén comprime os datos máis xuntos, o que reduce os tempos de busca. Sistemas multiprocesadores e multiprocesadores, onde moitos procesadores comparten un sistema de memoria, a compresión de datos permite aos procesadores compartir o ancho de banda da memoria de xeito máis eficiente.

Por desgraza, nada é gratuíto. O aforro de espazo na compresión ten un custo: o procesador debe descomprimir os datos para usalos. Polo tanto, non é suficiente escoller a técnica de compresión que poida almacenar a maior cantidade de datos na menor cantidade de espazo. Para aumentar o rendemento xeral, necesitamos escoller unha técnica de compresión que reduza o espazo e sexa fácil de descomprimir.

Para velo matemáticamente, supoña que algún procesador pode procesar *páx* publicacións de listas invertidas por segundo. Este procesador está conectado a un sistema de memoria co que pode fornecer o procesador *m* publicacións cada segundo. O número de envíos procesados cada segundo é entón min (*m, p).* Se *p> m,* entón o procesador pasará parte do seu tempo á espera de que cheguen as publicacións da memoria. Se *m> p,* o sistema de memoria ás veces estará inactivo.

Supoñamos que introducimos compresión no sistema. O noso sistema de compresión ten unha relación de compresión de *r,* o que significa que agora podemos gardalo *r* publicacións na mesma cantidade de espazo que unha publicación sen comprimir. Isto permite que o procesador lea *Señor* publicacións cada segundo. Non obstante, o procesador primeiro debe descomprimir cada publicación antes de procesala. Isto ralentiza o procesamento por un factor de descompresión, *d,* e deixa o proceso do procesador *dp* publicacións cada segundo. Agora podemos procesar min (*señor, dp)* publicacións cada segundo.

Cando non usamos ningunha compresión, $r = 1$ e $d = 1$. Calquera técnica de compresión razoable dános $r > 1$, pero $d < 1$. Podemos ver que a compresión é unha técnica de rendemento útil só cando o *p> m,* é dicir, cando o procesador pode procesar os datos da lista invertida máis rápido do que o sistema de memoria pode subministralos. Aumentará un esquema de compresión moi sinxelo *r* un pouco e reducir *d* un pouco. Aumentará un esquema de compresión complicado *r* moito, mentres se reduce *d* unha morea. Ideal sería querer escoller un esquema de compresión tal que min (*señor, dp)* maximízase, o que debería ocorrer cando *mr = dp.*

Nesta sección, considerámolo só *sen perdas* técnicas de compresión. As técnicas sen perdas almacenan datos en menos espazo, pero sen perder información. Tamén os hai *con perda*

técnicas de compresión de datos, que adoitan empregarse para vídeo, imaxes e audio. Estas técnicas logran índices de compresión moi altos (*r* no noso anterior debate-

sión), pero faino botando os datos menos importantes. As técnicas de poda de listas invertidas, que comentaremos máis adiante, poderían considerarse unha técnica de compresión con perdas, pero normalmente cando falamos de compresión só entendemos métodos sen perdas.

En particular, o noso obxectivo con estas técnicas de compresión é reducir o tamaño das listas invertidas que comentamos anteriormente. As técnicas de compresión desta sección son especialmente adecuadas para números de documentos, reconto de palabras e información sobre a posición do documento.

5.4.1 Entropía e ambigüidade

A estas alturas do libro, xa viches moitos exemplos de distribucións de probabilidade. As técnicas de compresión tamén se basean en probabilidades. A idea fundamental detrás da compresión é representar elementos de datos comúns con códigos curtos mentres se representan elementos de datos pouco comúns con códigos máis longos. As listas invertidas que comentamos son esencialmente listas de números e, sen compresión, cada número ocupa a mesma cantidade de espazo. Dado que algúns deses números son máis frecuentes que outros, se codificamos os números frecuentes con códigos curtos e os números pouco frecuentes con códigos máis longos, podemos acabar aforrando espazo.

Por exemplo, consideremos os números 0, 1, 2 e 3. Podemos codificar estes números usando dous bits binarios. Unha secuencia de números, como:

0, 1, 0, 3, 0, 2, 0

pódese codificar nunha secuencia de díxitos binarios:

00 01 00 10 00 11 00

Teña en conta que os espazos da secuencia binaria están aí para deixar claro onde comeza e para cada número e non son realmente parte da codificación.

Na nosa secuencia de exemplo, o número 0 aparece catro veces, mentres que cada un dos outros números aparece só unha vez. Podemos decidir aforrar espazo codificando 0 usando só un bit 0. O noso primeiro intento de codificación pode ser:

0 01 0 10 0 11 0

Parece moi exitoso porque esta codificación emprega só 10 bits no canto dos 14 bits empregados anteriormente. Non obstante, esta codificación é *ambiguo,* o que significa que non o é

claro como decodificalo. Lembre que os espazos do código só están aí para a nosa comodidade e non están realmente almacenados. Se engadimos algúns espazos diferentes, chegamos a unha interpretación perfectamente válida desta codificación:

0 01 01 0 0 11 0

que, cando se decodifica, convértese en:

0, 1, 1, 0, 0, 3, 0

Por desgraza, estes non son os datos que codificamos. O problema é que cando o vemos 010 nos datos codificados, non podemos estar seguros de se (0, 2) ou (1, 0) foi codificado.

A codificación sen comprimir non era ambigua. Sabiamos exactamente onde colocar os espazos porque sabiamos que cada número levaba exactamente 2 bits. No noso código comprimido, os números codificados consumen 1 ou 2 bits, polo que non está claro onde colocar os espazos. Para resolver este problema, debemos restrinxirnos a *unam- biguous* códigos, que se denominan confusamente ambos *códigos de prefixo* e *códigos sen prefixos.*
Un código inequívoco é aquel no que só hai un xeito válido de colocar espazos en datos codificados.

Fixemos o noso código para que non sexa ambiguo:

Código de número	
0	0
1	101
2	110
3	111

Isto resulta na seguinte codificación:

0 101 0 111 0 110 0

Esta codificación require 13 bits en lugar dos 14 bits requiridos pola versión sen comprimir, polo que aínda estamos aforrando algo de espazo. Non obstante, a diferenza do último código que consideramos, este non é ambiguo. Teña en conta que se un código comeza con 0, consome 1 bit; se un código comeza con 1, ten 3 bits de lonxitude. Isto dános un algoritmo determinista para colocar espazos no fluxo codificado.

Na sección "Exercicios", demostrará que non existe un código pouco ambiguo que poida comprimir todas as entradas posibles; algunhas entradas serán máis grandes. É por iso que é tan importante saber algo sobre que tipo de datos temos

quere codificar. No noso exemplo, observamos que o número 0 aparece con frecuencia e podemos usar ese feito para reducir a cantidade de espazo que require a versión codificada. *Entropía* mide a previsibilidade da entrada. No noso caso, a entrada parece algo previsible, porque é máis probable que apareza o número 0 que outros números. Aproveitamos esta entropía para producir un código utilizable para os nosos propósitos.

5.4.2 Codificación Delta

Todas as técnicas de codificación que consideraremos neste capítulo supoñen que é máis probable que se produzan pequenas cantidades que as grandes. Este é un suposto excelente para os datos do reconto de palabras; moitas palabras aparecen unha soa vez nun documento e algunhas aparecen dúas ou tres veces. Só un pequeno número de palabras aparecen máis de 10 veces. Polo tanto, ten sentido codificar números pequenos con códigos pequenos e números grandes con códigos grandes.

Non obstante, os números do documento non comparten esta propiedade. Agardamos que unha lista invertida típica conteña algúns números de documentos pequenos e algúns números de documentos moi grandes. É certo que algúns documentos conteñen máis palabras e, polo tanto, aparecerán máis veces nas listas invertidas, pero doutro xeito non hai moita entropía na distribución dos números de documentos nas listas invertidas.

A situación é diferente se temos en conta as diferenzas entre os números do documento en lugar dos números do documento. Lembre que as publicacións das listas invertidas normalmente están ordenadas polo número do documento. Unha lista invertida sen contas, por exemplo, é só unha lista de números de documento, como estes:

1, 5, 9, 18, 23, 24, 30, 44, 45, 48

Dado que estes números de documentos están ordenados, sabemos que cada número de documento da secuencia é máis que o anterior e inferior ao posterior. Este feito permítenos codificar a lista de números polas diferenzas entre números de documento adxacentes:

1, 4, 4, 9, 5, 1, 6, 14, 1, 3

Esta lista codificada comeza con 1, o que indica que 1 é o primeiro número de documento. A seguinte entrada é 4, o que indica que o segundo número de documento é 4 máis que o primeiro: 1 + 4 = 5. O terceiro número, 4, indica que o terceiro número de documento é 4 máis que o segundo: 5 + 4 = 9.

Este proceso chámase *codificación delta,* e as diferenzas chámanse a miúdo *ocos d.*

Teña en conta que a codificación delta non define os patróns de bits que se usan para almacenar

os datos, polo que non aforra espazo por si só. Non obstante, a codificación delta é especialmente exitosa ao cambiar unha lista ordenada de números por unha lista de números pequenos. Dado que estamos a piques de discutir métodos para comprimir listas de pequenos números, esta é unha propiedade útil.

Antes de seguir adiante, considere as listas invertidas para as palabras "entropía" e "quen". A palabra "quen" é moi común, polo que esperamos que a maioría dos documentos a conteñan. Cando empregamos a codificación delta na lista invertida para "quen", esperaríamos ver moitos pequenos ocos en d, como:

1, 1, 2, 1, 5, 1, 4, 1, 1, 3, ...

Pola contra, a palabra "entropía" raramente aparece no texto, polo que só algúns documentos a conteñen. Polo tanto, esperaríamos ver maiores diferenzas en d, como:

109, 3766, 453, 1867, 992, ...

Non obstante, dado que "entropía" é unha palabra rara, esta lista de grandes cantidades non será moi longa. En xeral, atoparemos que as listas invertidas para termos frecuentes se comprimen moi ben, mentres que os termos pouco frecuentes se comprimen menos ben.

5.4.3 Códigos aliñados en bits

O código que inventamos na sección 5.4.1 é un código aliñado en bits, o que significa que as roturas entre as rexións codificadas (os espazos) poden ocorrer despois de calquera posición de bit. Nesta sección describiremos algúns códigos populares aliñados en bits. Na seguinte sección, imos discutir métodos onde as palabras de código están restrinxidas ao terminar nos límites de bytes. En todas as técnicas que comentaremos, estamos a buscar formas de almacenar pequenos números en listas invertidas (como o reconto de palabras, as posicións de palabras e os números de documento codificado por delta) no menor espazo posible.

Un dos códigos máis sinxelos é o código unario. Probablemente estea familiarizado co binario, que codifica números con dous símbolos, normalmente 0 e 1. Un sistema numérico unario é unha codificación base-1, o que significa que usa un único símbolo para codificar números. Aquí tes algúns exemplos:

	Código de número
0	0
1	10
2	110
3	1110
4	11110
5	111110

En xeral, para codificar un número k en unario, saímos k 1s, seguido dun 0. Necesitamos o 0 ao final para que o código sexa inequívoco.

Este código é moi eficiente para números pequenos como 0 e 1, pero axiña resulta moi caro. Por exemplo, o número 1023 pode representarse en 10 bits binarios, pero require 1024 bits para representalo en código unario.

Agora sabemos de dous tipos de codificacións numéricas. Unario é conveniente porque é compacto para pequenos números e é inherentemente inequívoco. O binario é unha mellor opción para grandes cantidades, pero non é inherentemente inequívoco. Un esquema de compresión razoable necesita codificar números frecuentes con menos bits que números pouco frecuentes, o que significa que a codificación binaria non é útil por si mesma para a compresión.

Elías- γ códigos

The Elias- γ (O código Elias gamma) combina os puntos fortes dos códigos unarios e binarios. Para codificar un número k usando este código, calculamos dúas cantidades:

- $k_d = \lfloor \text{rexistro}_2 k \rfloor$
- $k_r = k - 2^{\lfloor \text{rexistro}_2 k \rfloor}$

Supoñamos que escribiches k formulario de entrada. O primeiro valor, k_d, é entón o número de díxitos binarios que necesitaría escribir. Supoñendo $k > 0$, o díxito binario máis á esquerda de k é 1. Se borra ese díxito, os restantes díxitos binarios son k_r.

Se codificamos k_d en unario e k_r en binario (en k_d cifras binarias), obtemos o Elias- γ código. Na táboa 5.2 móstranse algúns exemplos.

Número (k)	k_d	k_r	Código
1	0	0	0
2	1	0	10 0
3	1	1	10 1
6	2	2	110 10
15	3	7	1110 111
16	4	0	11110 0000
255	7	127	11111110 1111111
1023	9	511	1111111110 111111111

Táboa 5.2. Elías- γ exemplos de código

O truco con este código é que a parte unaria do código indícanos cantos bits podemos esperar na parte binaria. Rematamos cun código que usa bits nomores que o código unario para calquera número e para números maiores de 2 usa menos bits . O aforro para grandes cantidades é substancial. Por exemplo, agora podemos codificar 1023 en 19 bits, en vez de 1024 usando só código unario.

Para calquera número k, o Elías- γ o código require $\lfloor \textit{rexistro}_2 k \rfloor + 1$ bits para k_d en unario código e $\lfloor \textit{rexistro}_2 k \rfloor$ bits para k_r en binario. Polo tanto, $2\lfloor \textit{rexistro}_2 k \rfloor + 1$ os bits son necesarios en todos.

Elías- δ códigos

Aínda que o Elias- γ O código é unha mellora importante no código unario, non é ideal para entradas que poidan conter grandes cantidades. Sabemos que un número k pode expresarse en rexistro $_2 k$ díxitos binarios, pero o Elias- γ o código require o dobre de bits para facer a codificación inequívoca.

The Elias- δ código tenta resolver este problema cambiando a forma en que k_d está codificado. En vez de codificar k_d en unario, podemos codificar $k_d + 1$ en Elías- γ. En particular, dividímonos k_d en:

- $k_{dd} = \lfloor$ rexistro $_2 (k_d + 1) \rfloor$
- $k_{dr} = (k_d + 1) - 2\lfloor$ rexistro $_2 (k_d + 1) \rfloor$

Teña en conta que usamos $k_d + 1$ aquí, desde entón k_d pode ser cero, pero rexistro $_2 0$ está indefinido.

Logo codificamos k_{dd} inunario, k_{dr} binario e k_r inbinario. O valor de k_{dd} é a lonxitude de k_{dr}, e k_{dr} é a lonxitude de k_r, o que fai que este código sexa inequívoco. A táboa 5.3 ofrece algúns exemplos de Elias- δ codificacións.

Número (k)	k_d	k_r	k_{dd}	k_{dr}	Código
1	0	0	0	0	
2	1	0	1 0	10	0 0
3	1	1	1 0	10	0 1
6	2	2	1 1	10	1 10
15	3	7	2 0	110	00 111 0 2 1
16	4	1	10 0	1 00	00
255	7	127	3 0	1110	000 1111111 1023 9
511	8	255	3 1	1110	010 111111111

Táboa 5.3. Elías- δ exemplos de código

Elías- δ sacrifica certa eficacia por números pequenos para gañar eficacia na codificación de números maiores. Teña en conta que o código do número 2 aumentou a 4 bits no canto dos 3 bits requiridos polo Elias- γ código. Non obstante, para números maiores de 16, o Elias- δ o código require un espazo máis grande que o Elias- γ código, e para números maiores de 32, o Elias- δ require menos espazo.

En concreto, o Elias- γ o código require $\lfloor$ rexistro $_2 (\lfloor$ rexistro $_2 k \rfloor + 1) \rfloor + 1$ bits para k_{dd} en unario, seguido de $\lfloor$ rexistro $_2 (\lfloor$ rexistro $_2 k \rfloor + 1) \rfloor$ bits para k_{dr} en binario e $\lfloor$ rexistro $_2 k \rfloor$ bits para k_r en binario. O custo total é de aproximadamente 2 rexistro $_2$ rexistro $_2 k +$ rexistro $_2 k$.

5.4.4 Códigos aliñados en bytes

Aínda que algúns trucos poden axudarnos a decodificar códigos aliñados a bits rapidamente, os códigos de lonxitude de bits variable son engorrosos nos procesadores que procesan bytes. O procesador está construído para manexar bytes de xeito eficiente e non bits, polo que é lóxico que os códigos aliñados en bytes sexan máis rápidos na práctica.

Hai moitos exemplos de esquemas de compresión aliñados en bytes, pero aquí só consideramos un método popular. Este é o código comúnmente coñecido como *v-byte,* que é unha abreviatura de "lonxitude de byte variable". O método v-byte é moi similar á codificación UTF-8, que é un xeito popular de representar texto (ver sección 3.5.1).

Do mesmo xeito que os outros códigos que estudamos ata agora, o método v-bytemethod usa códigos curtos para números pequenos e códigos máis longos para números máis longos. Non obstante, cada código é unha serie de bytes, non bits. Entón, o código de byte v máis curto para un único enteiro é un byte. Nalgunhas circunstancias, isto podería ser moi inespecífico; a codificación do número 1 leva oito veces máis espazo en v-byte que en Elias- γ. Normalmente, a diferenza no uso do espazo non é tan dramática.

O código de byte v é realmente sinxelo. Os sete bits baixos de cada byte conteñen datos numéricos en binario. O bit alto é un bit terminador. O último byte de cada código ten o seu bit alto definido en 1; se non, establécese en 0. Calquera número que se poida representar en sete díxitos binarios require un byte para codificar. Na táboa 5.4 móstrase máis información sobre o uso do espazo.

Na táboa 5.5 móstranse algúns exemplos de codificación. Os números inferiores a 128 almacénanse nun só byte en forma binaria tradicional, excepto que o bit alto está definido. Para números maiores, os sete bits menos significativos almacénanse no primeiro byte. Os seguintes sete bits almacénanse no seguinte byte ata que se almacenan todos os bits distintos de cero.

Almacenar datos comprimidos cun código aliñado en bytes ten moitas vantaxes sobre un código aliñado en bits. Os códigos aliñados en bytes comprímense e descomprimense máis rápido, xa que

k	Número de bytes
$k < 2^7$	1
$2^7 \leq k < 2^{14}$	2
$2^{14} \leq k < 2^{21}$	3
$2^{21} \leq k < 2^{28}$	4

Táboa 5.4. Necesidades de espazo para números codificados en v-byte

k	Código binario	hexadecimal
1	1 0000001	81
6	1 0000110	86
127	1 1111111	FF
128	0 0000001 1 0000000	01 80
130	0 0000001 1 0000010	01 82
20000	0 0000001 0 0011100 1 0100000	01 1CA0

Táboa 5.5. Codificación de mostra para v-byte

os cesadores (e linguaxes de programación) están deseñados para procesar bytes en lugar de bits. Por estas razóns, o motor de busca Galago asociado a este libro usa v-byte exclusivamente para compresión.

5.4.5 Compresión na práctica

As técnicas de compresión que tratamos úsanse para codificar listas invertidas en sistemas reais de recuperación. Nesta sección veremos como Galago usa a compresión para codificar listas invertidas no PositionListWriter clase.

A figura 5.5 ilustra como se pode almacenar a información de posición en listas invertidas. Considere só a lista invertida para tropical:

(1, 1) (1, 7) (2, 6) (2, 17) (3, 1)

En cada par, o primeiro número representa o documento e o segundo número representa a posición da palabra. Por exemplo, a terceira entrada desta lista indica que a palabra tropical é a sexta palabra do documento 2. Porque axuda ao exemplo, engadiremos (2, 197) á lista:

(1, 1) (1, 7) (2, 6) (2, 17) (2, 197) (3, 1)

Podemos agrupar as posicións de cada documento de xeito que cada documento teña a súa propia entrada (documento, reconto, [posicións]), onde reconto é o número de ocorrencias do documento. Os nosos datos de exemplo agora teñen este aspecto:

(1, 2, [1, 7]) (2, 3, [6, 17, 197]) (3, 1, [1])

O reconto de palabras é importante porque fai que esta lista sexa descifrable aínda sen parénteses e soportes. O reconto indícanos cantas posicións se atopan entre parénteses e podemos interpretar estes números sen ambigüidades, aínda que estivesen impresos do seguinte xeito:

1, 2, 1, 7, 2, 3, 6, 17, 197, 3, 1, 1

Non obstante, por agora deixamos os corchetes no seu lugar para máis claridade.

Son números pequenos, pero coa codificación delta podemos facelos máis pequenos. Teña en conta que os números do documento están ordenados en orde ascendente, porque pode codificar delta con seguridade para codificalos:

(1, 2, [1, 7]) (1, 3, [6, 17, 197]) (1, 1, [1])

A segunda entrada comeza agora cun 1 en lugar dun 2, pero esta 1 significa que "este número de documento é un máis que o último número de documento". Dado que a información de posición tamén se ordena en orde ascendente, tamén podemos codificar por delta as posicións:

(1, 2, [1, 6]) (1, 3, [6, 11, 180]) (1, 1, [1])

Non podemos codificar por delta o reconto de palabras, porque non están en orde ascendente. Se os codificamos por delta, algúns dos deltas poden ser negativos e as técnicas de compresión que comentamos non manexan números negativos sen un traballo adicional.

Agora podemos eliminar os corchetes e considerar esta lista invertida como só unha lista de números:

1, 2, 1, 6, 1, 3, 6, 11, 180, 1, 1, 1

Dado que a maioría destes números son pequenos, podemos comprimalos con v-byte para aforrar espazo:

81 82 81 86 81 83 86 8B 01 B4 81 81 81

O 01 B4 é 180, que está codificado en dous bytes. O resto dos números codificáronse como bytes individuais, dando un total de 13 bytes para toda a lista.

5.4.6 Mirando cara adiante

Esta sección describiu tres esquemas de compresión para listas invertidas, e hai moitos outros de uso común. Aínda que a compresión é unha das áreas máis antigas da informática, cada ano desenvólvense novos esquemas de compresión.

Por que son necesarios estes novos esquemas? Lembre ao comezo desta sección falamos de como a compresión nos permite intercambiar o cálculo do procesador polo rendemento de datos. Isto significa que a mellor opción para un algoritmo de compresión está estreitamente unida ao estado das modernas CPUs e sistemas de memoria. Durante moito tempo, a velocidade da CPU aumentou moito máis rápido que o rendemento da memoria, polo que os esquemas de compresión con proporcións de compresión máis altas fixéronse máis atractivos. Non obstante, a tendencia de hardware dominante agora é cara a moitos núcleos de CPU con velocidades de reloxo máis baixas. Dependendo do rendemento da memoria destes sistemas, as relacións de compresión máis baixas poden ser atractivas.

Máis importante aínda, as CPU modernas deben gran parte da súa velocidade a trucos intelixentes como a predición de ramas, o que axuda ao procesador a adiviñar como se executará o código. O código máis predicible pode funcionar moito máis rápido que o código imprevisible. Moitos dos esquemas de compresión máis recentes están deseñados para facer a fase de descodificación máis predecible e, polo tanto, máis rápida.

5.4.7 Saltar e saltar punteiros

Para moitas consultas, non necesitamos toda a información almacenada nunha lista invertida particular. Pola contra, sería máis eficiente ler só a pequena parte dos datos relevantes para a consulta. Os saltos nos axudan a acadar ese obxectivo.

Considere a consulta booleana " galago E animal ". A palabra "animal" aparece nuns 300 millóns de documentos na rede fronte a aproximadamente 1 millón para "galago". Se supoñemos que as listas invertidas para "galago" e "animal" están na orde do documento, hai un algoritmo moi sinxelo para procesar esta consulta:

• Deixe d_g ser o primeiro número de documento da lista invertida para "galago".
• Deixe d_a ser o primeiro número de documento da lista invertida para "animal".
• Mentres aínda hai documentos nas listas do bucle "galago" e "animal":
  - Se $d_g < d_a$, conxunto d_g ao seguinte número de documento na lista "galago".
  - Se $d_a < d_g$, conxunto d_a ao seguinte número de documento na lista de "animais".

- Se $d_a = d_g$, o documento d_a contén "galago" e "animal". Mover os dous d_g e d_a aos seguintes documentos das listas invertidas para "galago" e "animal" respectivamente.

Desafortunadamente, este algoritmo é moi caro. Procesa case todos os documentos nas dúas listas invertidas, polo que esperamos que a computadora procese este lazo unhas 300 millóns de veces. Pasarase máis do 99% do tempo de procesamento procesando os 299 millóns de documentos que conteñen "animal" pero que non conteñen "galago".

