UNIVERSIDAD DE SANTIAGO DE COMPOSTELA

Tecnologías de Gestión de Información No Estructurada

Luis Ardévol Mesa

Profesor:
Losada

Escola Técnica Superior de Enxeñaría
Master en Tecnoloxías de Análise
de Datos Masivos: Big Data

Curso 2024-2025

Contents

1 Introducción

El avance en la capacidad de generar y almacenar datos ha llevado a la necesidad de transformar los datos crudos en conocimiento accionable que pueda optimizar la toma de decisiones en diversos sectores como salud, seguridad, educación, ciencia y negocios (BI). Para lograrlo, es fundamental desarrollar tecnologías que permitan ver y extraer información útil y conocimiento oculto en los datos, particularmente en grandes cantidades de datos textuales. Gestionar y analizar grandes cantidades de datos textuales puede ayudar a los usuarios a gestionar y hacer uso de este tipo de datos en todo tipo de aplicaciones.

1.1 Datos textuales como *Big Data*

La gestión de texto de lenguaje natural, que abarca desde páginas web y redes sociales hasta literatura científica y documentos gubernamentales, se ha convertido en una prioridad debido a la explosión de datos en temas de todo tipo. Como un tipo especial de *big data*, los datos textuales representan una gran oportunidad para el descubrimiento de conocimiento y optimización de decisiones en múltiples aplicaciones. Un ejemplo de ello son los datos textuales de opiniones (valoraciones de productos, foros, redes sociales, ...).

Dado el volumen creciente de datos textuales, es imposible para una persona consumir toda la información relevante en tiempo real, de modo que se requieren sistemas inteligentes de recuperación de información para facilitar el acceso rápido a los datos necesarios. El mundo genera entre 1 y 2 *exabytes* de datos anualmente, de los cuales una gran cantidad es textual.

Atendiendo a la estructura de los datos, se pueden clasificar en dos tipos:

- Los datos estructurados, que cuentan con esquemas bien definidos, son manejados fácilmente por los ordenadores.
- Los datos no estructurados, como el texto, necesitan procesamiento computacional para interpretar su contenido, al tener una estructura menos explícita.

El procesamiento de lenguaje natural (NLP) aun no ha alcanzado un punto que permita al ordenador entender de forma precisa el texto. Generalmente se usan aproximaciones estadísticas y heurísticas para extraer información y analizar los datos textuales. El texto es de las fuentes de información más útiles ya que

- Es la forma más natural de codificar el conocimiento humano. Por ejemplo, el conocimiento científico casi solo existe en literatura científica, los manuales técnicos dan explicaciones detalladas de como operar aparatos, ...
- Es el tipo de información más frecuente.
- Es la forma de información más expresiva.

1.1.1 Recuperación y minería de texto

Para gestionar y explotar grandes cantidades de datos textuales, se suele recurrir a dos técnicas. Se aplica la recuperación de textos sobre una gran cantidad de datos textuales para, sobre los datos relevantes extraídos, aplicar minería y obtener conocimiento que usar en distintas aplicaciones.

Recuperación de texto (*text retrieval*)

No se puede digerir toda la cantidad de información disponible, por lo que son necesarios sistemas inteligentes de recuperación de información para facilitar el acceso rápido a los datos necesarios. Para esto surgen los motores de búsqueda, útiles en cualquier contexto donde haya una gran cantidad de datos textuales, no solo en la *web*.

Minería de texto (DM)

Los datos textuales son ricos en contenido semántico. La minería de texto busca descubrir conocimiento valioso dentro del contenido textual, usando herramientas de *software* inteligentes para descubrir patrones interesantes y opiniones que optimicen la toma de decisiones. El proceso de minería de datos puede describirse como minar los datos textuales para descubrir conocimiento útil.

La minería de datos aún no es tan madura como los motores de búsqueda, ya que el texto tiene una estructura menos explícita. El desarrollo de minería inteligente requiere que los ordenadores entiendan el contenido codificado en el texto.

1.1.2 Modos de acceso a la información: Pull vs Push

En el modo *pull*, el usuario toma la iniciativa de buscar información en el sistema, mientras que este último juega un papel pasivo y espera la petición del usuario. En las consultas, el usuario especifica la información necesaria y el sistema devuelve documentos que considera relevantes.

En el modo *push*, el sistema recomienda información anticipando las preferencias y necesidades de información del usuario. Suele funcionar bien cuando el usuario tiene una necesidad de información relativamente estable, como un *hobby*. Aquí, el sistema puede conocer las preferencias e intereses del usuario con adelanto.

En la navegación, el usuario se mueve por estructuras que enlazan elementos de información y alcanza información relevante progresivamente. La navegación y las consultas se alternan de forma natural.