Podemos cambiar lixeiramente este algoritmo saltando cara adiante na lista de "animais". Cada vez que atopamos iso $d_a < d_g$, saltamos adiante k documentos da lista de "animais" a un novo documento, s_a. Se $s_a < d_g$, saltamos por outro k documentos. Facémolo ata $s_a \geq d_g$. Neste momento, reducimos a busca ata un rango de k documentos que poden conter d_g, que podemos buscar linealmente.

Canto tempo leva o algoritmo modificado? Dende a palabra "galago" aparece 1 millón de veces, sabemos que o algoritmo realizará un millón de buscas lineais de lonxitude k, dando un custo esperado de 500, 000 $\times k$ pasos. Tamén esperamos saltar cara adiante 300, 000, 000 / k veces. Este algoritmo entón leva aproximadamente 500, 000 $\times k +$ 300, 000, 000 / k pasos en total.

k	Pasos
5	62,5 millóns
10	35 millóns
20	25 millóns
25	24,5 millóns
40	27,5 millóns
50	31 millóns
100	53 millóns

Táboa 5.6. Saltar lonxitudes (k) e os pasos de procesamento esperados

A táboa 5.6 mostra o número de pasos de procesamento necesarios para algúns valores de exemplo de k. Obtemos o mellor rendemento esperado cando saltamos 25 documentos á vez. Teña en conta que a este valor de k, esperamos ter que saltar 12 veces na lista de "animais" por cada aparición de "galago". Isto é debido ao custo da busca lineal: un valor maior de k significa máis elementos para comprobar na busca lineal. Se no seu lugar escollemos unha busca binaria, o mellor valor de k sobe a uns 208, con preto de 9,2 millóns de pasos previstos.

Se a busca binaria combinada con saltar é moito máis eficiente, por que incluso considerar a busca lineal? O problema é a compresión. Para que a busca binaria funcione, necesitamos ser capaces de saltar directamente aos elementos da lista, pero despois da compresión, todos os elementos poderían ocupar unha cantidade diferente de espazo. Ademais, a codificación delta pode usarse nos números do documento, o que significa que aínda que puidésemos saltar a un lugar concreto na secuencia comprimida, teriamos que descomprimilo todo ata ese momento para poder descodificar os números do documento. Isto é desalentador porque o noso obxectivo é reducir a cantidade da lista que necesitamos procesar e parece que a compresión obríganos a descomprimir toda a lista.

Podemos resolver o problema de compresión cunha lista de saltadores. As listas Skippointer son pequenas estruturas de datos adicionais integradas no índice para permitirnos saltar as listas invertidas de xeito eficiente.

Un punteiro de salto (*d, p)* contén dúas partes, un número de documento *d* e unha posición de byte (ou bit) *páx.* Isto significa que hai unha publicación de lista invertida que comeza na posición *p,* e que a publicación inmediatamente antes é para documento *d.* Teña en conta que esta definición do punteiro de salto resolve os nosos problemas de compresión: podemos comezar a descodificar na posición *p,* e xa que sabemos iso *d* é o documento inmediatamente anterior *p,* podemos usalo para a descodificación.

Como exemplo sinxelo, considere a seguinte lista de números de documento sen comprimir:

5, 11, 17, 21, 26, 34, 36, 37, 45, 48, 51, 52, 57, 80, 89, 91, 94, 101, 104, 119

Se codificamos por delta esta lista, acabaremos cunha lista de ocos en d como este:

5, 6, 6, 4, 5, 9, 2, 1, 8, 3, 3, 1, 5, 23, 9, 2, 3, 7, 3, 15

A continuación, podemos engadir algúns punteiros para esta lista usando posicións baseadas en 0 (é dicir, o número 5 está na posición 0 da lista):

(17, 3), (34, 6), (45, 9), (52, 12), (89, 15), (101, 18)

Supoñamos que intentamos descodificar usando o punteiro de salto (34, 6). Pasamos á posición 6 na lista de ocos en d, que é o número 2. Engadimos 34 a 2, para decodificar o documento número 36.

Máis xeralmente, se queremos atopar o documento número 80 na lista, escanearemos a lista de punteiros de saltos ata atopar (52, 12) e (89, 15). 80 é maior que 52 pero menos de 89, polo que comezamos a descodificar na posición 12. Atopamos:

• 52 + 5 = 57
• 57 + 23 = 80

Neste momento, atopamos 80 na lista con éxito. Se en cambio estivésemos buscando 85, comezaríamos de novo no omitir o punteiro (52, 12):

• 52 + 5 = 57
• 57 + 23 = 80
• 80 + 9 = 89

Neste momento, desde 85 <89, saberiamos que 85 non figura na lista. Na análise dos saltos para o " galago E animal ”Exemplo, a eficacia dos punteiros de saltos dependía do feito de que“ animal ”era moito máis común que“ galago ”. Descubrimos que 25 era un bo valor para *k* dada esta consulta, pero só podemos escoller un valor para *k* para todas as consultas. A mellor forma de escoller *k* é atopar o mellor posible *k* para algunha mostra realista de consultas. Para a maioría das coleccións e cargas de consulta, a distancia de salto óptima rolda os 100 bytes.

5.5 Estruturas auxiliares

O ficheiro invertido é a estrutura de datos principal nun motor de busca, pero normalmente son necesarias outras estruturas para un sistema totalmente funcional.

Vocabulario e estatísticas

Un ficheiro invertido, como se describe neste capítulo, é só unha colección de listas invertidas. Para buscar no índice, é necesario algún tipo de estrutura de datos para atopar a lista invertida dun termo concreto. A forma máis sinxela de resolver este problema é almacenar cada lista invertida como un ficheiro separado, onde cada ficheiro leva o nome do termo de busca correspondente. Para atopar a lista invertida de "can", o sistema pode abrir o ficheiro chamado can e ler o contido. Non obstante, como vimos no capítulo 4, as coleccións de documentos poden ter miles de palabras únicas e a maioría destas palabras só aparecerán unha ou dúas veces na colección. Isto significa que un índice, se se almacena en ficheiros, consistiría en millóns de ficheiros, a maioría moi pequenos.

Desafortunadamente, os sistemas de ficheiros modernos non están optimizados para este tipo de almacenamento. Un sistema de ficheiros normalmente reserva uns poucos kilobytes de espazo para cada ficheiro, aínda que a maioría dos ficheiros conterán só uns poucos bytes de datos. O resultado é unha enorme cantidade de espazo desperdiciado. Como exemplo, na colección AP89 aparecen máis de 70.000 palabras

só unha vez (ver táboa 4.1). Estas listas invertidas requiriría uns 20 bytes cada unha, para un total de aproximadamente 2 MB de espazo. Non obstante, se o sistema de ficheiros require 1 KB para cada ficheiro, o resultado é 70 MB de espazo usado para almacenar 2 MB de datos. Ademais, moitos sistemas de ficheiros aínda almacenan información de directorios en matrices sen clasificar, o que significa que as buscas de ficheiros poden ser moi lentas para directorios de ficheiros grandes.

Para solucionar estes problemas, as listas invertidas adoitan almacenarse xuntas nun único ficheiro, o que explica o nome *ficheiro invertido.* Unha estrutura de directorio adicional, chamada *vocabulario* ou *léxico,* contén unha táboa de busca desde termos de índice ata o conxunto de bytes da lista invertida no ficheiro invertido.

En moitos casos, esta táboa de busca de vocabulario será o suficientemente pequena como para caber na memoria. Neste caso, os datos do vocabulario poden almacenarse de xeito razoable no disco e cargarse nunha táboa de hash ao iniciar o motor de busca. Se o motor de busca precisa manexar vocabularios máis grandes, debería usarse algún tipo de estrutura de datos baseada en árbores, como unha árbore B, para minimizar os accesos ao disco durante o proceso de busca.

Galago usa unha estratexia híbrida para a súa estrutura de vocabulario. Un pequeno ficheiro en cada índice, chamado vocabulario, almacena unha táboa de busca abreviada desde termos de vocabulario ata conxuntos de ff no ficheiro invertido. Este ficheiro contén só unha entrada de vocabulario por cada 32 K de datos no ficheiro invertido. Polo tanto, un ficheiro invertido de 32 TB requiriría menos de 1 GB de espazo de vocabulario, o que significa que sempre se pode gardar na memoria para colecciónss dun tamaño razoable. As listas do ficheiro invertido almacénanse en orde alfabética. Para atopar unha lista invertida, o motor de busca utiliza a busca binaria para atopar a entrada máis próxima da táboa de vocabulario e le o ff conxunto desa entrada. A continuación, o motor le 32 KB do ficheiro invertido, comezando polo conxunto o. Este enfoque atopa cada lista invertida cunha soa busca de disco.

Para calcular algunhas funcións de funcións, o índice debe conter certas estatísticas vocábulas, como o termo frecuencia ou frecuencia de documento (discutido no capítulo 4). Cando estas estatísticas pertencen a un termo específico, pódense gardar facilmente ao comezo da lista invertida. Algunhas destas estatísticas pertencen ao corpus, como o número total de documentos almacenados. Cando só hai algúns destes tipos de estatísticas, pódense ignorar con seguridade as consideracións sobre o almacenamento eficiente. Galago almacena estas estatísticas de toda a colección nun ficheiro XML chamado manifesto.

Documentos, fragmentos e sistemas externos

O motor de busca, como se describiu ata agora, devolve unha lista de números e puntuacións de documentos. Non obstante, un motor de busca real centrado no usuario precisa amosar información textual sobre cada documento, como o título do documento, o URL ou o resumo do texto.

(O capítulo 6 explica isto con máis detalle). Para obter este tipo de información, cómpre recuperar o texto do documento.

No capítulo 3 vimos algúns xeitos de almacenar documentos para un acceso rápido. Hai moitas formas de abordar este problema, pero ao final é necesario un sistema separado para converter os resultados dos motores de busca en números en algo lexible pola xente.

5.6 Construción do índice

Antes de que un índice poida usarse para o procesamento de consultas, ten que ser creado a partir da colección de texto. Construír un índice pequeno non é especialmente difícil, pero a medida que medran os tamaños de entrada, algúns trucos de construción de índices poden ser útiles. Nesta sección veremos primeiro a construción simple do índice na memoria e, a continuación, consideraremos o caso no que os datos de entrada non caben na memoria. Finalmente, consideraremos como construír índices usando máis dun ordenador.

5.6.1 Construción simple

O pseudocódigo para un simple indexador móstrase na figura 5.8. O proceso implica só algúns pasos. Unha lista de documentos pásase á función BuildIndex e a función analiza cada documento en fichas, como se comentou no capítulo 4. Estes símbolos son palabras, quizais con algún procesamento adicional, como descargas ou descargas. A función elimina fichas duplicadas, empregando, por exemplo, unha táboa hash. A continuación, para cada token, a función determina se hai que crear unha nova lista invertida *Eu,* e crea un se é necesario. Finalmente, o número de documento actual, *n,* engádese á lista invertida.

O resultado é unha táboa hash de fichas e listas invertidas. As listas invertidas son só listas de números de documentos enteiros e non conteñen información especial. Isto é suficiente para facer tipos de recuperación moi sinxelos, como vimos na sección 5.3.1.

Como se describe, este indexador pode usarse para moitas pequenas tarefas, por exemplo, deducir menos de algúns miles de documentos. Non obstante, está limitado de dous xeitos. En primeiro lugar, require que todas as listas invertidas se almacenen na memoria, o que pode non ser práctico para coleccións máis grandes. En segundo lugar, este algoritmo é secuencial, sen ningunha forma obvia de paralelizalo. A principal barreira para paralelizar este algoritmo é a táboa de hash, á que se accede constantemente no bucle interno. Engadir bloqueos á táboa de hash permitiría o paralelismo para analizar, pero só esa mellora

```
procedemento B •••• I •••• ( D)                    ◃ D é un conxunto de documentos de texto
    Eu ← HashTable ()                              ◃ Almacenamento da lista invertida
    n ← 0                                          ◃ Numeración de documentos
    para todos documentos d ∈ D facer
        n ← n + 1
        T ← Analizar ( d)                          ◃ Analice o documento en fichas
        Elimina os duplicados de T
        para todos fichas t ∈ T facer
            se Eu t ∉ Eu entón
                Eu t ← Array ()
            remate se
            Eu t. engadir ( n)
        final para
    final para
    regreso Eu
finalizar o procedemento
```

Fig. 5.8. Pseudocódigo para un simple indexador

Non é suficiente para usar máis dun puñado de núcleos de CPU. O manexo de grandes coleccións requirirá menos dependencia da memoria e un mellor paralelismo.

5.6.2 Fusión

O xeito clásico de resolver o problema da memoria no exemplo anterior é mediante *fusión.* Podemos construír a estrutura da lista invertida *Eu* ata que se esgote a memoria. Cando iso ocorre, escribimos o índice parcial *Eu* ao disco e logo comece a facer un novo. No ao final deste proceso, o disco está cheo de moitos índices parciais, Eu_1, Eu_2, Eu_3, ..., Eu_n. O sistema fusiona estes ficheiros nun único resultado.

Por definición, non é posible gardar nin sequera dous dos ficheiros de índice parcial á vez, polo que os ficheiros de entrada deben deseñarse coidadosamente para que se poidan combinar en pequenos anacos. Unha forma de facelo é almacenar os índices parciais en orde alfabética. É entón posible que un algoritmo de combinación combine os índices parciais empregando moi pouca memoria.

A figura 5.9 mostra un exemplo deste tipo de procedemento de fusión. A pesar de que esta cifra mostra só dous índices, é posible que se produzan moitos á vez. O algorithm é esencialmente o mesmo que o algoritmo estándar de clasificación por combinación. Xa que ambos Eu_1 e Eu_2 están ordenados, polo menos un deles apunta ao seguinte dato necesario para escribir *I.* Os datos dos dous ficheiros entrelázanse para producir un resultado ordenado.

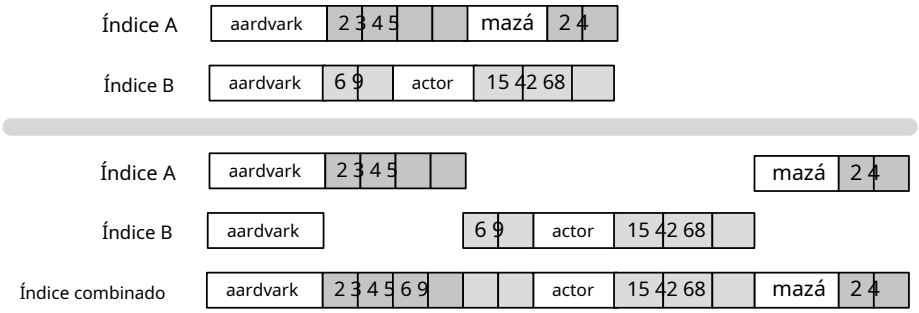

Fig. 5.9. Un exemplo de fusión de índices. O primeiro e o segundo índice combínanse para producir o índice combinado.

Desde Eu_1 e Eu_2 pode que usase os mesmos números de documento, a función de combinación renumera documentos en Eu_2.

Este proceso de fusión pode ter éxito aínda que só haxa memoria suficiente para almacenar dúas palabras (w_1 e w_2), unha única publicación de lista invertida e algúns apuntadores de ficheiros. En práctica, unha función de fusión real lería grandes anacos de Eu_1 e Eu_2, e logo escribe anacos grandes a Eu para empregar o disco de xeito máis eficiente.

Esta estratexia de fusión tamén mostra unha posible estratexia de indexación paralela. Se moitas máquinas constrúen os seus propios índices parciais, unha mesma máquina pode combinar todos eses índices nun único índice final. Non obstante, na seguinte sección exploraremos marcos de indexación distribuídos máis recentes que se están popularizando.

5.6.3 Paralelismo e distribución

O modelo tradicional para os motores de busca consistiu en empregar unha máquina rápida e única para crear o índice e procesar consultas. Esta segue sendo a opción axeitada para un gran número de aplicacións, pero xa non é unha boa opción para os sistemas máis grandes. Pola contra, para estes grandes sistemas, é cada vez máis popular usar moitos servidores inexpresivos xuntos e empregar software de procesamento distribuído para coordinar as súas actividades. MapReduce é unha ferramenta de procesamento distribuída que fai isto posible.

Dous factores forzaron este cambio. En primeiro lugar, a cantidade de datos a indexar nos sistemas máis grandes está a explotar. Os motores de busca Modernweb xa indexan decenas de millóns de páxinas, pero están chegando índices aínda máis grandes. Considere que se cada persoa do mundo escribise unha publicación de blog cada día, a web aumentaría en tamaño máis de dous billóns de páxinas cada ano. Optimistamente, un ordenador moderno típico pode manexar algúns centos de millóns de páxinas, aínda que non co tipo de tempos de resposta que

a maioría dos usuarios esperan. Isto deixa un enorme abismo entre o tamaño da web e o que podemos manexar coa tecnoloxía actual dun só ordenador. Teña en conta que este problema non está restrinxido a poucas empresas de busca na web; moitas empresas máis queren *analizar* o contido da web en lugar de facelo dispoñible para a busca pública. Estas empresas teñen o mesmo problema de escalabilidade.

O segundo factor é a economía simple. A incrible popularidade dos ordenadores persoais fíxolles moi poderosos e baratos. Pola contra, as grandes computadoras serven a un mercado moi pequeno e, polo tanto, teñen menos oportunidades para desenvolver economías de escala. Co paso do tempo, esta diferenza de escala fixo difícil fabricar un ordenador moito máis potente que un ordenador persoal que aínda se vende por unha cantidade razoable de diñeiro. Moitos grandes sistemas de recuperación de información funcionaban con mainframes no pasado, pero a plataforma de elección actual consiste en moitos servidores de mercadorías baratos.

Os servidores baratos teñen algunhas desvantaxes en comparación cos mainframes. En primeiro lugar, é máis probable que se rompan e a probabilidade de que polo menos un servidor falle a medida que engada máis servidores. En segundo lugar, son difíciles de programar. A maioría dos programadores están ben adestrados para a programación dun só fío, menos adestrados para a programación de fíos ou multiprocesos e non están ben adestrados para a programación de rede cooperativa. Desenvolvéronse moitos kits de ferramentas de programación para axudar a solucionar este tipo de problemas. RPC, CORBA, Java RMI e SOAP foron desenvolvidos para permitir chamadas de función a través dos límites da máquina. MPI proporciona unha abstracción diferente, chamada *pasando mensaxe,* que é popular en moitas tarefas científicas. Ningunha destas técnicas é especialmente robusta contra fallos do sistema e os modelos de programación poden ser complexos. En particular, estes sistemas non axudan a distribuír os datos de xeito uniforme entre as máquinas; ese é o traballo do programador.

Colocación de datos

Antes de mergullarse na mecánica do procesamento distribuído, considere os problemas de manexo de grandes cantidades de datos nun mesmo ordenador. O procesamento distribuído e o procesamento de datos a grande escala teñen un aspecto común en común, que é que non todos os datos de entrada están dispoñibles á vez. No procesamento distribuído, os datos poden estar espallados entre moitas máquinas. No procesamento de datos a gran escala, a maioría dos datos están no disco. En ambos casos, a clave para un procesamento eficiente de datos é colocar os datos correctamente.

Poñamos un exemplo sinxelo. Supoña que ten un ficheiro de texto que contén datos sobre transaccións con tarxeta de crédito. Cada liña do ficheiro contén un número de tarxeta de crédito

e unha cantidade de diñeiro. Como pode determinar o número de números de tarxeta de crédito únicos no ficheiro?

Se o ficheiro non é moi grande, podes ler cada liña, analizar o número da tarxeta de crédito e gardar o número da tarxeta de crédito nunha táboa de hash. Unha vez lido todo o ficheiro, a táboa de hash contería unha entrada por cada número único de tarxeta de crédito. Contar o número de entradas na táboa hash daríache a resposta. Afortunadamente, para un ficheiro grande, a táboa de hash sería demasiado grande para almacenala na memoria.

Agora supoña que tiña os mesmos datos da tarxeta de crédito, pero as transaccións do ficheiro ordenáronse por número de tarxeta de crédito. Contar o número de números únicos de tarxetas de crédito neste caso é moi sinxelo. Le cada liña do ficheiro e analízase o número da tarxeta de crédito da liña. Se o número de tarxeta de crédito atopado é diferente ao da liña anterior, increméntase un contador. Cando se chega ao final do ficheiro, o contador contén o número de tarxetas de crédito únicas no ficheiro. Non é necesaria ningunha táboa de hash para que isto funcione.

Agora volvemos á computación distribuída. Supoña que ten máis dun ordenador para usar para esta tarefa de reconto. Pode dividir o arquivo grande de transaccións en pequenos lotes de transaccións. Cada computador pode contar a súa fracción e entón os resultados pódense combinar para producir un resultado final.

Inicialmente, comezamos cun ficheiro de transaccións sen ordenar. Dividimos ese ficheiro en pequenos lotes de transaccións e contamos os números de tarxeta de crédito únicos de cada lote. Como combinamos os resultados? Poderiamos engadir o número de números de tarxeta de crédito atopados en cada lote, pero isto é incorrecto, xa que o mesmo número de tarxeta de crédito podería aparecer en máis dun lote e, polo tanto, contaríase máis dunha vez no total final. Pola contra, teriamos que manter unha lista dos números de tarxeta de crédito únicos que se atopan en cada lote e, a continuación, combinar esas listas para facer unha lista de resultados finais. O tamaño desta lista final é o número de números de tarxeta de crédito únicos no conxunto.

Pola contra, supoña que as transaccións se dividen en lotes con máis coidado, de xeito que todas as transaccións realizadas coa mesma tarxeta de crédito rematen no mesmo lote. Con esta restrición adicional, cada lote pódese contar individualmente e, a continuación, pódense engadir os contos de cada lote para obter un resultado final. É necesario nomear, porque non hai posibilidade de contar dobre. Cada número de tarxeta de crédito aparecerá precisamente nun lote.

Estes exemplos poden ser un pouco tediosos, pero a cuestión é que a adecuada agrupación de datos pode cambiar radicalmente as características de rendemento dunha tarefa. O uso dun ficheiro de entrada ordenado facilitou a tarefa de reconto, reduciu a cantidade de memoria necesaria a case cero e permitiu distribuír o cálculo facilmente.

MapReduce

MapReduce é un marco de programación distribuído que se centra na colocación e distribución de datos. Como vimos nos últimos exemplos, a colocación adecuada de datos pode facer que algúns problemas sexan moi sinxelos de computar. Ao centrarse na colocación de datos, MapReduce pode desbloquear o paralelismo nalgunhas tarefas comúns e facilitar o procesamento de grandes cantidades de datos.

MapReduce recibe o seu nome das dúas pezas de código que un usuario precisa para escribir para o marco: o *Mapeador* e o *Redutor.* A biblioteca MapReduce lanza automaticamente moitas tarefas Mapper e Reducer nun grupo de máquinas. A parte interesante sobre MapReduce, con todo, é o camiño que seguen os datos entre o Mapper e o Reducer.

Antes de ver como funcionan theMapper e Reducer, vexamos as bases da idea MapReduce. As funcións `mapa` e `reducir` atópanse normalmente en linguaxes funcionais. En termos moi sinxelos, o `mapa` A función transforma unha lista de elementos noutra lista de elementos da mesma lonxitude. O `reducir` A función transforma unha lista de elementos nun único elemento. O marco MapReduce non é tan estrito coas súas definicións: tanto os mapas como os redutores poden devolver un número arbitrario de elementos. Non obstante, a idea xeral é a mesma.