1.2 Sistemas de Información Textual (TIS)

Estos sistemas buscan dar acceso a la información: conectan la información adecuada con el usuario adecuado en el momento adecuado. Ejemplos clásicos son:

- Motores de búsqueda: permiten al usuario acceder a información textual a través de consultas.
- Sistemas de recomendación: pueden sugerir información relevante al usuario de forma proactiva.

Se debe realizar un análisis de texto suficiente como para emparejar la información relevante con la información que necesita el usuario. Los elementos de información originales se muestran en su forma original, aunque en ocasiones se muestra a través de resúmenes dinámicos que dependen de la búsqueda del usuario (*snippets*).

1.2.1 Análisis del texto

Un análisis del texto permite adquirir el conocimiento útil codificado en los datos textuales, el cuál no es fácil de obtener sin sintetizar y analizar una gran cantidad de los datos. Por ejemplo, un motor de búsqueda simplemente devuelve las valoraciones relevantes de un producto, mientras que un motor de análisis extrae las opiniones positivas y negativas y compara opiniones de multitud de productos.

Los sistemas de información textual anotan una colección de documentos textuales con estructuras (tópicos) relevantes. Estas estructuras añadidas permiten al usuario buscar con restricciones sobre las mismas o navegar siguiéndolas.

A diferencia del DM, que se mueve bajo la premisa de descubrir y extraer patrones interesantes en los datos textuales, el NLP se mueve bajo la premisa de entender del texto de lenguaje natural de forma parcial, convertirlo en una forma de representación de conocimiento y hacer inferencia basándose en el conocimiento extraido.

1.2.2 Sistemas de información textual

Los TIS integran servicios de análisis de contenido basados en procesamiento de lenguaje natural (NLP) para transformar datos textuales crudos en representaciones más significativas, apoyando así la recuperación, categorización y organización de información. Se suele combinar el aprendizaje automático estadístico con conocimiento linguístico limitado. Las técnicas poco profundas son robustas, pero un análisis semántico profundo solo es factible en dominios específicos.

Algunas habilidades, como la de resumir documentos, requieren técnicas de NLP más profundas que otras, como una simple búsqueda. Sin embargo, la mayorías de TIS usan técnicas poco profundas, como bolsas de palabras.

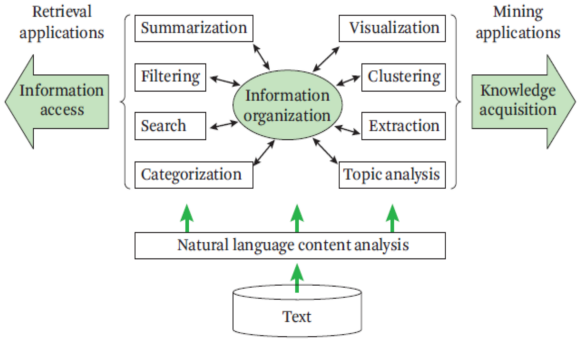

FIGURE I: Esquema conceptual de un TIS

Existen varias formas de organizar la información en un TIS:

- **Búsqueda**: toma una consulta del usuario y devuelve documentos relevantes.
- **Filtrado y Recomendación**: monitoriza el flujo de datos, selecciona los ítems relevantes para los intereses del usuario y luego recomienda los items relevantes (o filtra los no relevantes). Un sistema de recomendación tiene como objetivo recomendar información relevante al usuario, mientras que un sistema de filtrado tiene como objetivo filtrar la información no relevante, permitiendo que el usuario mantenga solo los *items* relevantes.

- **Categorización**: clasifica un objeto textual en una o varias categorías predefinidas. Puede anotar los objetos con todo tipo de categorías significativas, enriqueciendo la representación de los datos textuales. En este caso hay dos tipos de errores, los falsos positivos y los falsos negativos; hay que ver que métrica de rendimiento se le da al clasificador, dando más o menos *penalty* a un error concreto. Estos sistemas son tradicionales, y se usaban para, por ejemplo, clasificar correos electrónicos como *spam*.

- **Resumen**: toma uno o varios documentos de texto y genera un resumen conciso de los contenidos esenciales. Reduce el esfuerzo humano de digerir grandes cantidades de información textual.

- **Análisis de temáticas**: toma una serie de documentos y extrae y analiza los temas en ellos. Los temas aydan a digerir la información y permiten navegar por el texto de forma cómoda. Se puede combinar con datos no textuales, como tiempo, localización u otros metadatos. De este modo, es capaz de generar patrones de interés (tendencias temporales de temáticas, distribución espacio-temporal del tópicos, etc). Es tecnología no supervisada, por lo que los temas no están predefinidos, los da el propio algoritmo.

- **Extracción de información**: Extrae entidades y relaciones de entidades con otras áreas de conocimiento.

- **Clustering**: descubre grupos de objetos textuales similares (términos, oraciones, documentos, etc). Ayuda a los usuarios a explorar un espacio de la información, y es útil para descrubrir *outliers*.