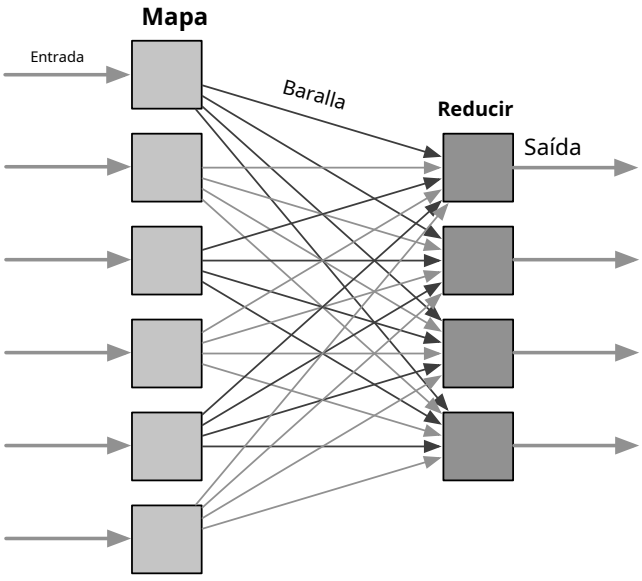

Fig. 5.10. MapReduce

Supoñemos que os datos aparecen nun conxunto de rexistros. Os rexistros envíanse ao Mapper, que transforma estes rexistros en pares, cada un cunha clave e un valor. O seguinte paso é o shu ffl e, que a biblioteca realiza por si só. Esta operación usa unha función de hash para que todos os pares coa mesma tecla acaben un ao lado do outro e na mesma máquina. O último paso é a fase de reducción, onde os rexistros se procesan de novo, pero esta vez por lotes, o que significa que todos os pares coa mesma clave se procesan á vez. Os pasos de MapReduce resúmense na figura 5.10.

```
procedemento M •• C ••••• C •••• (entrada)
    mentres non input.done () facer
        rexistro ←input.next ()
        tarxeta ←record.card
        cantidade ←rexistro.cantidade
        Emitir (tarxeta, cantidade)
    rematar mentres
finalizar o procedemento
```

Fig. 5.11. Mapeador para un algoritmo de suma de tarxetas de crédito

```
procedemento R ••••• C ••••• C •••• (clave, valores)
    total ←0
    tarxeta ←clave
    mentres not values.done () facer
        cantidade ←values.next ()
        total ←total + importe
    rematar mentres
    Emitir (tarxeta, total)
finalizar o procedemento
```

Fig. 5.12. Redutor dun algoritmo de suma de tarxetas de crédito

O exemplo de datos da tarxeta de crédito que vimos na sección anterior funciona ben como unha tarefa de MapReduce. No Mapper (Figura 5.11), cada rexistro divídese nunha clave (o número da tarxeta de crédito) e un valor (o importe do diñeiro na transacción). A fase shu sort e ordena os datos para que os rexistros co mesmo número de tarxeta de crédito rematen un ao lado do outro. A fase de reducción emite un récord por cada crédito único

número de tarxeta, polo que o número total de números de tarxeta de crédito únicos é o número de rexistros emitidos polo redutor (Figura 5.12).

Normalmente, supoñemos que o Mapper e o Reducer son *idempotente.* Por idempotente, queremos dicir que se ao Mapper ou Reducer se lle chama varias veces na mesma entrada, a saída será sempre a mesma. Esta idempotencia permite que a biblioteca MapReduce sexa tolerante a fallos. Se algunha parte do cómputo falla, quizais por un fallo da máquina de hardware, a biblioteca MapReduce só pode procesar esa parte da entrada de novo nunha máquina diferente. Mesmo cando as máquinas non fallan, ás veces as máquinas poden ser lentas por mor dunha configuración incorrecta ou por fallos lentos das pezas. Neste caso, unha máquina que parece normal podería producir resultados moito máis lentamente que outras máquinas dun clúster. Para evitar isto, a medida que o cómputo está a piques de rematar, a biblioteca MapReduce publica problemas *copia de seguridade* Mapas e redutores que duplican o procesamento feito nas máquinas máis lentas. Isto garante que as máquinas lentas non se convertan no pescozo dunha computación. A idempotencia do Mapper e Reducer son o que fan isto posible. Se o Mapper ou o Reducer modificaban directamente os ficheiros, por exemplo, non se poderían executar varias copias simultaneamente.

Vexamos o problema de indexar un corpus con MapReduce. No noso simple indexador almacenaremos listas invertidas con posicións de palabras.

```
procedemento M •• D •••••••• T • P ••••••• (entrada)
    mentres non input.done () facer
        documento ←input.next ()
        número ←número.documento
        posición ←0
        fichas ←Analizar (documento)
        para cada palabra w en fichas facer
            Emitir ( w, documento: posición)
            posición = posición + 1
        final para
    rematar mentres
finalizar o procedemento
```

Fig. 5.13. Mapeador de documentos

MapDocumentsToPostings (Figura 5.13) analiza cada documento na entrada. En cada posición de palabra, emite un par clave / valor: a clave é a palabra mesma e o valor é *documento: posición,* que é o número do documento e a posición

```
procedemento R ••••• P ••••••• T • L •••• (clave, valores)
    palabra ← clave
    WriteWord (palabra)
    mentres non input.done () facer
        EncodePosting (values.next ())
    rematar mentres
finalizar o procedemento
```

Fig. 5.14. Reductor para as publicacións de palabras

concatenados xuntos. Cando se chama ReducePostingsToLists (Figura 5.14), as publicacións emitidas cambiáronse para que todas as publicacións da mesma palabra estean xuntas. O Redutor chama a WriteWord para comezar a escribir unha lista invertida e despois usa EncodePosting para escribir cada publicación.

5.6.4 Actualización

Ata agora, asumimos que a indexación é un proceso por lotes. Isto significa que se entrega un conxunto de documentos ao indexador como entrada, o indexador constrúe o índice e, a continuación, o sistema permite aos usuarios executar consultas. Na práctica, as coleccióńs de documentos máis interesantes están en constante cambio. Como mínimo, as coleccióńs adoitan medrar co paso do tempo; cada día hai máis novas e máis correo electrónico. Noutros casos, como a busca na web ou a busca do sistema de ficheiros, o contido dos documentos tamén pode cambiar co paso do tempo. Un útil motor de busca ten que ser capaz de responder ás coleccióńs dinámicas.

Podemos resolver o problema da actualización con dúas técnicas: a combinación de índices e a combinación de resultados. Se o índice se almacena en memoria, hai moitas opcións para a actualización rápida do índice. Non obstante, aínda que o motor de busca estea a avaliar consultas en memoria, normalmente o índice almacénase nun disco. Inserir datos no medio dun ficheiro non é compatible con ningún sistema de ficheiros común, polo que a actualización directa baseada en disco non é sinxela. Non obstante, sabemos combinar índices entre si, como vimos na sección 5.6.2. Isto ofrécenos un enfoque sinxelo para engadir datos ao índice: make un novo índice máis pequeno ($Eu_{2)}$ e mestúreo co índice antigo ($Eu_{1)}$ facer un novo index que contén todos os datos (*Eu).* Publicacións en Eu_1 para calquera documento eliminado pódese ignorar durante a fase de combinación polo que non aparecen en *I.*

A combinación de índices é unha estratexia de actualización razoable cando as actualizacións de índices se producen en lotes grandes, quizais moitos miles de documentos á vez. Para actualizacións de documentos individuais, non é unha boa estratexia, xa que leva moito tempo escribir todo o índice no disco. Para estas pequenas actualizacións, é mellor crear un pequeno índice

para os novos datos, pero non inclúelos no índice máis grande. As consultas avalíanse separadamente do índice pequeno e do índice grande, e as listas de resultados xorden para atopar a parte superior k resultados.

A combinación de resultados resolve o problema de como xestionar novos documentos: só tes que colocalos nun novo índice. Pero, como eliminamos documentos do índice? A solución común é empregar unha lista de documentos eliminados. Durante o procesamento de consultas, o sistema comproba a lista de documentos eliminados para asegurarse de que ningún documento eliminado entra na lista de resultados mostrados ao usuario. Se cambia o contido dun documento, podemos eliminar a versión antiga do índice usando unha lista de documentos eliminada e despois engadir unha nova versión ao índice de documentos recentes.

A combinación de resultados permítenos considerar unha pequena estrutura de índice na memoria para manter novos documentos. Esta estrutura na memoria podería ser unha táboa hash de matrices, como se mostra na figura 5.8, e polo tanto sería sinxela e rápida de actualizar, incluso cun só documento.

Para obter aínda máis rendemento do sistema, en lugar de usar só dous índices (un índice en memoria e un índice baseado en disco), podemos usar moitos índices. Usar demasiados índices é unha mala idea, xa que cada novo índice ralentiza o procesamento de consultas. Non obstante, o uso de poucos índices produce un rendemento de compilación de índice lento debido a un exceso de trazo do disco. Unha solución especialmente elegante para este problema é *partición xeométrica.* No particionamento xeométrico, o índice máis pequeno, Eu_0, contén aproximadamente cantos datos caberían na memoria. O seguinte índice, Eu_1, contén aproximadamente r veces máis datos que Eu_1. Se m é a cantidade de bytes de memoria na máquina, índice Eu_n entón contén entre *Señor* $_n$ e (m + 1) r_n bytes de datos. Se índice Eu_n algunha vez contén máis de (m + 1) r_n, mestúrase en índice Eu_{n+1}. Se r = 2, o sistema pode aguantar 1000 m bytes de datos do índice usando só 10 índices.

5.7 Procesamento de consultas

Unha vez que se constrúe un índice, necesitamos procesar os datos nel para producir resultados de consulta. Aínda con algoritmos sinxelos, o procesamento de consultas mediante un índice é moito máis rápido do que é sen un. Non obstante, os algoritmos intelixentes poden aumentar a velocidade de procesamento de consultas entre dez e cen veces sobre as versións máis sinxelas. Exploraremos primeiro as dúas técnicas máis sinxelas de procesamento de consultas, chamadas documento por vez e termo por vez, e despois moverse con variantes máis rápidas e flexibles.

5.7.1 Avaliación de documentos á vez

A recuperación de documentos á vez é a forma máis sinxela, polo menos conceptual, de realizala cun ficheiro invertido. A figura 5.15 é unha imaxe de recuperación de documentos á vez para a consulta "auga salgada tropical". As listas invertidas móstranse horizontalmente, aínda que as publicacións aliñáronse de xeito que cada columna representa un documento diferente. As listas invertidas deste exemplo conteñen o reconto de palabras e a puntuación, para este exemplo, é só a suma das palabras de cada documento. As liñas grises verticais indican os diferentes pasos da recuperación. No primeiro paso, engádense todos os recontos do primeiro documento para producir a puntuación dese documento. Unha vez finalizada a puntuación do primeiro documento, anótase o segundo documento, despois o terceiro e despois o cuarto.

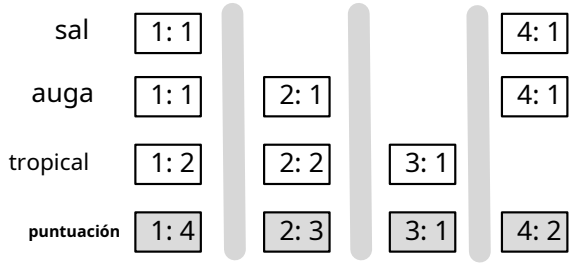

Fig 5.15. Avaliación da consulta de documentos á vez. Os números (*x: y)* representar un número de documento (*x)* e un reconto de palabras (*y).*

A figura 5.16 mostra unha implementación de pseudocódigos desta estratexia. Os parámetros son *Q,* a consulta; *Eu,* o índice; *f* e *g,* os conxuntos de funcións de funcións; e *k,* o número de documentos a recuperar. Este algoritmo puntúa documentos usando o modelo abstracto de clasificación descrito na sección 5.2. Non obstante, neste exemplo simplificado, supoñemos que o único valor de característica diferente de cero para *g (Q)* corresponden ás palabras da consulta. Isto dános unha correspondencia sinxela entre listas e funcións invertidas: hai unha lista para cada termo de consulta e unha característica para cada lista. Máis adiante neste capítulo exploraremos consultas estruturadas, que son un xeito estándar de ir máis alá deste simple modelo.

Por cada palabra w_{eu} na consulta, procúrase unha lista invertida do índice. Suponse que estas listas invertidas están ordenadas por número de documento. The Invert-

O obxecto edList comeza apuntando cara á primeira publicación de cada lista. Todas as listas invertidas recuperadas almacénanse nunha matriz, *L.*

```
procedemento D ••••••• A • AT ••• R •••••••• ( Q, I, f, g, k) L ← Array
    ()
    R ← PriorityQueue ( k)
    para todos termos w eu dentro P facer
        l eu ← InvertedList ( w eu, Eu)
        L. engadir ( l i)
    final para
    para todos documentos d ∈ Eu facer
        s d ← 0
        para todos listas invertidas l eu dentro L facer
            se l eu. getCurrentDocument () = d entón
                s d ← s d + g eu ( Q) f eu ( l i)        ◁ Actualiza a puntuación do documento
            rematar se
            l eu. movePastDocument ( d)
        final para
        R. engadir ( s d, d)
    final para
    regreso a parte superior k resultados de R
finalizar o procedemento
```

Fig. 5.16. Un sinxelo algoritmo de recuperación de documentos á vez

No bucle principal, a función bucle unha vez por cada documento da colección. En cada documento compróbanse todas as listas invertidas. Se aparece o documento nunha das listas invertidas, a función de función f_{eu} é avaliado e a documentación puntuación de mento s_D calcúlase sumando os valores da función ponderada. A continuación, o punteiro da lista invertida móvese ao punto na seguinte publicación. Ao final de cada doc- bucle de mento, calculouse unha nova puntuación do documento e engadiuse á cola de prioridade *R.*

Por claridade, este pseudocódigo está libre de cambios incluso sinxelos que melloren o rendemento. Non obstante, de xeito realista, a cola prioritaria *R* só precisa manter a parte superior *k* resultados en calquera momento. Se a cola prioritaria algunha vez contén máis de *k* resultados, os documentos con menor puntuación pódense eliminar ata só *k* permanecer, para aforrar memoria. Ademais, non é necesario circular por todos os documentos da colección; podemos cambiar o algoritmo para puntuar só documentos que aparecen en polo menos unha das listas invertidas.

O principal beneficio deste método é o seu frugal uso da memoria. O único uso importante da memoria procede da cola prioritaria, que só precisa almacenar *k* entradas

ao tempo. Non obstante, nunha implementación realista, grandes partes das listas invertidas tamén se gardarían na memoria durante a avaliación.

5.7.2 Avaliación de cada tempo

A figura 5.17 mostra a recuperación de termos á vez, usando a mesma consulta, función de puntuación e datos da lista invertida que no exemplo de documento á vez (Figura 5.15). Teña en conta que as puntuacións computadas son exactamente iguais en ambas as figuras, aínda que a estrutura de cada figura é diferente.

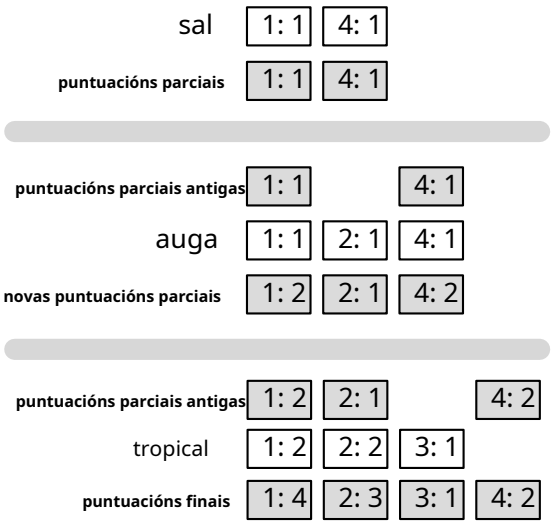

Fig. 5.17. Avaliación de consultas a un tempo

Como antes, as liñas grises indican os límites entre cada paso. No primeiro paso, a lista invertida de "sal" descodifícase e *puntuacións parciais* almacénanse en *acumuladores.* Estas puntuacións chámanse puntuacións parciais porque só son unha parte da puntuación final do documento. Os acumuladores, que reciben o seu nome do seu traballo, acumulan información de puntuación para cada documento. No segundo paso, as puntuacións parciais dos acumuladores combínanse cos datos da lista invertida para "auga" para producir un novo conxunto de puntuacións parciais. Despois de engadir os datos da lista para "tropical" no terceiro paso, o proceso de puntuación completouse.

A figura implica que se crea un novo conxunto de acumuladores para cada lista. Aínda que esta é unha posible técnica de implementación, na práctica os acumuladores almacénanse nunha táboa de hash. A información de cada documento actualízase como

procesáronse os datos da lista invertida. A táboa de hash contén as puntuacións finais do documento despois de que todas as listas invertidas fosen procesadas.

```
procedemento T ••• A • AT ••• R •••••••• ( Q, I, f, gk) A ← HashTable
    ()
    L ← Array ()
    R ← PriorityQueue ( k)
    para todos termos w eu dentro P facer
        l eu ← InvertedList ( w eu, Eu)
        L. engadir ( l i)
    final para
    para todos listas l eu ∈ L facer
        mentres l eu non está rematado facer
            d ← l eu. getCurrentDocument ()
            A d ← A d + g eu ( Q) f (l i)
            l eu. moveToNextDocument ()
        rematar mentres
    final para
    para todos acumuladores A d dentro A facer
        s d ← A d                    ◃ O acumulador contén a puntuación do documento
        R. engadir ( s d, d)
    final para
    regreso a parte superior k resultados de R
finalizar o procedemento
```

Fig. 5.18. Un sinxelo algoritmo de recuperación de termos á vez

O algoritmo de recuperación de termos por vez para o modelo de clasificación abstracta (Figura 5.18) é similar á versión de documento por vez ao comezo. Crea unha cola de prioridade e obtén unha lista invertida para cada termo da consulta, do mesmo xeito que o algoritmo de documento á vez. Non obstante, o seguinte paso é diferente. No canto dun bucle sobre cada documento do índice, o bucle exterior está sobre cada lista. O interior loop entón le cada publicación da lista, computando as funcións das funcións f_{eu} e g_{eu} e engadindo a súa contribución ponderada ao acumulador A_d. Despois de completar o bucle principal, os acumuladores escanéanse e engádense a unha cola de prioridade, que determina a parte superior k resultados que se devolverán.

A principal desvantaxe do algoritmo termo á vez é a utilización da memoria requirida pola táboa de acumuladores A. Lembre que o documento á vez

a estratexia require só a pequena cola de prioridade *R,* que ten un número limitado de resultados. Non obstante, o algoritmo termo por vez compensa isto debido ao seu acceso ao disco máis eficiente. Dado que le cada lista invertida de principio a fin, require unha busca mínima de discos e precisa moi pouca lista para acadar velocidades altas. Pola contra, o algoritmo de documento á vez cambia entre listas e require grandes listas para axudar a reducir o custo da busca.

Na práctica, nin os algoritmos de documento á vez nin os termos á vez non se usan sen optimizacións adicionais. Estas optimizacións melloran dramaticamente a velocidade de execución dos algoritmos e poden ter un gran efecto sobre a pegada da súa temática.

5.7.3 Técnicas de optimización

Hai dúas clases principais de optimizacións para o procesamento de consultas. O primeiro é ler menos datos do índice e o segundo é procesar menos documentos. Os dous están relacionados, xa que sería difícil puntuar o mesmo número de documentos ao ler menos datos. Cando se empregan funcións de funcións especialmente complexas, centrarse en puntuar menos documentos debería ser a principal preocupación. Para funcións de funcións simples, a mellor velocidade vén ignorando o maior número posible de datos da lista invertida.

Saltar lista

Na sección 5.4.7, cubrimos os apuntadores de saltos en listas invertidas. Este tipo de saltos cara adiante é de lonxe o xeito máis popular de ignorar porcións de listas invertidas (Figura 5.19). Tamén son posibles enfoques máis complexos (por exemplo, estruturas de árbores) pero non se usan con frecuencia.

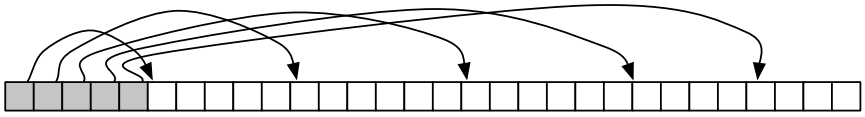

Fig. 5.19. Saltadores nunha lista invertida. As caixas grises amosan punteiros, que apuntan ás caixas brancas, que son publicacións de lista invertidas.

Os punteiros de salto non melloran o tempo de execución asintótico da lectura dunha lista inversa. Supoñamos que temos unha lista invertida que é n bytes de longo, pero engadimos punteiros despois de cada un c bytes e os punteiros son k bytes de longo. Lectura completa

a lista require lectura Θ (*n)* bytes, pero saltar pola lista usando os saltadores require Θ (*kn / c)* tempo, que equivale a Θ (*n).* Aínda que non hai ganancia asintótica no tempo de execución, o factor de *c* pode ser enorme. Para valores típicos de *c* = 100 e *k* = 4, saltar unha lista resulta en ler só o 2,5% do total de datos.

Teña en conta que como *c* medra, a cantidade de datos que precisa ler para saltar pola lista cae. Entón, por que non facer *c* o máis grande posible? O problema é que se *c* faise demasiado grande, o rendemento medio baixa. Vexamos este problema con máis detalle.

Supoña que quere atopar *páx* publicacións particulares nunha lista invertida, e a lista é *n* bytes de longo, con *k*- indicadores de saltos de bytes situados en *c*- intervalos de bytes. Polo tanto, hainos *n / c* intervalos totais na lista. Para atopalos *páx* publicacións, necesitamos ler *kn / c* bytes en punteiros de saltos, pero tamén necesitamos ler os datos *páx* intervalos. De media, supoñemos que as publicacións que queremos están aproximadamente a medio camiño entre dous apuntadores de saltos, polo que lemos un adicional *PC* / 2 bytes para atopar esas publicacións. O número total de bytes lidos é entón:

$$\frac{kn}{c} + \frac{PC}{2}$$

Teña en conta que esta análise asume iso *páx* é moito máis pequeno que *n / c;* iso é o que nos permite supor que cada publicación está no seu propio intervalo. Como *páx* achégase máis a *n / c,* é probable que algunhas das publicacións que queremos se atopen nos mesmos intervalos. Non obstante, fíxate nunha vez *páx* achégase a *n / c,* necesitamos ler case toda a lista invertida, polo que os indicadores para saltar non son moi útiles.

Volvendo á fórmula, podes ver isto cun valor maior de *c* fai que o primeiro termo sexa máis pequeno, tamén o fai máis grande. Polo tanto, escollendo o valor perfecto para *c* depende do valor de *p,* e non sabemos que *páx* é ata que se executa unha consulta. Non obstante, é posible usar consultas anteriores para simular o comportamento de saltar e obter unha boa estimación para *c.* Nos exercicios pediráselle que trazemos algúns gráficos desta fórmula e que resolvamos o punto de equilibrio.

Aínda que poida parecer que saltar lista podería aforrar en accesos ao disco, na práctica raramente o fai. Os discos modernos son moito mellores na lectura de datos secuenciais que no saltos a situacións aleatorias. Debido a isto, a maioría dos discos requiren un salto dunhas 100.000 publicacións antes de que se vexa calquera aceleración. Aínda así, saltar aínda é útil porque reduce a cantidade de tempo dedicado a descodificar datos comprimidos que se leu desde o disco e reduce drasticamente o tempo de procesamento das listas que se gardan na memoria caché.

Procesamento conxuntivo

O tipo máis sinxelo de optimización de consultas é *procesamento conxuntivo.* Por procesamento conxuntivo, só queremos dicir que todos os documentos devoltos ao usuario precisan conter todos os termos da consulta. O procesamento conxuntivo é o modo predeterminado para moitos motores de busca web, en parte debido á velocidade e en parte porque os usuarios o esperaron. Con consultas curtas, o procesamento conxuntivo pode realmente mellorar a eficacia e a eficacia simultaneamente. Pola contra, os motores de busca que utilizan consultas máis longas, como parágrafos enteiros, non serán bos candidatos para o procesamento conxuntivo.

O procesamento conxuntivo funciona mellor cando un dos termos da consulta é raro, como na consulta "locomoción dos peixes". A palabra "peixe" aparece unhas cen veces máis veces que a palabra "locomoción". Dado que só nos interesan os documentos que conteñen ambas palabras, o sistema pode saltar a maior parte da lista invertida de "peixe" para atopar só as publicacións nos documentos que tamén conteñan a palabra "locomoción".

O procesamento conxuntivo pódese empregar tanto cos sistemas termo-á-vez como con documentos-á-vez. Na figura 5.20 móstrase o algoritmo actualizado termo por vez para o procesamento conxuntivo. Ao procesar o primeiro termo, ($i = 0$), o procesamento procede normalmente. Non obstante, para os termos restantes, ($eu > 0$), o algoritmo procesa as publicacións comezando na liña ??. Comproba a táboa de acumuladores para a próxima documento que contén todos os termos de consulta anteriores e lista de instrucións l_{eu} para saltar a ese documento se hai unha publicación (liña ??). Se hai un publicando, o acumulador actualízase. Se a publicación non existe, elimínase o acumulador (liña ??).

A versión do documento á vez (Figura 5.21) é similar á versión antiga do documento á vez, excepto no bucle interno. Comeza atopando o maior documento d actualmente sinalado por unha lista invertida (liña 13). Este documento d Non está garantido que conteña todos os termos da consulta, pero é un candidato razoable. O seguinte bucle intenta saltar todas as listas cara adiante para apuntar a d (liña 16). Se isto non ten éxito, o bucle remata e outro documento d escóllese. Se ten éxito, o documento puntuarase e engadirase á cola de prioridade.

Ambos algoritmos, o sistema funciona máis rápido cando a primeira lista (l_0) é o máis curto e a última lista (l_n) é o máis longo. Isto resulta no maior número de distancias saltadas posibles na última lista, que é onde o saltar axudará máis.

```
1: procedemento T ••• A • AT ••• R •••••••• ( Q, I, f, g, k)
2:     A ← Mapa ()
3:     L ← Array ()
4:     R ← PriorityQueue ( k)
5:     para todos termos w eu dentro P facer
6:         l eu ← InvertedList ( w eu, Eu)
7:         L. engadir ( l i)
8:     final para
9:     para todos listas l eu ∈ L facer
10:        d 0 ← - 1
11:        mentres l eu non está rematado facer
12:            se i = 0 entón
13:                d ← l eu. getCurrentDocument ()
14:                A d ← A d + g eu ( Q) f (l i)
15:                l eu. moveToNextDocument ()
16:            senón
17:                d ← l eu. getCurrentDocument ()
18:                d ′ ← A. getNextAccumulator ( d)
19:                A. removeAccumulatorsBetween ( d 0, d ′)
20:                se d = d ′ entón
21:                    A d ← A d + g eu ( Q) f (l i)
22:                    l eu. moveToNextDocument ()
23:                senón
24:                    l eu skipForwardToDocument ( d ′)
25:                rematar se
26:                d 0 ← d ′
27:            rematar se
28:        rematar mentres
29:    final para
30:    para todos acumuladores A d dentro A facer
31:        s d ← A d                ◃ O acumulador contén a puntuación do documento
32:        R. engadir ( s d, d)
33:    final para
34:    regreso a parte superior k resultados de R
35: finalizar o procedemento
```

Fig. 5.20. Algoritmo de recuperación de termos á vez con procesamento conxuntivo

```
1: procedemento D ••••••• A • AT ••• R •••••••• ( Q, I, f, g, k)
2:     L ← Array ()
3:     R ← PriorityQueue ( k)
4:     para todos termos w eu dentro P facer
5:         I eu ← InvertedList ( w eu, Eu)
6:         L. engadir ( I i)
7:     final para
8:     d ← - 1
9:     mentres todas as listas en L non están rematados facer
10:        s d ← 0
11:        para todos listas invertidas I eu dentro L facer
12:            se I eu. getCurrentDocument ()> d entón
13:                d ← I eu. getCurrentDocument ()
14:            rematar se
15:        final para
16:        para todos listas invertidas I eu dentro L facer
17:            I eu. skipForwardToDocument ( d)
18:            se I eu. getCurrentDocument () = d entón
19:                s d ← s d + g eu ( Q) f eu ( I i)        ◃ Actualiza a puntuación do documento
20:                I eu. movePastDocument ( d)
21:            senón
22:                d ← - 1
23:                romper
24:            rematar se
25:        final para
26:        se d> - 1 entón R. engadir ( s d, d)
27:        rematar se
28:    rematar mentres
29:    regreso a parte superior k resultados de R
30: finalizar o procedemento
```

Fig. 5.21. Algoritmo de recuperación de documentos á vez con procesamento conxuntivo

Métodos limiares

Ata agora, os algoritmos que consideramos non fan moito co parámetro *k* ata a última afirmación. Lémbrate diso *k* é o número de resultados solicitados polo usuario e, en moitas aplicacións de busca, este número é algo pequeno, como 10 ou 20. Debido a este pequeno valor de *k,* a maioría dos documentos das listas invertidas

nunca se mostrará ao usuario. Os métodos limiares céntranse nisto k parámetro para marcar menos documentos.

En particular, teña en conta que, para cada consulta, hai algunha puntuación mínima que cada documento debe alcanzar antes de que se lle poida mostrar ao usuario. Esta puntuación mínima é a puntuación do k_{th} documento con maior puntuación. Nunca se lle mostrará ao usuario ningún documento que non teña puntuación polo menos. Nesta sección usaremos a letra grega tau (τ) para representar este valor, ao que chamamos *límite.*

Se puidésemos coñecer o valor axeitado para τ antes de procesar a consulta, serían posibles moitas optimizacións de consulta. Por exemplo, xa que un documento precisa unha puntuación de polo menos τ para ser útiles para o usuario, poderiamos evitar engadir documentos á cola de prioridade (no caso de documento por vez) que non acadaron unha puntuación de polo menos τ. En xeral, podemos ignorar con seguridade calquera documento cunha puntuación inferior a τ.

Desafortunadamente, non sabemos calcular o verdadeiro valor de τ sen avaliar a consulta, pero podemos aproximala. Estas aproximacións chamaranse τ : Queremos $\tau' \leq \tau$, de xeito que podemos ignorar con seguridade calquera documento cunha puntuación inferior a τ : Por suposto, canto máis se achega a nosa estimación τ' chega a τ, canto máis rápido se executará o noso algoritmo, xa que pode ignorar máis documentos.

Chegando cunha estimación para τ' é sinxelo cunha estratexia de documento á vez. Lémbrate diso R mantén unha lista dos primeiros k documentos con maior puntuación vistos ata o momento no proceso de avaliación. Podemos fixalo τ' á puntuación do documento con menor puntuación actualmente en R, asumindo R xa ten k documentos nela. Coa avaliación intermitente, non temos puntuacións completas do documento ata que a avaliación da consulta estea case rematada. Non obstante, aínda podemos configuralo τ' ser o k_{th} maior puntuación na táboa de acumuladores.

MaxScore

Con estimacións razoables para τ ; é posible comezar a ignorar algúns dos datos das listas invertidas. Esta estimación, τ ; representa un límite inferior na puntuación que necesita un documento para entrar na lista final clasificada. Polo tanto, cun pouco de matemáticas intelixentes, podemos ignorar partes das listas invertidas que non xerarán puntuacións documentais anteriores τ :

Vexamos máis detidamente como pode ocorrer isto cun exemplo sinxelo. Considere a consulta "eucalipto". A palabra "árbore" é aproximadamente 100 veces máis común que a palabra "eucalipto", polo que esperamos que a maior parte do tempo que pasamos avaliando esta consulta se dedique a anotar documentos que conteñan a palabra "árbore"

e non "eucalipto". Este é un mal uso do tempo, xa que estamos case seguros de atopar un conxunto de top k documentos que conteñen ambas palabras.

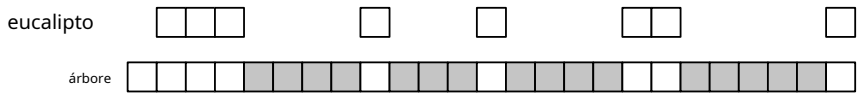

Fig. 5.22. Recuperación de MaxScore coa consulta "eucalipto". Os cadros grises indican publicacións que se poden ignorar con seguridade durante a puntuación.

A figura 5.22 mostra este efecto en acción. Vemos as listas invertidas de "eucalipto" e "árbore" que se estenden por toda a páxina, coas publicacións aliñadas por documentos, como nas figuras anteriores deste capítulo. Esta figura mostra que hai moitos documentos que conteñen a palabra “árbore” e non conteñen a palabra “eucalipto”. Supoñamos que o indexador calculou a maior puntuación parcial da lista de "árbores" e chámase ese valor $\mu_{\text{árbore}}$. Esta é a puntuación máxima (polo tanto, MaxScore) que podería ter calquera documento que conteña só esta palabra.

Supoñamos que só nos interesan os tres primeiros documentos da lista clasificada (é dicir, k é 3). O primeiro documento puntuado contén só a palabra "árbore". Os tres seguintes documentos conteñen "eucalipto" e "árbore". Usarémolo τ' para representar a puntuación máis baixa destes tres documentos. Neste momento, é moi probable que $\tau' > \mu_{\text{árbore}}$, porque τ' é a puntuación dun documento que contén os dous termos de consulta, mentres que $\mu_{\text{árbore}}$ é unha puntuación de consulta para un documento que contén só unha das consultas termos. Aquí é onde as caixas grises entran na historia. Unha vez $\tau' > \mu_{\text{árbore}}$, podemos saltar con seguridade todas as publicacións grises, xa que probamos que estes documentos non entrará na lista final clasificada.

Os datos da publicación na figura están fabricados, pero para as listas invertidas reais de "euca-" e "árbore", o 99% das publicacións de "árbore" serían caixas grises e, polo tanto, sería seguro ignoralas. Este tipo de saltos poden reducir drasticamente os tempos de consulta sen afectar á calidade dos resultados da consulta.

Terminación anticipada

O enfoque MaxScore garante que o resultado do procesamento de consultas será exactamente o mesmo na versión optimizada que sen optimización. Nalgúns casos, con todo, pode ser interesante correr algúns riscos con consultas de calidade e proceso dun xeito que poida levar a resultados diferentes que as mesmas consultas nun sistema non optimizado.

Por que podemos optar por facelo? Unha das razóns é que algunhas consultas son moito máis caras que outras. Considere a frase consulta "ser ou non ser". Esta consulta emprega termos moi comúns que terían listas invertidas moi longas. Executar esta consulta ata completar podería reducir gravemente a cantidade de recursos do sistema dispoñibles para atender outras consultas. Truncar o procesamento de consultas para esta custosa consulta pode axudar a garantir a equidade a outras persoas que usan o mesmo sistema.

Outra razón é que MaxScore é necesariamente conservador. Non saltará as rexións da lista invertida que poden ter un documento candidato utilizable. Debido a isto, MaxScore pode pasar moito tempo buscando un documento que poida que non exista. Asumir un risco calculado para ignorar estes improbables documentos pode supor unha diminución do consumo de recursos do sistema.

Como pode funcionar a rescisión anticipada? Nos sistemas de termo por vez, podemos finalizar o procesamento simplemente ignorando algúns dos termos de consulta moi frecuentes. Isto non é tan diferente de empregar unha lista de palabras clave, agás que neste caso ignoraríamos palabras que normalmente non se consideran palabras clave. Como alternativa, podemos decidir que despois de ler un número constante de publicacións, non se considerarán outros termos. O razoamento aquí é que, despois de procesar un número importante de publicacións, a clasificación debería estar bastante ben establecida. Ler máis información só cambiará un pouco a clasificación. Isto é especialmente certo para consultas con moitos (por exemplo, centos) de termos, que poden ocorrer cando se usan técnicas de ampliación de consultas.

Nos sistemas de documentos á vez, a terminación anticipada significa ignorar os documentos ao final das listas invertidas. Esta é unha mala idea se os documentos están clasificados en orde aleatoria, pero non ten por que ser así. Pola contra, os documentos poderían ser ordenados por unha métrica de calidade, como PageRank. Terminar cedo nese caso significaría ignorar os documentos que se consideran de calidade inferior aos documentos que xa foron puntuados.

Ordenación de listas

Ata agora, todos os exemplos deste capítulo supoñen que as listas invertidas se almacenan na mesma orde, por número de documento. Se os números do documento se asignan aleatoriamente, isto significa que a orde de clasificación do documento é aleatoria. O efecto neto é que os mellores documentos para unha consulta poden atoparse facilmente ao final das listas. Con bos documentos espallados pola lista, calquera algoritmo razoable de procesamento de consultas debe ler ou saltar toda a lista para asegurarse de que non se perdan bos documentos. Dado que estas listas poden ser longas, ten sentido considerar unha orde máis intelixente.

Unha forma de mellorar a ordenación de documentos é ordenar documentos en función da calidade do documento, como xa comentamos na última sección. Hai moitos métodos de calidade que se poderían empregar, como PageRank ou o número total de clics de usuario. Se os números de documentos máis pequenos se asignan aos documentos de maior calidade, resulta razoable considerar deter a busca cedo se se atoparon moitos bos documentos. Aquí pódense usar as técnicas limiares da sección MaxScore. Se sabemos que os documentos das listas están diminuíndo en calidade, podemos calcular un límite superior nas puntuacións dos documentos que quedan nas listas en cada punto durante a recuperación. Cando τ' se eleva por riba da puntuación de documento restante máis alta posible, a recuperación pódese deter con seguridade sen prexudicar a eficacia.

Outra opción é ordenar cada lista por puntuación parcial. Por exemplo, para a lista de "comida", primeiro poderiamos almacenar documentos que conteñan moitos exemplos da palabra "comida". Nunha aplicación web, isto pode corresponder a colocar páxinas de restaurantes no inicio da lista invertida. Para unha lista de "cans", primeiro poderiamos almacenar páxinas sobre cans (é dicir, que conteñan moitas instancias de "can"). Avaliar unha consulta sobre comida ou cans faise entón moi sinxelo. Non obstante, outras consultas poden ser máis difíciles. Por exemplo, como avaliamos a consulta "comida para cans"? A mellor forma de facelo é empregar unha táboa de acumuladores, como na recuperación de cada tempo. Non obstante, en vez de ler toda unha lista á vez, lemos só pequenos anacos de cada lista. Unha vez que a táboa de acumuladores demostre que se atoparon moitos bos documentos, podemos deixar de buscar. Como podes imaxinar, a recuperación funciona máis rápido con termos que probablemente aparecen xuntos, como "tortilla guacamole". Cando é probable que os termos non aparezan xuntos, por exemplo, "queixo de terra", é probable que tarde moito máis en atopar os mellores documentos.

5.7.4 Consultas estruturadas

Nos exemplos de avaliación de consultas que vimos ata agora, a nosa suposición é que cada lista invertida corresponde a unha única característica e que engadimos esas características para crear unha puntuación final do documento. Aínda que isto funciona en casos sinxelos, é posible que desexemos unha función de puntuación máis interesante. Por exemplo, na Figura 5.2 a consulta tiña moitas características interesantes, incluíndo unha frase ("peixes tropicais"), un sinónimo ("chíclidos") e algunhas características non tópicas (por exemplo, ligazóns entrantes).

Un xeito de facelo é escribir un código de clasificación especializado no sistema de recuperación que detecte estas funcións adicionais e use directamente os datos da lista invertida para calcular puntuacións, pero dun xeito máis complicado que unha combinación lineal de funcións. Esta visión aumenta moito o tipo de puntuación que podes usar e é moi eficiente. Desafortunadamente, non é moi flexible.

Outra opción é construír un sistema compatible *consultas estruturadas.* As consultas estruturadas son consultas escritas nun idioma de consulta, o que lle permite cambiar as funcións empregadas nunha consulta e a forma en que se combinan esas funcións. Os usuarios normais do sistema non usan a linguaxe de consulta. Pola contra, a *tradutor de consulta* converte a entrada do usuario nunha representación de consulta estruturada. Neste proceso de tradución é onde vai a intelixencia do sistema, incluído como ponderar as características das palabras e que sinónimos usar. Unha vez creada esta consulta estruturada, pásase ao sistema de recuperación para a súa execución.

Pode que xa estea familiarizado con este tipo de modelos, xa que os sistemas de bases de datos funcionan así. As bases de datos relacionais contrólanse empregando o Structured Query Language (SQL). Moitas aplicacións importantes consisten nunha interface de usuario e un xerador de consultas estruturado, co resto da lóxica controlada por unha base de datos. Esta separación da lóxica da aplicación da lóxica da base de datos permite que a base de datos sexa altamente optimizada e moi xeral.

Galago contén un sistema de procesamento de consultas estruturado que se describe detalladamente no capítulo 7. Esta linguaxe de consulta tamén se usa nos exercicios.

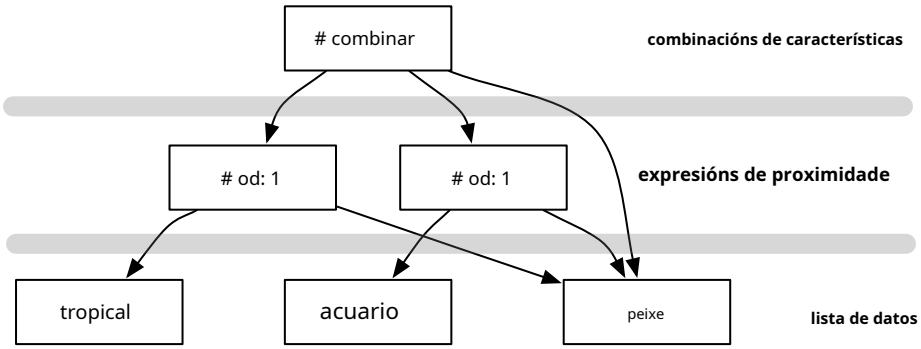

Fig. 5.23. Árbore de avaliación da consulta estruturada # combinar (#od: 1 (peixe tropical) # od: 1 (peixe de acuario) peixe)

A figura 5.23 mostra unha representación en árbore dunha consulta estruturada escrita no Linguaxe de consulta estructurada de Galago: # combine (#od: 1 (peixe tropical) #od: 1 (peixe de acuario) peixe).

Esta consulta indica que a puntuación do documento debe ser unha combinación das puntuacións de tres subconsultas. A primeira consulta é # od: 1 (peixe tropical). Na linguaxe de consulta de Galago, o # od: 1 operador significa que os termos dentro del deben aparecer uns ao lado dos outros, por esa orde, nun documento coincidente. O mesmo ocorre con

od: 1 (peixes de acuario). O termo de consulta final é peixe. Cada unha destas subconsultas actúa como unha característica de documento que se combina usando o # combinar operador.

Esta consulta contén exemplos dos principais tipos de expresións de consulta estruturadas. Na parte inferior da árbore temos termos índice. Son termos que se corresponden con listas invertidas no índice. Por riba dese nivel, temos expresións de proximidade. Estas expresións combinan listas invertidas para crear funcións máis complexas, como unha característica para "peixes" que aparece no título dun documento ou "peixe tropical" como frase. No nivel superior, os datos de funcións calculados a partir das listas invertidas combínanse nunha puntuación de documento. Neste nivel, ignórase a información de posición das listas invertidas.

Galago avalía as consultas estruturadas facendo unha árbore de obxectos iteradores que se parece á árbore mostrada na Figura 5.23. Por exemplo, créase un iterador que devolve os documentos coincidentes para # od: 1 (peixe tropical). O iterador atopa estes documentos coincidentes usando datos de iteradores de listas invertidas para "tropical" e "peixe". O # combinar O operador é un iterador de puntuacións de documentos que usa iteradores para # od: 1 (peixe tropical), #od: 1 (peixe de acuario), e peixe. Unha vez unha árbore de iteradores feitos como este, puntuar documentos só é unha cuestión de usar o iterador raíz para percorrer os documentos.

5.7.5 Avaliación distribuída

Unha única máquina moderna pode soportar unha carga sorprendente e probablemente sexa suficiente para a maioría das tarefas. Non obstante, para tratar cun corpus grande ou un gran número de usuarios pode ser necesario empregar máis dunha máquina.

O enfoque xeral para usar máis dunha máquina é enviar todas as consultas a *director* máquina. A continuación, o director envía mensaxes a moitos *servidores de índice,* que fan algunha parte do procesamento da consulta. A continuación, o director organiza os resultados deste proceso e devólveos ao usuario.

Chámase a estratexia de distribución máis sinxela *distribución de documentos.* Nesta estratexia, cada servidor de índice actúa como motor de busca dunha pequena fracción da colección total de documentos. O director envía unha copia da consulta a cada un dos servidores de índice, cada un dos cales devolve a parte superior *k* resultados, incluíndo as puntuacións do documento para estes resultados. Estes resultados son fusionados nunha única lista clasificada polo director, que logo devolve os resultados ao usuario.

Algúns algoritmos de clasificación dependen de estatísticas de colección, como o número de aparicións dun termo na colección ou o número de documentos que conteñen un termo. Estas estatísticas deben compartirse entre os servidores de índice para producir puntuacións comparables que se poidan combinar de xeito efectivo. En grupos moi grandes de

máquinas, o termo estatísticas a nivel de servidor de índice pode variar enormemente. Se cada servidor de index utiliza só as súas propias estatísticas de termos, o mesmo documento podería recibir tipos de puntuacións moi diferentes, dependendo do servidor de índice que se use.

Chámase outro método de distribución *distribución de termos.* Na distribución termal, constrúese un índice único para todo o clúster de máquinas. Cada lista invertida dese índice atribúese a un servidor de índice. Por exemplo, a palabra "can" pode ser tratada polo terceiro servidor, mentres que "gato" é tratada polo quinto servidor. Para un sistema con n servidores de índice e a k consulta de termos, é probable que todos os termos de consulta estean no mesmo servidor $1 / n^{k-1}$. Para un grupo de 10 máquinas, esta probabilidade é de só o 1% para unha consulta de tres termos. Polo tanto, na maioría dos casos os datos para procesar unha consulta non se almacenan todos nunha máquina.

Para procesar a consulta elíxese un dos servidores de índice, normalmente o que ten a lista máis longa invertida. Se outros servidores de índice teñen datos relevantes, estes datos envíanse a través da rede ao servidor de índice que procesa a consulta. Cando se completa o procesamento da consulta, os resultados envíanse a unha máquina director.

O enfoque de distribución de termos é máis complexo que a distribución de documentos debido á necesidade de enviar datos de lista invertida entre máquinas. Dado o tamaño das listas invertidas, as mensaxes implicadas no envío destes datos poden saturar unha rede. Ademais, cada consulta procésase empregando só un procesador en lugar de moitos, o que aumenta a latencia de consulta global fronte á distribución de documentos. A principal vantaxe da distribución de termos é buscar tempo. Se temos un k- consulta a termo e n servidores de índice, o número total de discos necesarios para procesar unha consulta é $O(kn)$ para un sistema distribuído por documentos, pero só $Ok)$ nun sistema distribuído por termos. Para un sistema ligado ao disco, e especialmente ao ligado á busca, a distribución de termos pode ser atractiva. Non obstante, investigacións recentes demostran que a distribución de termos raramente vale o esforzo.

5.7.6 Caché

Vimos no capítulo 4 como as frecuencias de palabras no texto seguen a distribución Zipfian: algunhas palabras aparecen con moita frecuencia, pero hai un número enorme de palabras con pouca frecuencia. Resulta que as distribucións de consultas son similares. Algunhas consultas, como as de celebridades populares ou eventos actuais, adoitan ser moi populares entre os buscadores públicos. Non obstante, aproximadamente a metade das consultas que cada día recibe un motor de busca son únicas.

Isto lévanos a unha discusión sobre *caché.* A grandes liñas, almacenar en caché significa almacenar algo que quizais queiras usar máis tarde. Cos motores de busca, normalmente

quere almacenar en caché listas de resultados clasificadas para consultas, pero os sistemas tamén poden almacenar en caché listas invertidas desde o disco.

A caché é perfectamente adecuada para os motores de busca. As consultas e as listas clasificadas son pequenas, o que significa que non se necesita moito espazo nunha caché para gardalas. Pola contra, procesar unha consulta contra un corpus grande pode ser moi computacional. Isto significa que unha vez que se calcula unha lista clasificada, normalmente ten sentido mantela.

Non obstante, a caché non resolve todos os nosos problemas de rendemento, porque aproximadamente a metade de todas as consultas recibidas cada día son únicas. Polo tanto, o motor de busca en si debe construírse para xestionar a consulta ffi c moi rapidamente. Isto leva á competencia por recursos entre o motor de busca e o sistema de caché. Investigacións recentes suxiren que cando o espazo de memoria é reducido, a caché debería centrarse nas consultas máis populares, deixando moito espazo para almacenar o índice en caché. As consultas únicas con varios termos aínda poden compartir un termo e usar a mesma lista invertida. Isto explica por que o almacenamento en caché de listas invertidas pode ter maiores taxas de éxito que o almacenamento en caché de consultas. Unha vez que se cachea todo o índice, todos os recursos restantes poden dirixirse cara á caché dos resultados das consultas.

Cando se usan sistemas de caché, é importante protexerse contra datos obsoletos. A caché funciona porque supoñemos que os resultados das consultas non cambiarán co paso do tempo, pero ao final si. As entradas na caché necesitan un tempo de espera aceptable que permita obter novos resultados. Isto é máis doado cando se trata de índices particionados como os que discutimos na sección 5.6.4. Cada caché pode asociarse cunha partición de índice particular e, cando se elimina esa partición, tamén se pode eliminar a caché. Teña presente que un sistema que está construído para xestionar un determinado rendemento máximo con caché habilitado xestionará un rendemento moito menor con caché o ff. Isto significa que se o seu sistema precisa destruír a súa caché, prepárese para ter un sistema lento ata que a caché se cubra adecuadamente. Se é posible, os arquivos de caché deberían ocorrer nos tempos máximos de carga.

Referencias e lecturas posteriores

Este capítulo contén información sobre moitos temas: indexación, procesamento de consultas, compresión, actualización de índices, caché e distribución só por citar algúns. Todos estes temas están nun capítulo para resaltar como estes compoñentes funcionan xuntos.

Debido á forma en que están interconectados estes compoñentes, é útil ver estudos de sistemas de traballo reais. Brin e Page (1998) escribiron un artigo sobre o sistema inicial de Google que é unha visión instructiva do que se necesita para construír un

sistema de traballo. Postos posteriores mostran como a arquitectura de Google cambiou o tempo, por exemplo Barroso et al. (2003). O documento MapReduce, de Dean e Ghemawat (2008), ofrece máis detalles do que fai este capítulo sobre como se desenvolveu MapReduce e como funciona na práctica.

O funcionamento interno dos motores de busca comerciais adoita considerarse segredo comercial, polo que non adoitan publicarse os detalles exactos de como funcionan. Unha excepción importante é o motor TodoBR, un popular motor de busca web brasileiro. Antes de que TodoBR fose adquirido por Google, os seus enxeñeiros publicaban con frecuencia artigos sobre o seu funcionamento. Un exemplo é o seu traballo sobre un esquema de caché de dous niveis (Saraiva et al., 2001), pero hai moitos outros.

O libro *Xestión de gigabytes (* Witten et al., 1999) é a referencia estándar para a construción de índices e é particularmente detallada no seu debate sobre as técnicas de compresión. O traballo sobre compresión para listas invertidas segue a ser unha área activa de investigación. Un dos aspectos máis recentes desta investigación é a serie PFOR de compresores de Zukowski et al. (2006), que aproveitan as características de rendemento dos procesadores modernos para facer un esquema especialmente rápido para descomprimir enteiros pequenos. Büttcher e Clarke (2007) fixeron un estudo recente sobre como se comparan os esquemas de compresión co hardware máis recente.

Zobel e Mo ff at (2006) escribiron un artigo de revisión que describe todas as importantes investigacións recentes en índices invertidos, tanto na construción de índices como no procesamento de consultas. Este artigo é o mellor lugar para buscar unha comprensión de como se axusta esta investigación.

Turtle and Flood (1995) desenvolveu a serie de algoritmos MaxScore. Fagin et al. (2003) adoptaron un enfoque similar con entradas ordenadas por puntuacións, aínda que inicialmente non aplicaron as súas ideas á recuperación de información. Anh e Mo ff at (2006) refinaron estas ideas para facer un sistema de recuperación particularmente eficiente.

Anh andMo ff at (2005) e Metzler et al. (2008) abarcan métodos para calcular puntuacións que se poden almacenar en listas invertidas. En particular, estes traballos describen como calcular puntuacións que son útiles na recuperación e que se poden almacenar de xeito compacto na lista. Strohman (2007) explora todo o proceso de creación de índices de puntuación e procesamento de consultas con eficacia con eles.

Moitos dos algoritmos deste capítulo baséanse na fusión de dúas entradas ordenadas; a construción do índice confía niso, como fai calquera tipo de proceso de recuperación de documentos á vez. Knuth escribiu un volume enteiro sobre só clasificación e busca, que inclúe grandes cantidades de fusión de material, incluída a fusión baseada en disco (Knuth, 1998). Se o libro de Knuth é demasiado desalentador, calquera manual de algoritmos estándar debería poder darlle máis detalles sobre como funciona.

Lester et al. (2005) desenvolveron o método de particionamento xeométrico para a actualización de índices. Büttcher et al. (2006) engadiron algunhas extensións a este modelo, centrándose en como se deben tratar termos moi comúns durante a actualización. Strohman e Croft (2006) mostran como actualizar o índice sen deter o procesamento de consultas.

Exercicios

5.1. A sección 5.2 introduciu un modelo abstracto de clasificación, onde os documentos e as consultas están representados por funcións. Cales son algunhas vantaxes de representar documentos e consultas por funcións? Cales son algunhas desvantaxes?

5.2. O noso modelo de clasificación contén unha función de clasificación $R(Q, D)$, que compara cada documento coa consulta e calcula unha puntuación. Estas puntuacións úsanse entón para determinar a lista clasificada final.

Un modelo de clasificación alternativo pode conter un tipo diferente de función de clasificación, $f(A, B, Q)$, onde A e B hai dous documentos diferentes na colección e P é a consulta. Cando A debería clasificarse máis alto que B, $f(A, B, Q)$ avalía a 1. Cando A debería clasificarse a continuación B, $f(A, B, Q)$ avalía a –1.

Se tes unha función de clasificación $R(Q, D)$, mostra como podes usalo nun sistema que require un formulario $f(A, B, Q)$. Por que non pode ir por outro camiño (use $f(A, B, Q)$ nun sistema que require $R(Q, D)$)?

5.3. Supoña que crea un motor de busca que usa cen ordenadores cun millón de documentos almacenados en cada un, para que poida buscar unha colección de 100 millóns de documentos. ¿Prefires unha función de clasificación como $R(Q, D)$ ou un como $f(A, B, Q)$ (do problema anterior). Por que?

5.4. Supoña que o seu motor de busca acaba de recuperar os 50 mellores documentos da súa colección en función das puntuacións dunha función de clasificación $R(Q, D)$. A súa interface de usuario pode amosar só 10 resultados, pero pode escoller calquera dos 50 mellores documentos para mostrar. Por que podes escoller amosarlle ao usuario algo diferente aos 10 primeiros documentos do conxunto de documentos recuperados?

5.5. Os documentos poden conter facilmente miles de funcións diferentes a cero. Por que é importante que as consultas teñan só algunhas características distintas de cero?

5.6. Non son necesarios índices para buscar documentos. O seu navegador web, por exemplo, ten unha función Buscar que busca texto sen usar un índice. Cando debería usar un índice invertido para buscar texto? Cales son algunhas vantaxes de usar un índice invertido? Cales son algunhas desvantaxes?

5.7. A sección 5.3 explica moitos xeitos diferentes de almacenar a información do documento en listas invertidas. Que tipo de listas invertidas poderías construír se necesitas un índice moi pequeno? Que tipo construirías se necesitas buscar nomes de cidades, como Kansas City ou São Paulo?

5.8. Escribe un programa que poida construír un índice invertido sinxelo dun conxunto de documentos de texto. Cada lista invertida conterá os nomes dos ficheiros dos documentos que conteñen esa palabra.

Supoñamos que o ficheiro A contén o texto "o raposo marrón rápido" e o ficheiro B contén "o raposo lento azul". A saída do seu programa sería:

```
% ./vostro-programa AB
azul B
marrón A
raposo AB
rápido A
lento B
o AB
```

5.9. Na sección 5.4.1, creamos un esquema de compresión inequívoco para números binarios de 2 bits. Atopa unha secuencia de números que ocupe máis espazo cando se "comprime" usando o noso esquema que cando está "sen comprimir".

5.10. Supoñamos que unha empresa desenvolve un novo esquema de compresión sen perda inequívoco para os números de 2 bits chamados *SuperShink.* Os seus desenvolvedores afirman que reducirá o tamaño de calquera secuencia de números de 2 bits polo menos 1 bit. Probe que os desenvolvedores están mentindo. Máis concretamente, demostre que:

- *SuperShink* nunca usa menos espazo que unha codificación sen comprimir, ou
- Hai unha entrada para *SuperShink* de tal xeito que a versión comprimida é maior que a entrada non comprimida

Pode supoñer que cada número de entrada de 2 bits está codificado por separado.

5.11. Por que necesitamos saber algo sobre o tipo de datos que comprimiremos antes de escoller un algoritmo de compresión? Céntrese especialmente no resultado do exercicio 5.10.

5.12. Desenvolver un codificador para as Elias- *y* código. Verifique que o seu programa produce os mesmos códigos que na táboa 5.2.

5.13. Identifique a distancia de salto óptima *k* ao realizar unha consulta booleana de dous termos onde un termo aparece 1 millón de veces e o outro termo aparece 100 millóns de veces. Supoña que se empregará unha busca lineal unha vez que se atope unha rexión adecuada para buscar.

5.14. Na sección 5.7.3, vimos que a distancia de salto óptima *c* pódese determinar minimizando a cantidade $kn/c + pc/2$, onde *k* é a lonxitude do punteiro de salto, *n* é o tamaño total da lista invertida, *c* é o intervalo de saltos e *páx* é o número de publicacións que se poden atopar.

Trace esta función usando $k = 4$, $n = 1.000.000$ e $p = 1.000$, pero variando *c*. Despois, traza a mesma función, pero configúrase $p = 10.000$. Observe como o valor óptimo para *c* cambios.

Finalmente, tome a derivada da función $kn/c + pc/2$ en termos de *c* para atopar o valor óptimo para *c* para un conxunto dado doutros parámetros (*k, n,* e *p).*

5.15. No capítulo 4, coñeceu a lei de Zipf e como aproximadamente o 50% das palabras atopadas nunha colección de documentos só se producirán unha vez. O seu traballo é deseñar un programa que verifique a lei de Zipf usando MapReduce.

O seu programa emitirá unha lista de pares de números, como este:

195840,1
70944,2
34039,3
. . .
1.333807

Esta mostra de mostra indica que 195.840 palabras apareceron unha vez na colección, 70.944 apareceron dúas veces e 34.039 apareceron tres veces, pero apareceu unha palabra 333.807 veces. O seu programa imprimirá este tipo de lista para unha colección de documentos. O seu programa empregará MapReduce dúas veces (dúas fases de mapa e dúas fases de reducción) para producir esta saída.

5.16. Escribe o programa descrito no exercicio 5.15 empregando o xogo de ferramentas de busca de Galago. Verifique que funciona indexando a colección Wikipedia fornecida no sitio web do libro.

7

Modelos de recuperación

"Non hai certeza, só hai oportunidade".

V, *V de Vendetta*

7.1 Visión xeral dos modelos de recuperación

Durante os últimos 45 anos de investigación de recuperación de información, un dos obxectivos principais foi comprender e formalizar os procesos que subxacen nunha persoa que toma a decisión de que un anaco de texto é relevante para a súa necesidade de información. Para desenvolver unha comprensión completa probablemente requiriría comprender como a linguaxe se representa e se procesa no cerebro humano, e estamos moi lonxe diso. Non obstante, podemos propoñer teorías sobre a relevancia nos modelos de recuperación matemáticos e probar esas teorías. comparándoos coas accións humanas. Os bos modelos deben producir resultados que se correlacionen ben coas decisións humanas sobre relevancia. Dito doutro xeito, os algoritmos de clasificación baseados en bos modelos de recuperación recuperarán os documentos relevantes preto da parte superior da clasificación (e, polo tanto, terán unha alta eficacia).

Que éxito tivo o modelado? Como exemplo, os algoritmos de clasificación para a procura xeral melloraron a eficacia en máis do 100% na década de 1990, medido empregando as coleccións de probas TREC. Estes cambios na eficacia corresponderon a melloras nos modelos de recuperación asociados. A eficacia na busca na web tamén mellorou substancialmente nos últimos 10 anos. En experimentos con coleccións web TREC, os algoritmos de clasificación máis eficientes proveñen de modelos de recuperación ben definidos. No caso dos buscadores comerciais na web, está menos claro cales son os modelos de recuperación, pero non hai dúbida de que os algoritmos de clasificación dependen de sólidos fundamentos matemáticos.

É posible desenvolver algoritmos de clasificación sen un modelo de recuperación explícito mediante probas e erros. Non obstante, o uso dun modelo de recuperación demostrou ser o mellor enfoque. Os modelos de recuperación, como todos os modelos matemáticos, proporcionan a

marco para definir novas tarefas e explicar suposicións. Cando se observan problemas cun algoritmo de clasificación, o modelo de recuperación proporciona unha estrutura para probar alternativas que serán moito máis eficientes que un *forza bruta (* probe todo) enfoque.

Nesta discusión non debemos pasar por alto o feito de que a relevancia é un concepto complexo. É bastante difícil que unha persoa explique por que un documento é máis relevante que outro e cando se lle pide ás persoas que xulguen a relevancia dos documentos para unha determinada consulta, moitas veces poden estar en desacordo. Os científicos da información escribiron volumes sobre a natureza da relevancia, pero non imos mergullarnos nese material aquí. Pola contra, discutimos dous aspectos clave de relevancia que son importantes tanto para os modelos de recuperación como para as medidas de avaliación.

O primeiro aspecto é a diferenza entre *tópico* e *usuario* relevancia, que se mencionou na sección 1.1. Adocument é tópicamente relevante para unha consulta se se xulga sobre o mesmo tema. Noutras palabras, a consulta e o documento son case o mesmo. Unha páxina web que conteña unha biografía de Abraham Lincoln seguramente sería tópicamente relevante para a consulta "AbrahamLincoln" e tamén sería tópicamente relevante para as consultas "presidentes dos Estados Unidos" e "Guerra Civil". A relevancia do usuario ten en conta todos os demais factores que entran no xuízo de relevancia dun usuario. Isto pode incluír a antigüidade do documento, o idioma do documento, o público obxectivo, a novidade do documento, etc. Un documento que contén só unha lista de todos os presidentes dos Estados Unidos, por exemplo, sería tópicamente relevante para a consulta "Abraham Lincoln", pero pode non considerarse relevante para a persoa que presentou a consulta porque buscaba máis detalles sobre a vida de Lincoln. Os modelos de recuperación non poden incorporar todos os factores adicionais implicados na relevancia do usuario, pero algúns si os teñen en conta.

O segundo aspecto de relevancia que consideramos é se o é *binario* ou *multivalorado*. A relevancia binaria significa simplemente que un documento é relevante ou non. Parece obvio que algúns documentos son menos relevantes que outros, pero aínda máis relevantes que os documentos que son completamente temáticos. Por exemplo, podemos considerar que o documento que contén unha lista de presidentes dos Estados Unidos é menos relevante tópicamente que a biografía de Lincoln, pero certamente máis relevante que un anuncio dun automóbil Lincoln. Baseado nesta observación, algúns modelos de recuperación e medidas de avaliación introducen explícitamente a relevancia como unha variable de valor múltiple. Múltiples niveis de relevancia son certamente importantes na avaliación, cando se solicita ás persoas que xulguen a relevancia. Demostrar que só hai tres niveis (relevante, non relevante, incerto) facilita moito a tarefa dos xuíces. Non obstante, no caso dos modelos de recuperación, as vantaxes de varios niveis son menos claras. Isto

é porque a maioría dos algoritmos de clasificación calculan a *probabilidade* de relevancia e pode representar a incerteza implicada.

Ao longo dos anos propuxéronse moitos modelos de recuperación. Dous dos máis antigos son o *Booleano* e *espazo vectorial* modelos. Aínda que estes modelos foron substituídos en gran parte por enfoques probabilísticos, adoitan mencionarse nas discusións sobre a recuperación de información e descríbense antes antes de entrar nos detalles doutros modelos.

7.1.1 Recuperación booleana

O modelo de recuperación booleano foi usado polos primeiros motores de busca e aínda está en uso na actualidade. Tamén se coñece como *recuperación da coincidencia exacta* xa que os documentos recupéranse se coinciden exactamente coa especificación de consulta e, se non, non se recuperan. Aínda que isto define unha forma de clasificación moi sinxela, a recuperación booleana non se describe xeralmente como un algoritmo de clasificación. Isto débese a que o modelo de recuperación booleano asume que todos os documentos do conxunto recuperado son equivalentes en termos de relevancia, ademais da suposición de que a relevancia é binaria. O nome booleano provén do feito de que só hai dous resultados posibles para a avaliación da consulta (VERDADEIRO e FALSO) e porque a consulta normalmente se especifica empregando operadores da lóxica booleana (AND, OR, NOT). Como se menciona no capítulo 6, os operadores de proximidade e os caracteres comodín tamén se usan habitualmente nas consultas booleanas. Buscando cunha utilidade de expresión regular como `grep` é outro exemplo de recuperación da coincidencia exacta.

A recuperación booleana presenta algunhas vantaxes. Os resultados do modelo son moi previsibles e fáciles de explicar aos usuarios. Os operandos dunha consulta booleana poden ser calquera característica de documento, non só palabras, polo que é sinxelo incorporar metadatos como a data dun documento ou o tipo de documento na especificación de consulta. Desde o punto de vista da implementación de Froman, a recuperación booleana adoita ser máis eficiente que a recuperación clasificada porque os documentos poden eliminarse rapidamente da consideración no proceso de puntuación.

A pesar destes aspectos positivos, o principal inconveniente deste enfoque de busca é que a eficacia depende totalmente do usuario. Debido á falta dun algoritmo de clasificación tan sofisticado, as consultas sinxelas non funcionarán ben. Recupararanse todos os documentos que conteñan as palabras de consulta especificadas e este conxunto recuperado presentarase ao usuario nalgunha orde, como por data de publicación, que pouco ten que ver con relevancia. É posible construír consultas booleanas complexas que restrinxan o conxunto recuperado a documentos relevantes na súa maioría, pero esta é unha tarefa difícil

iso require unha experiencia considerable. En resposta á dificultade de formular consultas, unha clase de usuarios coñecidos como intermediarios de busca (mencionados no último capítulo) asociouse aos sistemas de busca booleanos. A tarefa dun intermediario é traducir a necesidade de información dun usuario nunha consulta booleana complexa para un motor de busca en particular. Os intermediarios aínda se usan nalgunhas áreas especializadas, como en xurídicas. Non obstante, a simplicidade e a eficacia dos motores de busca modernos permitiu á maioría da xente facer as súas propias buscas.

Como exemplo de formulación de consultas booleanas, considere as seguintes consultas para un motor de busca que indexou unha colección de noticias. A simple consulta:

> Lincoln

recuperaría un gran número de documentos que mencionaban coches e lugares chamados Lincoln ademais de historias sobre o presidente Lincoln. Todos estes documentos serían equivalentes en termos de clasificación no modelo de recuperación booleano, independentemente de cantas veces se produza a palabra "lincoln" ou en que contexto ocorra. Ante isto, o usuario pode intentar restrinxir o alcance da busca coa seguinte consulta:

> presidente E lincoln

Esta consulta recuperará un conxunto de documentos que conteñen ambas palabras, que se producen en calquera parte do documento. Se hai unha serie de historias que inclúen a xestión dos coches Ford Motor Company e Lincoln, estas recupereranse no mesmo conxunto que as historias sobre o presidente Lincoln, por exemplo:

> Ford Motor Company anunciou hoxe que Darryl Hazel sucederá a Brian Kelley presidente de Lincoln Mercurio.

Se se recuperou o suficiente destes tipos de documentos, o usuario pode intentar eliminar documentos sobre coches usando o NON operador, do seguinte xeito:

> presidente AND lincoln AND NOT (automóbil OU coche)

Isto eliminaría calquera documento que conteña unha soa mención ás palabras "automóbil" ou "coche" en calquera parte do documento. O uso do NON o operador, en xeral, elimina moitos documentos relevantes xunto con documentos non relevantes e non se recomenda. Por exemplo, un dos mellores documentos na busca na rede de "Presidente Lincoln" foi unha biografía que contiña a frase:

> Lincoln o corpo sae de Washington nun nove coche tren funerario.

Usando NON (automóbil OU coche) na consulta eliminaríase este documento. Se o conxunto recuperado aínda é demasiado grande, o usuario pode intentar restrinxir a consulta engadindo palabras adicionais que deberían aparecer nas biografías:

> presidente E Lincoln E biografía E vida E lugar de nacemento E Gettysburg E NON (automóbil OU coche)

Desafortunadamente, nun motor de busca booleano, introducindo moitos termos de busca na consulta co E o operador a miúdo resulta que non se recupera nada. Para evitalo, o usuario pode intentar usar un OU no seu lugar:

> presidente AND lincoln AND (biografía OU vida ou lugar de nacemento OU Gettysburg) E NON (automóbil OU coche)

Recuperará calquera documento que conteña as palabras "presidente" e "lincoln", xunto con calquera das palabras "biografía", "vida", "lugar de nacemento" ou "gettysburg" (e non menciona "automóbil" nin "coche" ”).

Despois de todo isto, temos unha consulta que pode facer un traballo razoable ao recuperar un conxunto que contén algúns documentos relevantes, pero aínda non podemos especificar que palabras son máis importantes ou que ter máis palabras asociadas é mellor que ningunha delas. Por exemplo, un documento que contiña o seguinte texto foi recuperado no rango 500 por unha busca na web (que usa medidas de importancia das palabras):

> Presidente 's Day - Actividades de vacacións: artesanía, labirintos, buscas de palabras, ...
> A vida ofWashington ”Le todo o libro en liña! Abraham Lincoln Busca de novo no sitio ...

Un sistema de recuperación booleano non faría distinción entre este documento e os outros 499 que están mellor clasificados polo buscador web. Podería ser, por exemplo, o primeiro documento da lista de resultados.

Chamouse ao proceso de desenvolvemento de consultas con foco no tamaño do conxunto recuperado *buscando por números,* e é consecuencia das limitacións do modelo de recuperación booleano. Para abordar estas limitacións, os investigadores desenvolveron modelos, como o modelo do espazo vectorial, que incorporan a clasificación.

7.1.2 O modelo de espazo vectorial

O modelo de espazo vectorial foi a base para a maior parte das investigacións sobre recuperación de información nos anos sesenta e setenta, e os artigos que usan este modelo seguen a aparecer en conferencias. Ten a vantaxe de ser un marco sinxelo e intuitivamente atractivo para implementar comentarios de ponderación, clasificación e relevancia de termos.

Historicamente foi moi importante na introdución destes conceptos e as técnicas efectivas desenvolvéronse a través de anos de experimentación. Non obstante, como modelo de recuperación ten grandes defectos. Aínda que proporciona un conveniente marco computacional, ofrece poucas orientacións sobre os detalles de como os algoritmos de ponderación e clasificación están relacionados coa relevancia.

Neste modelo asúmense documentos e consultas como parte dun t- espazo vectorial dimensional, onde t é o número de termos índice (palabras, talos, frases, etc.). A documento D_{eu} está representado por un vector de termos índice:

$$D_{i} = (d_{eu1}, d_{eu2}, \ldots, d_{it}),$$

onde d_{ij} representa o peso do j th trimestre. Unha colección de documentos que contén n os documentos pódense representar como unha matriz de pesos termais, onde cada fila representa un documento e cada columna describe os pesos que se asignaron a un termo para un determinado documento:

$$\begin{array}{lccc} & Prazo_1 & Prazo_2 & \ldots Prazo_t \\ Doc_1 & d_{11} & d_{12} & \ldots d_{1t} \\ Doc_2 & d_{21} & d_{22} & \ldots d_{2t} \\ \vdots & \vdots & & \\ Doc_n & d_{n1} & d_{n2} & \ldots d_{nt} \end{array}$$

A figura 7.1 ofrece un exemplo sinxelo da representación vectorial para catro documentos. A matriz termo-documento xirouse de xeito que agora os termos son as filas e os documentos son as columnas. Os pesos termais son simplemente o reconto dos termos do documento. As palabras clave non están indexadas neste exemplo e as palabras foron detidas. Documento D_3, por exemplo, represéntase co vector (1, 1, 0, 2, 0, 1, 0, 1, 0, 0, 1).

As consultas represéntanse do mesmo xeito que os documentos. É dicir, unha consulta P está representado por un vector de t pesos:

$$Q = (q_1, q_2, \ldots, q_t),$$

onde q_j é o peso do j th termo na consulta. Se, por exemplo, a consulta era "peixes tropicais", entón usando a representación vectorial da figura 7.1, a consulta faríase ser (0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1). Un dos aspectos atractivos do modelo de espazo vectorial é o uso de diagramas sinxelos para visualizar os documentos e as consultas. Normalmente, móstranse como puntos ou vectores nunha imaxe tridimensional, como en

D 1 Peixe acuario de auga doce tropical. Peixes tropicais, coidado
D 2 do acuario, instalación de tanques. Conservación de peixes e
D 3 peixes vermellos tropicais en acuarios e cuncas.

D 4 Páxina de inicio de Tropical Tank: peixes e acuarios tropicais.

Condicións	Documentos			
	D 1	**D 2**	**D 3**	**D 4**
acuario	1	1	1	1
cunca	0	0	1	0
coidado	0	1	0	0
peixe	1	1	2	1
auga doce	1	0	0	0
peixe dourado	0	0	1	0
páxina de inicio	0	0	0	1
manter	0	0	1	0
montar	0	1	0	0
tanque	0	1	0	1
tropical	1	1	1	2

Fig. 7.1. Matriz de termos para unha colección de catro documentos

Figura 7.2. Aínda que isto pode ser útil para o ensino, é enganoso pensar que unha intuición desenvolvida usando tres dimensións pode aplicarse ao espazo documental de alta dimensión real. Lembre que o *t* Os termos representan todas as funcións do documento indexadas. Nas aplicacións empresariais e web, isto responde a centos de miles ou incluso *millóns* de dimensións.

Dada esta representación, os documentos poderían clasificarse calculando a distancia entre os puntos que representan os documentos e a consulta. Máis comúnmente, a *medida de semellanza* úsase (en lugar dunha distancia ou *desemellanza* medida), de xeito que os documentos con puntuacións máis altas sexan os máis similares á consulta. Propuxéronse e probáronse unha serie de medidas de semellanza para este propósito. O máis exitoso é o *correlación coseno* medida de semellanza. A correlación coseno mide o coseno do ángulo entre a consulta e os vectores do documento. Cando os vectores están *normalizado* de xeito que todos os documentos e consultas están representados por vectores de igual lonxitude, o coseno do ángulo entre dous vectores idénticos será 1 (o ángulo é cero) e para dous vectores que non comparten ningún termo diferente de cero, o o coseno será 0. A medida do coseno defínese como:

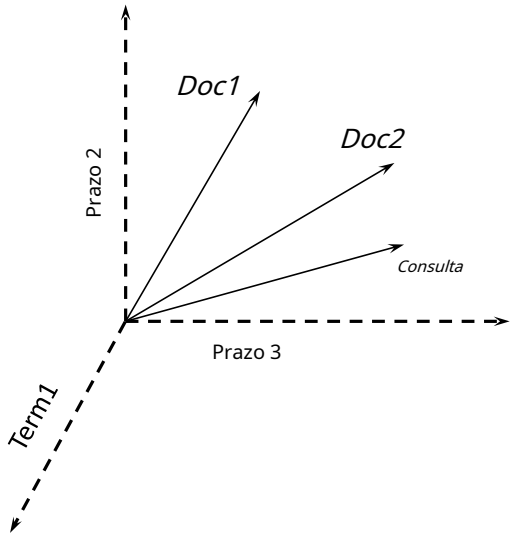

Fig. 7.2. Representación vectorial de documentos e consultas

$$Coseno\,(D_{eu},\,Q) = \frac{\sum_{j=1}^{t} d_{ij} \cdot q_j}{\sqrt{\sum_{j=1}^{t} d_{ij}^2 \cdot \sum_{j=1}^{t} q_j^2}}$$

O numerador desta medida é a suma dos produtos do termo ponderacións para a consulta coincidente e os termos do documento (coñecidos como *produto punto* ou produto interior). O denominador normaliza esta puntuación dividindo polo produto das lonxitudes dos dous vectores. Non hai ningunha razón teórica pola que se debería preferir a correlación coseno a outras medidas de semellanza, pero ten un mellor rendemento nas avaliacións da calidade da busca.

Como exemplo, considere dous documentos $D_1 = (0{,}5, 0{,}8, 0{,}3)$ e $D_2 = (0{,}9, 0{,}4, 0{,}2)$ indexado por tres termos, onde os números representan pesos termais. Dada a consulta $Q = (1{,}5, 1{,}0, 0)$ indexadas polos mesmos termos, as medidas do coseno para os dous documentos son:

$$Coseno\,(D_1, Q) = \sqrt{\frac{(0,5 \times 1,5) + (0,8 \times 1.0)}{(0,5^2 + 0,8^2 + 0,3^2)(1,5^2 + 1.0^2)}} = \sqrt{\frac{1,55}{(0,98 \times 3,25)}} = 0,87$$

$$Coseno\,(D_2, Q) = \sqrt{\frac{(0,9 \times 1,5) + (0,4 \times 1.0)}{(0,9^2 + 0,4^2 + 0,2^2)(1,5^2 + 1.0^2)}} = \sqrt{\frac{1,75}{(1,01 \times 3,25)}} = 0,97$$

O segundo documento ten unha puntuación máis alta porque ten un peso elevado para o primeiro trimestre, que tamén ten un peso elevado na consulta. Incluso este sinxelo exemplo mostra que a clasificación baseada no modelo de espazo vectorial é capaz de reflectir a importancia do termo e o número de termos coincidentes, o que non é posible na recuperación booleana.

Nesta discusión, aínda non dixemos nada sobre a forma do termo ponderación empregado no modelo espacial vectorial. De feito, ao longo dos anos probáronse moitos esquemas de ponderación. A maioría destas son variacións en *tf.idf* ponderación, que se describiu brevemente no capítulo 2. O termo compoñente de frecuencia, *tf,* re-
reflicte a importancia dun termo nun documento $D_{eu\,(}$ ou consulta). Normalmente componse como un reconto normalizado do termo ocorrencias nun documento, por exemplo
por

$$tf_{ik} = \sum_{j=1}^{t} \frac{f_{ik}}{f_{ij}}$$

onde tf_{ik} é o termo peso en frecuencia do termo k no documento $D_{eu,}$ e f_{ik} é o número de aparicións de termo k no documento. No modelo de espazo vectorial,
a normalización forma parte da medida do coseno. Unha colección de documentos pode conter documentos de moitas lonxitudes diferentes. Aínda que a normalización explica isto nalgún grao, os documentos longos poden ter moitos termos unha vez e outros centos de veces. Os experimentos de recuperación demostraron que, para reducir o impacto destes termos frecuentes, é efectivo empregar o logaritmo do número de aparicións de termos en *tf* pesos en lugar do reconto bruto.

O compoñente de frecuencia do documento inverso (*idf)* reflicte a importancia do termo na colección de documentos. Cantos máis documentos teña un termo, menos *discriminante* o termo está entre documentos e, en consecuencia, menos útil será na recuperación. A forma típica deste peso é

$$idf_k = \text{rexistro}\ \frac{N}{n_k}$$

onde idf_k é o peso inverso da frecuencia do documento para o termo k, N é o número de documentos da colección e n_k é o número de documentos en que termo k ocorre. A forma deste peso desenvolveuse por intuición e experimento, aínda que se pode argumentar que *idf* mide a cantidade de información que leva o termo, tal e como se define en *teoría da información (* Robertson, 2004).

Os efectos destes dous pesos combínanse multiplicándoos (de aí o nome *tf.idf).* A razón para combinalos deste xeito é, unha vez máis, sobre todo empírica. Tendo en conta isto, a forma típica de ponderación de termos de documentos no modelo de espazo vectorial é:

$$d_{ik} = \sqrt{\sum_{k=1}^{t} \frac{(\text{rexistro}(f_{ik}) + 1) \cdot \text{rexistro}(N / n_k)}{[(\text{rexistro}(f_{ik}) + 1.0) \cdot \text{rexistro}(N / n_k)]^2}}$$

A forma de ponderación de termos de consulta é esencialmente a mesma. Engadindo 1 ao compoñente de frecuencia térmica garante que os termos con frecuencia 1 teñan un peso diferente de cero. Teña en conta que, neste modelo, os pesos térmicos só se calculan para os termos que aparecen no documento (ou consulta). Dado que a normalización da medida do coseno está incorporada aos pesos, a puntuación dun documento calcúlase empregando simplemente o produto punto do documento e os vectores de consulta.

Aínda que non hai unha definición explícita de relevancia no modelo espacial vectorial, hai unha suposición implícita de que a relevancia está relacionada coa semellanza dos vectores de consulta e documento. Noutras palabras, é máis probable que os documentos "máis próximos" á consulta sexan relevantes. Este é principalmente un modelo de relevancia tópica, aínda que as características relacionadas coa relevancia do usuario poderían incorporarse á representación de vectores. Non se supón se a relevancia é binaria ou multivalora.

No último capítulo describimos os comentarios de relevancia, unha técnica para modificar a consulta baseada en documentos relevantes identificados polo usuario. Esta técnica introduciuse por primeira vez usando o modelo de espazo vectorial. O coñecido *Algoritmo de Rocchio* (Rocchio, 1971) baseouse no concepto de *consulta óptima,* que maximiza a diferenza entre o vector medio que representa os documentos relevantes e o vector medio que representa os documentos non relevantes. Dado que só hai información de relevancia limitada normalmente dispoñible, a forma máis común (e efectiva) do algoritmo de Rocchio modifica os pesos iniciais no vector de consulta P para producir unha nova consulta P' dacordo con

$$q_j' = \alpha . q_j + \beta . \frac{1}{|Rel|} \sum_{D_{eu} \in Rel} d_{ij} - \gamma . \frac{1}{|Nonrel|} \sum_{D_{eu} \in Nonrel} d_{ij}$$

onde q_j é o peso inicial do termo de consulta *j, Rel* é o conxunto de documentos relevantes identificados, *Nonrel* é o conxunto de documentos non relevantes, |. | dá o tamaño dun conxunto, d_{ij} é o peso do *j* th termo no documento *eu,* e *α, β,* e *γ* son parámetros que controlan o efecto de cada compoñente. Estudos previos demostraron iso o conxunto de documentos non relevantes aproxímase mellor por todos os documentos non vistos (é dicir, todos os documentos non identificados como relevantes) e que os valores razoables para os parámetros son 8, 16 e 4 para *α, β,* e *γ,* respectivamente.

Esta fórmula modifica os pesos do termo da consulta engadindo un compoñente baseado no peso medio dos documentos relevantes e restando un compoñente en función do peso medio dos documentos non relevantes. Elimínanse os termos de consulta con pesos negativos. Isto resulta nunha consulta máis longa ou ampliada porque se engadirán termos que aparecen con frecuencia nos documentos relevantes pero non na consulta orixinal (é dicir, terán pesos positivos diferentes a cero na consulta temática). Para restrinxir a cantidade de expansión, normalmente só se engadirá á consulta un certo número (por exemplo, 50) dos termos con maior peso medio dos documentos relevantes.

7.2 Modelos probabilísticos

Unha das características que debería proporcionar un modelo de recuperación é unha afirmación clara sobre os supostos sobre os que se basea. Os enfoques booleanos e do espazo vectorial fan suposicións implícitas sobre a relevancia e a representación de texto que afectan o deseño e a eficacia dos algoritmos de clasificación. A situación ideal sería mostrar que, dadas as suposicións, un algoritmo de clasificación baseado no modelo de recuperación logrará unha eficacia mellor que calquera outro enfoque. Esas probas son realmente difíciles de atopar na recuperación de información, xa que estamos intentando formalizar unha actividade humana complexa. A validez dun modelo de recuperación xeralmente ten que ser validada empíricamente, en vez de teoricamente.

Unha das primeiras afirmacións teóricas sobre a eficacia, coñecida como *Principio de clasificación de probabilidade (* Robertson, 1977/1997), fomentou o desenvolvemento de modelos de recuperación probabilísticos, que son o paradigma dominante na actualidade. Estes modelos acadaron este status porque *teoría da probabilidade* é unha forte base para representar e manipular a incerteza que é unha parte inherente

do proceso de recuperación de información. O principio de clasificación de probabilidade, como se dixo orixinalmente, é o seguinte:

> Se un sistema de recuperación de referencia [1] a resposta a cada solicitude é unha clasificación dos documentos da colección por orde de probabilidade decrecente de relevancia para o usuario que presentou a solicitude, onde as probabilidades se estiman coa maior precisión posible en función dos datos dispoñibles para para este propósito, a eficacia xeral do sistema para o seu usuario será a mellor que se poida obter a partir deses datos.

Dados algúns supostos, como que a relevancia dun documento para unha consulta é independente doutros documentos, é posible demostrar que esta afirmación é certa, no sentido de que a clasificación por probabilidade de relevancia maximizará a precisión, que é a proporción de documentos relevantes, en calquera rango dado (por exemplo, entre os 10 primeiros documentos). Desafortunadamente, o principio de clasificación de probabilidade non nolo di *como* para calcular ou estimar a probabilidade de relevancia. Hai moitos modelos de recuperación probabilística e cada un propón un método diferente para estimar esta probabilidade. A maior parte do resto deste capítulo discute algúns dos modelos probabilísticos máis importantes.

Nesta sección comezamos cun modelo probabilístico sinxelo baseado no tratamento da recuperación de información como un problema de clasificación. Despois describimos un algoritmo de clasificación popular e efectivo baseado neste modelo.

7.2.1 Recuperación de información como clasificación

En calquera modelo de recuperación que asuma a relevancia é binario, haberá dous conxuntos de documentos, os documentos relevantes e os documentos non relevantes, para cada consulta. Dado un novo documento, a tarefa dun motor de busca podería describirse como decidir se o documento pertence ao conxunto relevante ou ao non relevante [2]

conxunto. É dicir, o sistema debería *clasificar* o documento como relevante ou non relevante e recuperalo se é relevante.

Dado algún xeito de calcular o *probabilidade* que o documento é relevante e a probabilidade de que non sexa relevante, entón parecería razoable clasificalo no conxunto que ten a maior probabilidade. Noutras palabras,

[1] Un "sistema de recuperación de referencia" agora chamaríase motor de busca.

[2] Nótese que nunca se fala de documentos "irrelevantes" na recuperación de información; en cambio, son "non relevantes".

decidiriamos que un documento *D* é relevante se *P (R | D)> P (NR | D),* onde *P (R | D)* é un *condicional* probabilidade que representa a probabilidade de relevancia dada a representación dese documento e *P (NR | D)* é a probabilidade condicional de non relevancia (Figura 7.3). Isto coñécese como *Regra de decisión de Bayes,* e un sistema que clasifica os documentos deste xeito chámase a *Clasificador de Bayes.*

No capítulo 9, discutimos outras aplicacións de clasificación (como o filtrado de spam) e outras técnicas de clasificación, pero aquí centrámonos no algoritmo de clasificación que resulta deste modelo de recuperación probabilística baseado na clasificación.

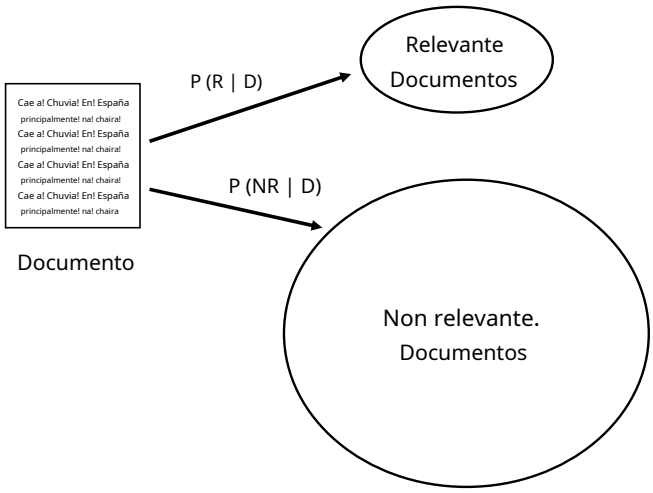

Fig. 7.3. Clasificar un documento como relevante ou non relevante

A pregunta que nos enfronta agora é como calcular estas probabilidades. Para comezar, centrémonos en *P (R | D).* Non está claro como imos calcular isto, pero dada a información sobre o conxunto relevante, deberiamos ser capaces de calcular *P (D | R).* Por exemplo, se tivésemos información sobre a frecuencia coa que ocorreron palabras específicas no conxunto relevante, entón, dado un novo documento, sería relativamente sinxelo calcular a probabilidade de ver a combinación de palabras no documento no caso correspondente. conxunto. Supoñamos que a probabilidade da palabra “presidente” no conxunto correspondente é 0,02 e a probabilidade de “lincoln” é 0,03. Se un novo documento contén as palabras "presidente" e "lincoln", poderiamos dicir que a probabilidade de observar esa combinación de palabras no

conxunto relevante é 0,02 × 0,03 = 0,0006, supoñendo que as dúas palabras se producen de xeito independente. [3]

Entón, como fai o cálculo $P(D \mid R)$ chéganos á probabilidade de relevancia? Acontece que hai unha relación entre $P(R \mid D)$ e $P(D \mid R)$ iso exprésase por *Regra de Bayes:* [4]

$$P(R \mid D) = \frac{P(D \mid R)\,P(R)}{P(D)}$$

onde $P(R)$ é o *a priori* probabilidade de relevancia (noutras palabras, a probabilidade de que calquera documento sexa relevante) e $P(D)$ actúa como unha constante normalizadora. Ante isto, podemos expresar a nosa regra de decisión do seguinte xeito: clasificar un documento como relevante se $P(D \mid R)\,P(R) > P(D \mid NR)\,P(NR)$. Isto é o mesmo que clasificar un documento como relevante se:

$$\frac{P(D \mid R)}{P(D \mid NR)} > \frac{P(NR)}{P(R)}$$

O lado esquerdo desta ecuación coñécese como *relación de verosimilitude.* Na maioría das aplicacións de clasificación, como o filtrado de spam, o sistema debe decidir a que clase pertence o documento para tomar as medidas adecuadas. Para a recuperación de información, un motor de busca só precisa clasificar os documentos en lugar de tomar esa decisión (que é difícil). Se usamos a razón de verosimilitude como puntuación, os documentos altamente clasificados serán os que teñen unha alta probabilidade de pertencer ao conxunto relevante.

Para calcular as puntuacións do documento, aínda debemos decidir como atopar valores para $P(D \mid R)$ e $P(D \mid NR)$. O enfoque máis sinxelo é facer os mesmos supostos que fixemos no noso exemplo anterior; é dicir, representamos os documentos como unha combinación de palabras e os conxuntos relevantes e non relevantes empregando probabilidades de palabras. Neste modelo, os documentos represéntanse como un vector de binario características, $D = (d_1, d_2, \ldots, d_t)$, onde $d_i = 1$ se termo *eu* está presente no documento e 0 se non. A outra suposición importante que facemos é *termo independencia (* tamén coñecido como o *Naïve Bayes* suposición). Isto significa que podemos estimar $P(D \mid R)$ polo produto das probabilidades individuais de termo $P(D \mid NR)$). Porque este modelo supón $\prod_{i=1}^{t}$ para os termos de independencia, podemos dicir para os e características binarias nos documentos, coñécese como *modelo de independencia binario.*

[3] Dados dous eventos A e B, a probabilidade conxunta $P(A \cap B)$ é a probabilidade de que ambos eventos ocorran xuntos. En xeral, $P(A \cap B) = P(A \mid B)\,P(B)$. Se A e B son independentes, isto significa que $P(A \cap B) = P(A)\,P(B)$.

[4] Chamado así por Thomas Bayes, un matemático británico.

Obviamente, as palabras non aparecen de xeito independente no texto. Se a palabra "Microsoft" aparece nun documento, é moi probable que tamén apareza a palabra "Windows". Non obstante, a asunción do termo independencia é habitual xa que normalmente simplifica as matemáticas implicadas no modelo. Os modelos que permiten algunha forma de dependencia entre termos serán discutidos máis adiante neste capítulo.

Lembre que un documento deste modelo é un vector de 1s e 0s que representa a presenza e ausencia de termos. Por exemplo, se houbese cinco termos indexados, unha das representacións do documento podería ser (1, 0, 0, 1, 1), é dicir, o documento contén os termos 1, 4 e 5. Para calcular a probabilidade de que este documento se produza no conxunto correspondente, necesitamos as probabilidades de que os termos sexan 1 ou 0 no conxunto correspondente. Se $páx_{eu}$ é a probabilidade de que o termo *eu* ocorre (ten o valor 1) nun documento do conxunto relevante, entón a probabilidade do noso documento de exemplo ocorrendo no conxunto correspondente é $páx_1 \times (1 - páx_2) \times (1 - páx_3) \times páx_4 \times páx_5$. A probabilidade $(1 - páx_2)$ é a probabilidade do termo 2 *non* que se producen no conxunto correspondente. Para o conxunto non relevante, usamos s_{eu} para representar a probabilidade de termo *eu* ocorrendo. [5]

Volvendo á relación de probabilidade, usando $páx_{eu}$ e s_{eu} dános unha puntuación de

$$\frac{P(D \mid R)}{P(D \mid NR)} = \prod_{i:\, d_i = 1} \frac{páx_{eu}}{s_{eu}} \cdot \prod_{i:\, d_i = 0} \frac{1 - páx_{eu}}{1 - s_{eu}}$$

onde $\prod_{i:\, d_i = 1}$ significa que é un produto superior aos termos que teñen o valor 1 in o documento. Agora podemos facer un pouco de manipulación matemática para obter:

$$\prod_{i:\, d_i = 1} páx_{eu} \cdot \left(\frac{s_{eu}}{} \right) \cdot \prod_{i:\, d_i = 1} \frac{1 - s_{eu}}{1 - páx_{eu}} \cdot \prod_{i:\, d_i = 1} \frac{1 - páx_i)}{1 - s_{eu}} \cdot \prod_{i:\, d_i = 0} \frac{1 - páx_{eu}}{1 - s_{eu}}$$

$$= \prod_{i:\, d_i = 1} \frac{páx_{eu}(1 - s_i)}{s_{eu}(1 - páx_i)} \cdot \prod_{eu} \frac{1 - s_{eu}}{}$$

O segundo produto é superior a todos os termos e, polo tanto, é o mesmo para todos os documentos, polo que podemos ignoralo para a clasificación. Dado que multiplicar moitos números pequenos pode levar a problemas coa precisión do resultado, podemos usar de xeito equivalente o logaritmo do produto, o que significa que a función de puntuación é:

$$\sum_{i:\, d_i = 1} \text{rexistro} \frac{páx_{eu}(1 - s_i)}{s_{eu}(1 - páx_i)}$$

[5] En moitas descricións deste modelo, $páx_{eu}$ e q_{eu} úsanse para estas probabilidades. Usamos s_{eu} para evitar confusións co q_{eu} usado para representar termos de consulta.

Quizais se estea preguntando onde foi a consulta, dado que se trata dun algoritmo de clasificación de documentos para unha consulta específica. En moitos casos, a consulta ofrécenos a única información que temos sobre o conxunto relevante. Podemos supor que, en ausencia doutra información, os termos que non estean na consulta terán a mesma probabilidade de aparición nos documentos relevantes e non relevantes (é dicir, $páx_{i=s_i}$). Nese caso, a suma só se realizará por termos que están tanto na consulta como no documento. Isto significa que, dada unha consulta, a puntuación dun documento é simplemente a suma dos pesos termais para todos os termos coincidentes.

Se non temos outra información sobre o conxunto relevante, poderiamos elaborala supostos adicionais que $páx_{eu}$ é unha constante e iso s_{eu} podería estimarse empregando como aproximación o termoocorrencias en toda a colección.Wake o segundo suposto baseouse en que o número de documentos relevantes é moito menor que o número total de documentos da colección. Cun valor de 0,5 para $páx_{eu}$ na función de puntuación descrita anteriormente, isto dá un peso a termo para termo eu de

$$\text{rexistro } \frac{0{,}5\ (1-0{,}5)}{\frac{n_{eu}}{N}(1-0{,}5)} \approx \text{rexistro } \frac{N-n_{eu}}{n_{eu}}$$

onde n_{eu} é o número de documentos que conteñen termo eu, e N é o número de documentos da colección. Isto demostra que, en ausencia de información sobre nos documentos relevantes, o termo peso derivado do modelo de independencia binario é moi similar a un idf peso. Non hai tf compoñente, porque se supoñía que os documentos tiñan características binarias.

Se temos información sobre aparicións de termos nos conxuntos relevantes e non relevantes, pódese resumir nun *táboa de continxencia,* amosado na táboa 7.1. Esta información podería obterse mediante comentarios de relevancia, onde os usuarios se identifican documentos relevantes nos rankings iniciais. Nesta táboa, r_{eu} é o número de relevantes documentos que conteñen termo eu, n_{eu} é o número de documentos que conteñen termo eu, N é o número total de documentos da colección e R é o número de documentos relevantes para esta consulta.

	Relevante	Non relevante	Total
$d_{i=}1$	r_{eu}	$n_{eu} - r_{eu}$	n_{eu}
$d_{i=}0$	$R - r_{eu}$	$N - n_{eu} - R + r_{eu}$	$N - n_{eu}$
Total	R	$N - R$	N

Táboa 7.1. Táboa de continxencia de aparicións de termos para unha consulta concreta

Dada esta táboa, o obvio *estimacións*[6] para $páx_{eu}$ e s_{eu} sería $páx_i = r_{eu} / R$ (o número de documentos relevantes que conteñen un termo dividido polo número total de documentos relevantes) e $s_i = (n_{eu} - r_i) / (N - R)$ (o número de documentos non relevantes que conteñen un termo dividido polo número total de documentos non relevantes. umentos). Non obstante, o uso destas estimacións pode causar un problema se algúns dos as entradas na táboa de continxencias eran ceros. Se r_{eu} era cero, por exemplo, o termo peso sería rexistro 0. Para evitalo, unha solución estándar é engadir 0,5 a cada un conta (e 1 ata os totais), o que nos proporciona estimacións de $páx_i = (r_i + 0{,}5) / (R + 1)$ e $s_i = (n_{eu} - r_i + 0{,}5) / (N - R + 1.0)$. Poñer estas estimacións na función de puntuación ofrécenos:

$$\sum_{i:\, d_i = q_i = 1} \text{rexistro} \frac{(r_i + 0{,}5) / (R - r_i + 0{,}5)}{(n_{eu} - r_i + 0{,}5) / (N - n_{eu} - R + r_i + 0{,}5)}$$

Aínda que esta puntuación do documento suma ponderacións de termos só para os termos de consulta coincidentes, a consulta pode ser pertinente *expandido* para incluír outros termos importantes do conxunto relevante. Teña en conta que se non temos información de relevancia, podemos fixalo r e R a 0, o que daría un $páx_{eu}$ un valor de 0,5 e produciría o *idf*- como o peso termo comentado antes.

Entón, que puntuación ten este documento cando se usa para a clasificación? Non moi bo, resulta. Aínda que proporciona un método para incorporar información de relevancia, na maioría dos casos non dispoñemos desta información e, no seu lugar, empregariamos pesos de termos similares a *idf* pesos. A ausencia dun *tf* o compoñente fai unha diferenza significativa coa eficacia do ranking e a maioría das medidas de eficacia caerán arredor dun 50% se o ranking ignora esta información. Isto significa, por exemplo, que poderiamos ver un 50% menos de documentos relevantes nos primeiros postos se empregásemos o modelo binario de independencia en lugar do mellor *tf.idf* clasificación.

Non obstante, resulta que o modelo de independencia binaria é a base dun dos algoritmos de clasificación máis efectivos e populares, coñecido como BM25.[7]

---

[6] Usamos o termo *estimación* para un valor de probabilidade calculado usando datos como unha táboa de continxencia porque este valor só é unha estimación do verdadeiro valor da probabilidade e cambiaría se se dispoña de máis datos.

[7] BM significa Best Match, e 25 é só un esquema de numeración usado por Robertson e os seus compañeiros de traballo para facer un seguimento das variantes de ponderación (Robertson & Walker, 1994).

7.2.2 O algoritmo de clasificación BM25

BM25 amplía a función de puntuación do modelo de independencia binaria para incluír o peso e termos do documento. A extensión baséase en argumentos probabilísticos e validación experimental, pero non é un modelo formal.

BM25 funcionou moi ben en experimentos de recuperación de TREC e influíu nos algoritmos de clasificación de motores de busca comerciais, incluídos os motores de busca web. Hai algunhas variacións da función de puntuación para BM25, pero a forma máis común é:

$$\sum_{eu \in P \text{ rexistro}} \frac{(r_i + 0{,}5) / (R - r_i + 0{,}5)}{(n_{eu} - r_i + 0{,}5) / (N - n_{eu} - R + r_i + 0{,}5)} \cdot \frac{(k_1 + 1) f_{eu}}{K + f_{eu}} \cdot \frac{(k_2 + 1) qf_{eu}}{k_2 + qf_{eu}}$$

onde a suma está agora en todos os termos da consulta; e N, R, n_{eu}, e r_{eu} son os mesmos que se describen na última sección, coa condición adicional de que r e R póñense a cero se non hai información de relevancia; f_{eu} é a frecuencia do termo eu no documento; qf_{eu} é a frecuencia do termo eu na consulta; e k_1, k_2, e K son parámetros cuxos valores se establecen empiricamente.

A constante k_1 determina como tf compoñente do termo cambia de peso como f_{eu} aumenta. Se $k_1 = 0$, ignoraríase o termo compoñente de frecuencia e só tería a presenza ou ausencia do termo. Se k_1 é grande, o termo compoñente aumentaría case linealmente con f_{eu}. Os experimentos InTREC, un valor típico para k_1 é 1,2, o que provoca o efecto de f_{eu} ser moi non lineal, semellante ao uso de rexistro f nos pesos termais comentados na sección 7.1.2. Isto significa que despois das tres ou catro ocorrencias dun termo, as ocorrencias adicionais terán pouco impacto. O constante k_2 ten un papel similar no termo da consulta peso. Os valores típicos deste parámetro están comprendidos entre 0 e 1.000, o que significa que o rendemento é menos sensible k_2 do que é k_1. Isto débese a que as frecuencias de termos de consulta son moito máis baixas e menos variables que as frecuencias de termos de documentos.

K é un parámetro máis complicado que normaliza o tf compoñente pola lonxitude do documento. En concreto

$$K = k_1 ((1 - b) + b \cdot dl / avdl)$$

onde b é un parámetro, dl é a lonxitude do documento e $avdl$ é a lonxitude media dun documento da colección. A constante b regula o impacto da normalización da lonxitude, onde $b = 0$ non corresponde a ningunha normalización de lonxitude e

$b = 1$ é a normalización total. InTRECexperiments, un valor de $b = 0{,}75$ atopouse eficaz.

Como exemplo de cálculo, consideremos unha consulta con dous termos, "presidente" e "lincoln", cada un dos cales só se produce unha vez na consulta ($qf = 1$). Consideraremos o caso típico en que non temos información relevante (r e R son cero). Supoñamos que estamos a buscar unha colección de 500.000 documentos (N), e que nesta colección, o "presidente" aparece en 40.000 documentos ($n_1 = 40,000$) e "lincoln" aparece en 300 documentos ($n_2 = 300$). No documento que estamos puntuando (que trata do presidente Lincoln), "presidente" aparece 15 veces ($f_1 = 15$) e "lincoln" ocorre 25 veces ($f_2 = 25$). A lonxitude do documento é o 90% da lonxitude media ($dl / avdl = 0{,}9$). Os valores dos parámetros que usamos son $k_1 = 1.2$, $b = 0{,}75$, e $k_2 = 100$. Con estes valores, $K = 1.2 \cdot (0{,}25 + 0{,}75 \cdot 0{,}9) = 1{,}11$, e a puntuación do documento é:

$$BM25(Q, D) = \text{rexistro} \frac{(0 + 0{,}5) / (0 - 0 + 0{,}5)}{(40000 - 0 + 0{,}5) / (500000 - 40000 - 0 + 0 + 0{,}5)} \times \frac{(1.2 + 1) 15}{1{,}11 + 15} \times \frac{(100 + 1) 1}{100 + 1} + \text{rexistro} \frac{(0 + 0{,}5) / (0 - 0 + 0{,}5)}{(300 - 0 + 0{,}5) / (500000 - 300 - 0 + 0 + 0{,}5)} \times \frac{(1.2 + 1) 25}{1{,}11 + 25} \times \frac{(100 + 1) 1}{100 + 1}$$

$$= \text{rexistro } 460000{,}5 / 40000{,}5 \cdot 33 / 16.11 \cdot 101/101 + \text{rexistro } 499700{,}5 / 300{,}5 \cdot 55 / 26.11 \cdot 101/101$$

$$= 2{,}44 \cdot 2.05 \cdot 1 + 7{,}42 \cdot 2.11 \cdot 1 = 5{,}00 + 15{,}66 = 20{,}66$$

Nótese o impacto da primeira parte do peso que, sen información relevante, é case o mesmo que un *idf* peso (como comentamos na sección 7.2.1). Debido a que o termo "lincoln" é moito menos frecuente na colección, ten un significado moito máis alto *idf* compoñente (7,42 fronte a 2,44). A táboa 7.2 ofrece puntuacións para diferentes números de aparicións de termos. Isto amosa a importancia do termo "lincoln" e que incluso unha aparición dun termo pode producir unha gran diferenza na puntuación. Reducir o número de ocorrencias terminais de 25 ou 15 a 1 fai que sexa significativo

non diferenza dramática. Este exemplo tamén demostra que é posible que un documento que conteña un gran número de ocorrencias dun único termo importante poida obter puntuacións superiores a un documento que contén os dous termos de consulta (15,66 fronte a 12,74).

Frecuencia de "Presidente"	Frecuencia de "Lincoln"	BM25 puntuación
15	25	20,66
15	1	12,74
15	0	5.00
1	25	18.2
0	25	15,66

Táboa 7.2. Puntuacións BM25 para un documento de exemplo

O cálculo da puntuación pode parecer complicado, pero lembre que algúns dos cálculos dos pesos do termo poden producirse no momento da indexación, antes de procesar calquera consulta. Se non hai información de relevancia, puntuar un documento implica simplemente engadir os pesos para coincidir os termos de consulta, cun pequeno cálculo adicional se os termos de consulta ocorren máis dunha vez (é dicir, se $qf > 1$). Outro punto importante é que os valores dos parámetros para o algoritmo de clasificación BM25 poden axustarse (é dicir, axustados para obter a mellor eficacia) para cada aplicación. O proceso de axuste descríbese máis adiante na sección 7.7 e no capítulo 8.

Para resumir, BM25 é un algoritmo de clasificación efectivo derivado dun modelo de recuperación de información visto como clasificación. Este modelo céntrase na relevancia tópica e fai unha suposición explícita de que a relevancia é binaria. Na seguinte sección, discutiremos outro modelo probabilístico que incorpora o termo frecuencia directamente no modelo, en lugar de ser engadido como unha extensión para mellorar o rendemento.

7.3 Clasificación baseada en modelos de idiomas

Os modelos de linguaxe úsanse para representar texto nunha variedade de *tecnoloxías da linguaxe,* como o recoñecemento de voz, tradución automática e recoñecemento de escritura a man. A forma máis sinxela de modelo de linguaxe, coñecida como a *unigrama* modelo de lingua, é un *distribución de probabilidade* sobre as palabras do idioma. Isto significa que o modelo de lingua asocia unha probabilidade de ocorrencia con cada palabra do in-

vocabulario dex para unha colección. Por exemplo, se os documentos dunha colección contiñan só cinco palabras diferentes, un modelo de idioma posible para esa colección podería ser (0,2, 0,1, 0,35, 0,25, 0,1), onde cada número é a probabilidade de que apareza unha palabra. Se tratamos cada documento como un *secuencia* de palabras, entón as probabilidades no modelo de lingua predicen cal será a seguinte palabra na secuencia. Por exemplo, se as cinco palabras do noso idioma eran "nena", "gato", "o", "neno" e "tocado", entón as probabilidades predicen cal destas palabras será a seguinte. Estas palabras abarcan todas as posibilidades, polo que as probabilidades deben sumarse a 1. Debido a que se trata dun modelo unigramático, as palabras anteriores non teñen ningún impacto na predición. Con este modelo, por exemplo, é igual de probable que se obteña a secuencia de "gata nena" (probabilidade $0,2 \times 0,1$) como "rapaza tocada" (probabilidade $0,2 \times 0,1$).

En aplicacións como o recoñecemento de voz úsanse modelos de linguaxe n-gram que predicen palabras baseadas en secuencias máis longas. Un n-gramatic predice unha palabra baseada na anterior n - 1 palabras. Os modelos de n-gram máis comúns son *bi-gram (* predicindo en función da palabra anterior) e *trigrama (* predición baseada nas dúas palabras anteriores) modelos. Aínda que os bigrammodels se empregaron na recuperación de información para representar frases de dúas palabras (ver sección 4.3.5), centramos a nosa discusión en modelos unigram porque son máis sinxelos e demostraron ser moi efectivos como base para os algoritmos de clasificación.

Para aplicacións de busca, empregamos modelos de linguaxe para representar o contido actual dun documento. A *tema* é algo do que se fala a miúdo pero raramente se define nas discusións sobre recuperación de información. Neste enfoque, definimos un tema como unha distribución de probabilidade sobre palabras (outras palabras, un modelo lingüístico). Por exemplo, se un documento trata sobre a pesca en Alasca, esperaríamos ver palabras asociadas á pesca e lugares en Alaska con altas probabilidades no modelo de lingua. Se se trata de pescar en Florida, algunhas das palabras de alta probabilidade serán as mesmas, pero haberá palabras de maior probabilidade asociadas a lugares de Florida. Se no seu lugar o documento trata de xogos de pesca para ordenadores, a maioría das palabras de alta probabilidade asociaranse aos fabricantes de xogos e ao uso do ordenador, aínda que aínda haberá algunhas palabras importantes sobre a pesca. Teña en conta que un tema está en idioma, ou *modelo de tema* en suma, contén probabilidades para todas as palabras, non só para as máis importantes. A maioría das palabras terán probabilidades "predeterminadas" que serán iguais para calquera texto, pero as palabras importantes para o tema terán probabilidades inusualmente altas.

Unha representación lingüística dun modelo pode usarse para "xerar" novo texto tomando mostras de palabras segundo a distribución de probabilidade. Se imaxinamos o modelo de linguaxe como un gran balde de palabras, onde determinan as probabilidades

cantas instancias dunha palabra hai no cubo, entón podemos xerar texto acadando (sen mirar), sacando unha palabra, anotándoa, colocando a palabra no balde e debuxando de novo. Ten en conta que non dicimos que poidamos xerar o documento orixinal mediante este proceso. De feito, porque só estamos a usar un modelo unigram, o texto xerado quedará bastante mal, sen estrutura sintáctica. Non obstante, as palabras importantes para o tema do documento aparecerán a miúdo. Intuitivamente, estamos a usar o modelo de linguaxe como modelo moi aproximado para o tema no que estaba a pensar o autor do documento cando o escribía.

Cando o texto se modela como unha secuencia finita de palabras, onde hai en cada punto da secuencia *t* Diferentes palabras posibles, isto corresponde a supoñer a *multinomial* distribución sobre palabras. Aínda que hai alternativas, os modelos lingüísticos multiminais son os máis comúns na recuperación de información.[8] Unha das limitacións dos modelos multinomiais que se sinalou é que non describen texto *burstiness* ben, que é a observación de que unha vez que unha palabra é "sacada do balde", tende a ser sacada repetidamente.

Ademais de representar documentos como modelos de idioma, tamén podemos representar o tema da consulta como modelo de idioma. Neste caso, a intuición é que o modelo de linguaxe é unha representación do tema que a persoa que buscaba información tiña en mente cando escribía a consulta. Isto leva a tres posibilidades obvias para modelos de recuperación baseados en modelos de linguaxe: unha baseada na probabilidade de xerar o texto de consulta a partir dun modelo de linguaxe de documento, unha baseada na xeración de texto de documento a partir dun modelo de linguaxe de consulta e outra baseada sobre a comparación dos modelos de linguaxe que representan os temas de consulta e documento. Nas seguintes dúas seccións, describimos estes modelos de recuperación con máis detalle.

7.3.1 Ránking de probabilidade de consulta

No *probabilidade de consulta* modelo de recuperación, clasificamos os documentos pola probabilidade de que o texto da consulta poida ser xerado polo modelo de linguaxe do documento. Noutras palabras, calculamos a probabilidade de que puidemos sacar as palabras da consulta do "balde" de palabras que representan o documento. Este é un modelo de relevancia tópica, no sentido de que a probabilidade de xeración de consulta é a medida da probabilidade de que un documento sexa o mesmo tema que a consulta.

Xa que comezamos cunha consulta, en xeral gustaríanos calcular $P(D \mid Q)$ para clasificar os documentos. Usando a regra de Bayes, podemos calcular isto por

[8] Discutimos o modelo multinomial no contexto da clasificación no capítulo 9.

$$p(D \mid Q)_{ra} = {}^{nk} P(Q \mid D)\, P(D)$$

onde o símbolo $_{ra} = {}^{nk,}$ como mencionamos anteriormente, significa que a man dereita lado é o rango equivalente ao lado esquerdo (é dicir, podemos ignorar a constante normalizadora *P (Q)), P (D)* é a probabilidade previa dun documento e *P (Q | D)* é a probabilidade de consulta dada o documento. Na maioría dos casos, *P (D)* suponse que é *uniforme (* o mesmo para todos os documentos), polo que non afectará a clasificación. Os modelos que asignan probabilidades previas non uniformes baseadas en, por exemplo, a data do documento ou a lonxitude do documento poden ser útiles nalgunhas aplicacións, pero faremos o suposto uniforme máis sinxelo aquí. Dado ese suposto, o modelo de recuperación especifica os documentos de clasificación por *P (Q | D),* que calculamos empregando o modelo de linguaxe unigram para o documento

$$P(Q \mid D) = \prod_{i=1}^{n} P(q_{eu} \mid D)$$

onde q_{eu} é unha palabra de consulta, e hai *n* palabras na consulta.

Para calcular esta puntuación, necesitamos ter estimacións para o modelo de idioma habilidades $P(q_{eu} \mid D)$. A estimación obvia sería

$$P(q_{eu} \mid D) = \frac{f_{q_{eu}, D}}{\mid D \mid}$$

onde $f_{q_{eu, D}}$ é o número de veces que a palabra *qi* ocorre no documento *D,* e | *D* | é o número de palabras en *D.* Para unha distribución multinomial, este é o *máximo probabilidade* estimación, o que significa que esta é a estimación que fai o observado valor de $f_{q_{eu, D}}$ máis probable. O principal problema con esta estimación é que se algunha das palabras de consulta sae do documento, a puntuación dada pola consulta gústame-modelo de lihood para *P (Q | D)* será cero. É evidente que isto non é apropiado para consultas máis longas. Por exemplo, faltar unha palabra de cada seis non debería producir unha puntuación de cero. Tampouco seremos capaces de distinguir entre documentos que conteñen numerosas palabras de consulta. Ademais, porque estamos a construír un modelo de tema para un documento, as palabras asociadas a ese tema deberían ter algunha probabilidade de aparecer, aínda que non fosen mencionadas no documento. Por exemplo, un modelo de linguaxe que represente un documento sobre xogos de ordenador debería ter algunha probabilidade diferente á cero para a palabra "RPG" aínda que esa palabra non fose mencionada no documento. Unha pequena probabilidade para esa palabra permitirá ao documento recibir unha puntuación diferente de cero para a consulta "Xogos de ordenador RPG", aínda que será inferior á puntuación dun documento que contén as tres palabras.

Alisado é unha técnica para evitar este problema de estimación e superar *escaseza de datos,* o que significa que normalmente non dispoñemos de grandes cantidades de texto para as estimacións de probabilidade do modelo de lingua. O enfoque xeral para suavizar é baixar (ou *desconto)* as estimacións de probabilidade das palabras que se ven no texto do documento e asigna esa probabilidade de "sobra" ás estimacións das palabras que non se ven no texto. As estimacións de palabras non vistas adoitan basearse na frecuencia de aparición de palabras en todo o documento

colección. Se $P(q_{eu} \mid C)$ é a probabilidade de palabra de consulta *eu* no *modelo de linguaxe de colección* para a recollida de documentos *C,* entón a estimación que empregamos para unha palabra non vista en un documento é $\alpha_D P(q_{eu} \mid C)$, onde α_D é un coe ciente que controla a probabilidade asignado a palabras non vistas. [9] En xeral, α_D pode depender do documento. Para que as probabilidades somen unha, a estimación de probabilidade dunha palabra que se ve nun documento está $(1 - \alpha_D) P(q_{eu} \mid D) + \alpha_D P(q_{eu} \mid C)$.

Déixeo claro, considere un exemplo sinxelo onde só hai tres palabras, w_1, w_2, e w_3, no noso vocabulario índice. Se as probabilidades de recollida para estas tres palabras, baseadas en estimacións de probabilidade máxima máximas, son 0,3, 0,5 e 0,2 e as probabilidades do documento baseadas nas estimacións de probabilidade máxima son 0,5, 0,5 e 0,0, entón o *alisado* as estimacións de probabilidade para o modelo de linguaxe do documento son:

$$
\begin{aligned}
P(w_1 \mid D) &= (1 - \alpha_D) P(w_1 \mid D) + \alpha_D P(w_1 \mid C) \\
&= (1 - \alpha_D) \cdot 0{,}5 + \alpha_D \cdot 0{,}3 \\
P(w_2 \mid D) &= (1 - \alpha_D) \cdot 0{,}5 + \alpha_D \cdot 0{,}5 \\
P(w_3 \mid D) &= (1 - \alpha_D) \cdot 0{,}0 + \alpha_D \cdot 0{,}2 = \alpha_D \cdot 0{,}2
\end{aligned}
$$

Teña en conta ese termo w_3 ten unha estimación de probabilidade diferente de cero, aínda que non se produciu no texto do documento. Se sumamos estas tres probabilidades, obteremos

$$
\begin{aligned}
P(w_1 \mid D) + P(w_2 \mid D) + P(w_3 \mid D) &= (1 - \alpha_D) \cdot (0{,}5 + 0{,}5) \\
&\quad + \alpha_D \cdot (0{,}3 + 0{,}5 + 0{,}2) \\
&= 1 - \alpha_D + \alpha_D \\
&= 1
\end{aligned}
$$

o que confirma que as probabilidades son consistentes.

[9] A probabilidade de modelo de linguaxe de colección tamén se coñece como *fondo* probabilidade de modelo de linguaxe ou só a probabilidade de fondo.

Diferentes formas de estimación resultan de especificar o valor de α_D. A opción máis sinxela sería fixala nunha constante, é dicir, $\alpha_D = \lambda$. A linguaxe da colección modelo de estimación de probabilidade que empregamos para a palabra q_{eu} é $|C_{q_{eu}}| / |C|$, onde c_q é $_{eu}$ o número Algunhas veces aparece unha palabra de consulta na colección de documentos e $|C|$ é o número total de aparicións de palabras na colección. Isto dános unha estimación para $P(q_{eu} | D)$ de:

$$p(q_{eu} | D) = (1 - \lambda) \frac{f_{q_{eu}, D}}{|D|} + \lambda \frac{c_{q_{eu}}}{|C|}$$

Esta forma de suavizado coñécese como *Jelinek-Mercer* método. A substitución desta estimación na puntuación do documento polo modelo de verosimilitude de consulta dá:

$$P(Q | D) = \prod_{i=1}^{n} \left((1 - \lambda) \frac{f_{q_{eu}, D}}{|D|} + \lambda \frac{c_{q_{eu}}}{|C|}\right)$$

Como dixemos antes, multiplicando xuntas moitas cantidades xuntas pode producir problemas de precisión, podemos usar logaritmos para converter esta puntuación nunha suma equivalente de rango do seguinte xeito:

$$\text{rexistro } P(Q | D) = \sum_{i=1}^{n} \text{rexistro} \left((1 - \lambda) \frac{f_{q_{eu}, D}}{|D|} + \lambda \frac{c_{q_{eu}}}{|C|}\right)$$

Pequenos valores de λ producen menos suavizados e, en consecuencia, a consulta tende a actuar máis como un booleano E xa que a ausencia de calquera palabra de consulta penalizará substancialmente a puntuación. Ademais, a ponderación relativa das palabras, medida polas estimacións de probabilidade máxima, será importante para determinar a puntuación. Como λ aproxímase a 1, a ponderación relativa será menos importante e a consulta actúa máis como un booleano OU ou a *partido de nivel de coordinación.*[10] Nas avaliacións de TREC, demostrouse que os valores de λ ao redor de 0,1 funcionan ben para consultas curtas, mentres que os valores ao redor de 0,7 son mellores para consultas máis longas. As consultas curtas adoitan conter só palabras significativas e baixa λ valor favorecerá os documentos que conteñan todas as palabras de consulta. Con consultas moito máis longas, perder unha palabra é moito menos importante e é alto λ fai máis fincapé nos documentos que conteñen varias palabras de alta probabilidade.

Neste momento, pode ocorrerlle que o modelo de recuperación de probabilidade de consulta non teña nada que pareza un *tf.idf* peso, e aínda así demostran os experimentos

[10] Unha coincidencia de nivel de coordinación simplemente clasifica os documentos polo número de consultas coincidentes termos.

que é tan eficaz como o algoritmo de clasificación BM25. Non obstante, podemos demostrar unha relación con *tf.idf* pesos manipulando a puntuación de probabilidade de consulta do seguinte xeito:

$$\text{rexistro } P(Q \mid D) = \sum_{i=1}^{n} \text{rexistro}\left((1-\lambda)\frac{f_{q_{eu},D}}{|D|} + \lambda \frac{c_{q_{eu}}}{|C|}\right)$$

$$= \sum_{i:\, f_{qi,D}>0} \text{rexistro}\left((1-\lambda)\frac{f_{q_{eu},D}}{|D|} + \lambda \frac{c_{q_{eu}}}{|C|}\right) + \sum_{i:\, f_{qi,D}=0} \text{rexistro}\left(\lambda \frac{c_{q_{eu}}}{|C|}\right)$$

$$= \sum_{i:\, f_{qi,D}>0} \text{rexistro}\left((1-\lambda)\frac{f_{qi,D}}{|D|} + \lambda \frac{c_{qi}}{|C|}\right) + \sum_{i=1}^{n} \text{rexistro}\left(\lambda \frac{c_{q_{eu}}}{|C|}\right)$$

$$\overset{\text{rango}}{=} \sum_{i:\, f_{qi,D}>0} \text{rexistro}\left((1-\lambda)\frac{f_{qi,D}}{|D|}\cdot\frac{|D|}{\lambda c_{qi}} + 1\right)$$

Na segunda liña, dividimos a puntuación nas palabras que aparecen no documento e as que non aparecen ($f_{q_{eu},D}=0$). Na terceira liña, engadimos

$$\sum_{i:\, f_{qi,D}>0} \text{rexistro}\left(\lambda \frac{c_{q_{eu}}}{|C|}\right)$$

para o último termo e restalo do primeiro (onde remata no denominador), polo que non hai ningún efecto neto. O último termo agora é o mesmo para todos os documentos e non se pode ignorar para a clasificación. A expresión final dá a puntuación do documento en termos de "peso" para coincidir os termos da consulta. Aínda que este peso non é idéntico a *tf.idf* peso, hai claras similitudes en que é directamente proporcional á frecuencia do termo do documento e inversamente proporcional á frecuencia de recollida.

Admite unha forma de estimación e, polo xeral, máis eficaz de usar un valor de α_D iso depende da lonxitude do documento. Este enfoque coñécese como *Dirichlet* suavizado, por razóns que comentaremos máis adiante e usos

$$\alpha_D = \frac{\mu}{|D| + \mu}$$

onde μ é un parámetro cuxo valor se establece empíricamente. Substituíndo esta expresión para α_D en $(1-\alpha_D)P(q_{eu}|D) + \alpha_D P(q_{eu}|C)$ resulta na fórmula de estimación de probabilidade

$$p(q_{eu} \mid D) = \frac{f_{q_{eu},D} + \mu c_{q_{eu}} / |C|}{|D| + \mu}$$

o que á súa vez leva á seguinte puntuación do documento:

$$\text{rexistro } P(Q \mid D) = \sum_{i=1}^{n} \text{rexistro } \frac{f_{q_{eu},D} + \mu c_{q_{eu}} / |C|}{|D| + \mu}$$

Semellante ao suavizado Jelinek-Mercer, pequenos valores do parámetro (μ neste caso) dan máis importancia á ponderación relativa das palabras e os grandes valores favorecen o número de termos coincidentes. Valores típicos de μ que acadan os mellores resultados nos experimentos TREC están entre 1.000 e 2.000 (lembre que as probabilidades de colección son moi pequenas) e que o suavizado de Dirichlet é xeralmente máis eficaz que Jelinek-Mercer, especialmente para as consultas curtas que son comúns na maioría das aplicacións de busca. .

Entón, de onde vén o suavizado Dirichlet? Resulta que unha distribución de Dirichlet[11] é o xeito natural de especificar coñecementos previos á hora de estimar as probabilidades nunha distribución multinomial. O proceso de *Estimación bayesiana* determina estimacións de probabilidade en función deste coñecemento previo e do texto observado. A estimación de probabilidade resultante pódese ver como combinando o reconto real de palabras do texto con *pseudocontos* da distribución de Dirichlet. Se non tivésemos texto, a estimación de probabilidade para o prazo q_{eu} sería $\mu(c_{q_{eu}} / |C|) / \mu$, que é unha suposición razoable baseada na colección. Canto máis texto temos $_{eu}$ teñen (é dicir, para documentos máis longos), menos influencia terá o coñecemento previo.

Podemos demostrar o cálculo das puntuacións do documento de verosimilitude de consulta empregando o exemplo dado na sección 7.2.2. Os dous termos de consulta son "presidente" e "lincoln". Para o termo "presidente", $f_{q_{eu},D} = 15$, e supoñamos que $c_{q_{eu}} = 160.000$. Para o termo "lincoln", $f_{q_{eu},D} = 25$, e asumiremos iso $c_{q_{eu}} = 2.400$. O número unha serie de aparicións de palabras no documento $|d|$ suponse que é $_{eu}$ 1.800, e o número de aparicións de palabras na colección é 10^9 (500.000 documentos unha media de 2.000 palabras). O valor de μ usado é 2.000. Dados estes números, a puntuación do documento é:

[11] Chamado así polo matemático alemán Johann Peter Gustav Lejeune Dirichlet (o o nome empregado parece variar).

$$QL(Q, D) = \text{rexistro} \frac{15 + 2000 \times (1.6 \times 10^{5} / 10^{9})}{1800 + 2000} + \text{rexistro} \frac{25 + 2000 \times (2400/10^{9})}{1800 + 2000}$$

$$= \text{rexistro}\ (15{,}32 / 3800) + \text{rexistro}\ (25.005 / 3800) = -5.51 + -5{,}02 = -10.53$$

Un número negativo? Lembre que tomamos logaritmos de probabilidades nesta función de puntuación e as probabilidades de aparición de palabras son pequenas. A cuestión importante é a eficacia das clasificacións producidas empregando estas puntuacións. A táboa 7.3 mostra as puntuacións de probabilidade de consulta para as mesmas variacións de ocorrencia de termos que se empregaron na táboa 7.2. Aínda que as puntuacións parecen moi diferentes para BM25 e QL, a clasificación é similar, coa excepción de que o documento que contén 15 ocorrencias de "presidente" e 1 de "lincoln" está máis alto que o documento que contén 0 ocorrencias de "presidente" e 25 ocorrencias de "lin-coln" nas puntuacións QL, mentres que o contrario é certo para BM25.

Frecuencia de "Presidente"	Frecuencia de "Lincoln"	QL puntuación
15	25	- 10,53
15	1	- 13,75
15	0	- 19.05
1	25	- 12,99
0	25	- 14.40

Táboa 7.3. Puntuacións de probabilidade de consulta para un documento de exemplo

Para resumir, a probabilidade de consulta é un simple modelo de recuperación probabilística que incorpora directamente a frecuencia do termo. O problema de chegar a ponderacións de termos efectivos substitúese pola estimación de probabilidade, que se entende mellor e ten unha base formal. A puntuación básica de probabilidade de consulta co alisado de Dirichlet ten unha eficacia similar á BM25, aínda que o fai mellor na maioría das coleccións TREC. Se se utiliza un suavizado máis sofisticado baseado en modelos temáticos (descrito máis adiante na sección 7.6), a probabilidade de consulta supera constantemente a BM25. Isto significa que, en vez de suavizar usando as probabilidades de recollida de palabras, usamos as probabilidades de palabras de documentos similares.

A sinxeleza do marco lingüístico do modelo, combinada coa capacidade de describir unha variedade de aplicacións de recuperación e a eficacia do asociado

algoritmos de clasificación, fan deste enfoque unha boa opción para un modelo de recuperación baseado na relevancia tópica.

7.3.2 Modelos de relevancia e comentarios de pseudo-relevancia

Aínda que o modelo de verosimilitude de consulta básico ten unha serie de vantaxes, limítase en termos de como modela as necesidades e as consultas de información. É difícil, por exemplo, incorporar información sobre documentos relevantes ao algoritmo de clasificación ou representar o feito de que unha consulta é só unha das moitas consultas posibles que se poderían empregar para describir unha necesidade de información concreta. Nesta sección, amosamos como se pode facer estendendo o modelo básico.

Na introdución á sección 7.3, mencionou que é posible representar o tema dunha consulta como un modelo lingüístico. En vez de chamalo modelo de linguaxe de consulta, usamos o nome *modelo de relevancia* xa que representa o tema tratado polos documentos relevantes. A consulta pódese ver como unha mostra moi pequena de texto xerado a partir do modelo de relevancia, e os documentos relevantes son mostras de texto moito maiores do mesmo modelo. Dados algúns exemplos de documentos relevantes para unha consulta, poderiamos estimar as probabilidades do modelo de relevancia e despois usar este modelo para predicir a relevancia de novos documentos. De feito, trátase dunha versión do modelo de clasificación presentado na sección 7.2.1, onde interpretamos $P(D \mid R)$ como a probabilidade de xerar o texto nun documento cun modelo de relevancia. Isto tamén se chama o *probabilidade do documento* modelo. Aínda que este modelo, a diferenza do modelo de independencia binaria, incorpora directamente a frecuencia do termo, resulta que $P(D \mid R)$ é difícil calcular e comparar entre documentos. Isto débese a que os documentos conteñen un número moi grande de palabras comparadas a unha consulta. Considere dous documentos D_a e D_b, por exemplo, que contén 5 e 500 palabras respectivamente. Debido á gran diferenza no número de palabras implicado, a comparación de $P(D_a \mid R)$ e $P(D_b \mid R)$ para a clasificación será máis di ffi culto que comparar $P(Q \mid D_a)$ e $P(Q \mid D_b)$, que usan a mesma consulta e representacións suavizadas para os documentos. Ademais, aínda temos a probe- lema de obter exemplos de documentos relevantes.

Non obstante, hai outra alternativa. Se podemos estimar un modelo de relevancia a partir dunha consulta, podemos comparar este modelo de linguaxe directamente co modelo dun documento. Os documentos clasificaranse entón pola similitude do modelo de documento co modelo de relevancia. É probable que un documento cun modelo moi similar ao modelo de relevancia estea sobre o mesmo tema. A seguinte pregunta obvia é como comparar dous modelos de idiomas. Unha medida ben coñecida da teoría da probabilidade e da teoría da información, a *Diverxencia Kullback-Leibler (* referido como

KL-diverxencia neste libro),[12] mide a diferenza entre dúas distribucións de probabilidade. Dado o *certo* distribución de probabilidade *Páx* e outra distribución *P* é unha aproximación a *P,* a diverxencia KL defínese como:

$$KL(P \mid\mid Q) = \sum_{x} P(x) \text{ rexistro } \frac{P(x)}{Q(x)}$$

Dado que a diverxencia KL sempre é positiva e é maior para as distribucións que están máis separadas, empregamos o *negativo* Diverxencia KL como base para a función de clasificación (é dicir, diferenzas menores significan puntuacións máis altas). Ademais, a diverxencia KL non é simétrica e importa que distribución escollamos como a verdadeira distribución. Se asumimos que a verdadeira distribución é o modelo de relevancia para a consulta (*R)* e a aproximación ao modelo de linguaxe do documento (*D),* entón a diverxencia KL negativa pódese expresar como

$$\sum_{w \in V} P(w \mid R) \text{ rexistro } P(w \mid D) - \sum_{w \in V} P(w \mid R) \text{ rexistro } P(w \mid R)$$

onde a suma é superior a todas as palabras *w* no vocabulario *V.* O segundo termo do lado dereito desta ecuación non depende do documento e pode ignorarse para clasificalo. Dada unha estimación simple de máxima probabilidade para *P (w | R),* baseado na frecuencia no texto da consulta ($f_{w, Q}$) e o número de palabras da consulta (| *Q* |), a puntuación dun documento será:

$$\sum_{w \in V} \frac{f_{w, Q}}{| Q |} \text{ rexistro } P(w \mid D)$$

Aínda que este resumo é superior a todas as palabras do vocabulario, as palabras que non aparecen na consulta teñen unha estimación de probabilidade máxima cero e non contribuirán á puntuación. Ademais, consulta palabras con frecuencia *k* contribuirá *k* × rexistro *P (w | D)* á puntuación. Isto significa que esta puntuación é equivalente á puntuación de probabilidade de consulta descrita na sección anterior. Noutras palabras, a probabilidade de consulta é un caso especial dun modelo de recuperación que se clasifica comparando un modelo de relevancia baseado nunha consulta cun modelo de linguaxe de documento.

A vantaxe do modelo máis xeral é que non se restrinxe ao método simple de estimar o modelo de relevancia usando frecuencias de termos de consulta. Se nós

[12] A diverxencia KL tamén se denomina diverxencia de información, ganancia de información ou relativo entropia.

considere as palabras de consulta como unha mostra do modelo de relevancia, entón parece razoable basear a probabilidade dunha nova palabra de mostra nas palabras de consulta que vimos. Noutras palabras, a probabilidade de tirar unha palabra w fóra do "balde" que representa o modelo de relevancia debería depender do n palabras de consulta que acabamos de sacar. Máis formalmente, podemos relacionar a probabilidade de w ao condicional probabilidade de observar w dado que acabamos de observar as palabras de consulta $q_{1...} q_n$ pola aproximación:

$$P(w \mid R) \approx P(w \mid q_{1...} q_n)$$

Por definición, podemos expresar a probabilidade condicional en termos da probabilidade conxunta de observar w coas palabras de consulta:

$$P(w \mid R) \approx \frac{P(w, q_{1...} q_n)}{P(q_{1...} q_n)}$$

$P(q_{1...} q_n)$ é unha constante normalizadora e calcúlase como:

$$P(q_{1...} q_n) = \sum_{w \in V} P(w, q_{1...} q_n)$$

Agora a cuestión é como estimar a probabilidade conxunta $P(w, q_{1...} q_n)$. Dado un conxunto de documentos C representado por modelos de linguaxe, podemos calcular a articulación probabilidade do seguinte xeito:

$$P(w, q_{1...} q_n) = \sum_{D \in C} p(D) P(w, q_{1...} q_n \mid D)$$

Tamén podemos asumir que:

$$P(w, q_{1...} q_n \mid D) = P(w \mid D) \prod_{i=1}^{n} P(q_{eu} \mid D)$$

Cando substituímos esta expresión por $P(w, q_{1...} q_n \mid D)$ na ecuación anterior, obtemos a seguinte estimación para a probabilidade conxunta:

$$P(w, q_{1...} q_n) = \sum_{D \in C} P(D) P(w \mid D) \prod_{i=1}^{n} P(q_{eu} \mid D)$$

Como interpretamos esta fórmula? A probabilidade previa $P(D)$ normalmente asúmese para ser uniforme e pode ignorarse. A expresión $\prod_{i=1}^{n} P(q_{eu} \mid D)$ é, de feito,

a puntuación de probabilidade de consulta para o documento *D.* Isto significa que a estimación para $P(w, q_{1} \ldots q_{n})$ é simplemente unha media ponderada das probabilidades do modelo de linguaxe para *w* nun conxunto de documentos, onde os pesos son as puntuacións de probabilidade de consulta para eses documentos.

A clasificación baseada en modelos relevantes realmente require dous pases. O primeiro pase clasifica os documentos utilizando a probabilidade de consulta para obter os pesos necesarios para a estimación do modelo de relevancia. No segundo paso, usamos a diverxencia KL para clasificar os documentos comparando o modelo de relevancia e o modelo de documento. Teña en conta tamén que estamos efectivamente engadindo palabras á consulta suavizando o modelo de relevancia empregando documentos similares á consulta. Moitas palabras que tiñan cero probabilidades no modelo de relevancia baseado nas estimacións de frecuencia de consulta terán agora valores distintos de cero. O que estamos a describir aquí é *exactamente* o proceso de retroalimentación de pseudo-relevancia descrito na sección 6.2.2. En outras palabras, os modelos de relevancia proporcionan un modelo de recuperación formal de retroalimentación de pseudo-relevancia e expansión de consultas. O seguinte é un resumo dos pasos implicados na clasificación mediante modelos de relevancia:

1. Clasifique os documentos utilizando a puntuación de probabilidade de consulta para a consulta *P.*
2. Seleccione un número dos documentos máis ben clasificados para ser o conxunto *C.*
3. Calcula as probabilidades do modelo de relevancia $P(w \mid R)$ empregando a estimación para $P(w, q_{1} \ldots q_{n})$.
4. Volva clasificar os documentos usando a puntuación de diverxencia KL:[13]

$$\sum_{w} P(w \mid R) \text{ rexistro } P(w \mid D)$$

Algúns destes pasos requiren máis explicacións. Nos pasos 1 e 4, as probabilidades do modelo de linguaxe do documento ($P(w \mid D)$) debe estimarse empregando o suavizado Dirich- let. No paso 2, o modelo permite o conxunto *C* para ser toda a colección, pero porque os documentos de baixo rango teñen pouco efecto na estimación de $P(w \mid R)$, normalmente só se usan entre 10 e 50 dos mellores documentos. Isto tamén fai o cálculo de $P(w \mid R)$ substancialmente máis rápido.

Por razóns similares, a suma no paso 4 non se fai sobre todas as palabras do vocabulario. Normalmente só se emprega un pequeno número (10-25) das palabras con maior probabilidade. Ademais, a importancia das palabras de consulta orixinais enfatízase combinando as estimacións de frecuencia de consulta orixinais coa relevancia

[13] Máis precisamente, esta puntuación é a negativa *entropía cruzada* porque eliminamos o termo $\sum_{w \in V} P(w \mid R)$ rexistro $P(w \mid R)$.

estimacións do modelo empregando un enfoque similar a Jelinek-Mercer, é dicir, $\lambda P(w \mid Q) + (1 - \lambda) P(w \mid R)$, onde λ é un parámetro de mestura cuxo valor se determina empíricamente (0,5 é un valor típico para experimentos TREC). Esta combinación deixa claro que a estimación de modelos de relevancia é basicamente un proceso de ampliación e suavización de consultas.

A seguinte pregunta importante, como para todos os modelos de recuperación, é o ben que funciona. Baseado en experimentos TREC, a clasificación mediante modelos de relevancia é unha das mellores técnicas de retroalimentación de pseudo-relevancia. Ademais, os modelos de relevancia producen unha mellora significativa na eficacia en comparación coa clasificación de probabilidade de consulta promediada en varias consultas. Non obstante, como todas as técnicas actuais de retroalimentación de pseudo-relevancia, as melloras non son consistentes e algunhas consultas poden producir peores clasificacións ou resultados estraños.

As táboas 7.4 e 7.5 mostran as 16 palabras de maior probabilidade de modelos relevantes estimadas empregando esta técnica con algunhas consultas de exemplo e unha gran colección de TRECnove historias dos noventa.[14] A táboa 7.4 utiliza os 10 primeiros documentos do ránking de probabilidade de consulta para construír o modelo de relevancia, mentres que a táboa 7.5 usa os 50 primeiros documentos.

O primeiro que hai que notar é que, aínda que as palabras son razoables, dependen moito da colección de documentos que se usa. Na colección de novas TREC, por exemplo, moitas das historias que mencionan Abraham Lincoln están sobre o tema do cuarto Lincoln na White House, que o presidente Clinton utilizou para os hóspedes e o presidente Lincoln usou como ano durante a guerra civil. Este tipo de historias reflíctense nas palabras de maior probabilidade para as consultas "presidente lincoln" e "abraham lincoln". Ampliar a consulta empregando estas palabras favorecería claramente a recuperación deste tipo de historia en vez de máis biografías xerais de Lincoln. A segunda observación é que non hai moita diferenza entre as palabras baseadas en 10 documentos e as palabras baseadas en 50 documentos. Non obstante, as palabras baseadas en 50 documentos son algo máis xeral porque o maior conxunto de documentos contén unha maior variedade de temas. No caso da consulta "peixes tropicais", as palabras de relevancia do modelo baseadas en 10 documentos están claramente máis relacionadas co tema.

En resumo, clasificar comparando un modelo da consulta cun modelo do documento usando diverxencia KL é unha xeneralización da puntuación de probabilidade de consulta.

---

14 Esta é unha colección considerablemente maior da que se usou para xerar o termo asociación táboas do capítulo 6. Esas táboas baseáronse nos datos da pista ROBUST, que consisten en algo máis de medio millón de documentos. Estas táboas xeráronse utilizando todas as coleccións de novas TREC, que suman máis de seis millóns de documentos.

presidente Lincoln	*Abraham Lincoln*	*pesca*	*peixes tropicais*
Lincoln	Lincoln	peixe	peixe
presidente	América	granxa	trópico
cuarto	presidente	salmón	Xapón
dormitorio	fe	novo	acuario
casa	invitado	salvaxe	auga
branco	Abraham	auga	especies
América	novo	collido	acuático
invitado	cuarto	coller	xusto
atender	cristián	etiqueta	China
cama	historia	tempo	coral
Washington	público	comer	fonte
vello	dormitorio	subir	tanque
o ffi ce	guerra	cidade	arrecife
guerra	política	xente	animal
longo	vello	pescadores	tarpon
Abraham	nacional	barco	pesqueira

Táboa 7.4. Termos de maior probabilidade do modelo de relevancia para catro consultas de exemplo (calculadas usando os 10 mellores documentos)

Esta xeneralización permite realizar consultas máis precisas que reflictan a importancia relativa das palabras do tema que o buscador de información tiña en mente cando escribía a consulta. A estimación do modelo de relevancia é unha técnica de retroalimentación de pseudo-relevancia eficaz baseada no marco formal dos modelos de linguaxe, pero como con todas estas técnicas, hai que ter precaución na aplicación da expansión de consultas baseada no modelo de relevancia a unha aplicación específica de recuperación.

Os modelos de linguaxe proporcionan un método formal pero sinxelo de describir modelos de recuperación baseados na relevancia tópica. Pódense desenvolver modelos aínda máis sofisticados incorporando termos de dependencia e frases, por exemplo. Non obstante, a relevancia tópica é só unha parte do necesario para a busca eficaz. Na seguinte sección, centrarémonos nun modelo de recuperación para combinar todas as probas que contribúen á relevancia do usuario, que é o que realmente se preocupa ás persoas que usan un motor de busca.

presidente Lincoln	*Abraham Lincoln*	*pesca*	*peixes tropicais*
Lincoln	Lincoln	peixe	peixe
presidente	presidente	auga	trópico
América	América	coller	auga
novo	Abraham	arrecife	tormenta
nacional	guerra	pescadores	especies
xenial	home	río	barco
branco	civil	novo	mar
guerra	novo	ano	río
Washington	historia	tempo	país
clinton	dúas	baixo	atún
casa	cuarto	barco	mundo
historia	posto	mundo	millóns
tempo	tempo	granxa	estado
centro	política	ángulo	tempo
Kennedy	público	voar	Xapón
cuarto	invitado	troita	milla

Táboa 7.5. Termos de maior probabilidade do modelo de relevancia para catro consultas de exemplo (calculadas usando os 50 mellores documentos)

7.4 Consultas complexas e probas combinadas

A recuperación efectiva require a combinación de moitas probas sobre a potencial relevancia dun documento. No caso dos modelos de recuperación descritos en seccións anteriores, a evidencia consiste en aparicións de palabras que reflicten contido tópico. Non obstante, en xeral pode haber moitos outros tipos de probas que se deben considerar. Mesmo considerando palabras, é posible que queiramos ter en conta se certas palabras se producen preto unhas das outras, se as palabras se producen en estruturas concretas de documentos, como títulos de sección ou títulos, ou se as palabras están relacionadas entre si. Ademais, tamén serán importantes evidencias como a data de publicación, o tipo de documento ou, no caso da busca na web, o número de PageRank. Aínda que un algoritmo de recuperación como a probabilidade de consulta ou BM25 podería ampliarse para incluír algúns destes tipos de probas, é difícil non recorrer a "correccións" heurísticas que fan que o algoritmo de recuperación ffi culto para sintonizar e adaptarse ás novas aplicacións de recuperación. Pola contra, o que realmente necesitamos é un marco onde podemos describir os diferentes tipos de probas, a súa importancia relativa e como deben combinarse. O *rede de inferencia* modelo de recuperación, que foi