- **Visualización**: permite representar visualmente patrones en los datos textuales.

2 Datos textuales

2.1 Procesamiento de lenguaje natural

El procesamiento de lenguaje natural (NLP) se ocupa de desarrollar técnicas computacionales para permitir que una computadora entienda el significado del texto en lenguaje natural. El NLP es una base fundamental de los sistemas de información textual (TIS) porque la efectividad de un TIS en ayudar a los usuarios a acceder y analizar datos textuales depende en gran medida de qué tan bien el sistema pueda entender el contenido de los datos textuales. Por lo tanto, el análisis de contenido es lógicamente el primer paso en el análisis y gestión de datos textuales.

Mientras que un humano puede entender instantáneamente una oración en su idioma nativo, resulta bastante complicado para un ordenador. En general, esto puede involucrar las siguientes tareas.

- Análisis léxico: determina las unidades significativas básicas en un idioma, como palabras. Determina el significado de cada palabra y delimita las fronteras de las palabras. Esto último puede ser más complicado en idiomas como el chino.
- Análisis sintáctico: determinar cómo se relacionan las palabras entre ellas dentro de una oración. Permite obtener más conocimiento que un análisis léxico, pero no tiene por qué tener idea del significado; no hay conocimiento.
- Análisis semántico: determina el significado de una oración. Esto típicamente implica el cálculo del significado de una oración completa o una unidad mayor basada en los significados de las palabras y su estructura sintáctica. Aquí es cuando se empieza a obtener conocimiento real.
- Análisis pragmático: determina el significado en contexto, por ejemplo, inferir los actos de habla del lenguaje. Una comprensión más profunda del lenguaje natural que el análisis semántico es, por lo tanto, entender aún más el propósito en la comunicación.
- Análisis del discurso: analiza segmentos extensos de texto, considerando varias oraciones, conexiones entre ellas y el contexto en el que se encuentran, considerando el resto de oraciones. Esto es lo que se busca que haga la tecnología ("te lo digo para que hagas algo al respecto"). Este tipo de análisis es muy complicado para textos de lenguaje natural no restringidos, y la tecnología "pre *deep learning*" lo hacía muy mal.

En la figura II, se muestra lo que implica entender una oración muy simple en inglés: *A dog is chansing a boy on the playground*. El análisis léxico en este caso implica determinar las categorías sintácticas (partes del discurso) de todas las palabras (por ejemplo, "*dog*" es un sustantivo y "*chasing*" es un verbo). El análisis sintáctico consiste en determinar que "*a*" y "*boy*" forman una frase nominal. Lo mismo ocurre con "*the*" y "*playground*", y "*on the playground*" es una frase preposicional. El análisis semántico consiste en mapear las frases nominales a entidades y las frases verbales a relaciones para obtener una representación formal del significado de la oración. Por ejemplo, la frase nominal "*a boy*" puede mapearse a una entidad semántica que denota a un niño (es decir, b1), y "*a dog*" a una entidad que denota a un perro (es decir, d1). La frase verbal puede mapearse a un predicado de relación "*chasing*(d1, b1, p1)" como se muestra en la figura. Nótese que con este nivel de comprensión, también

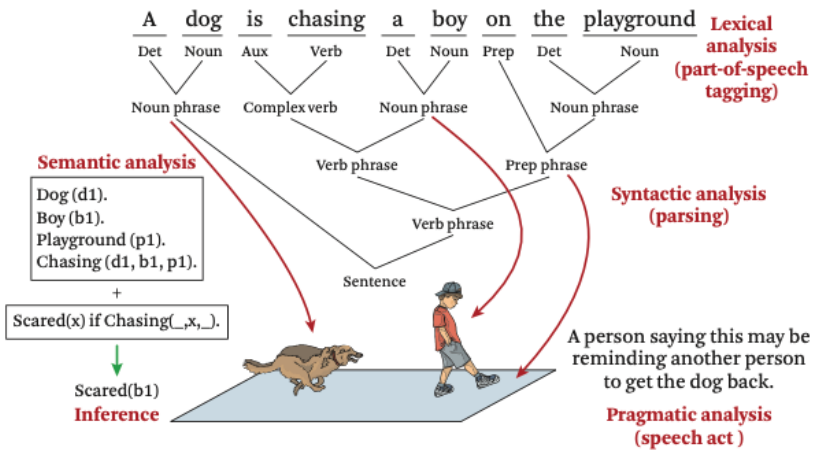

FIGURE II: Ejemplo de las tareas en el procesamiento de lenguaje natural.

se puede inferir información adicional basada en cualquier conocimiento de sentido común relevante. Por ejemplo, si se asume que si alguien está siendo perseguido puede estar asustado, se podría inferir que el niño que está siendo perseguido (b1) puede estar asustado. Finalmente, el análisis pragmático podría revelar además que la persona que dijo esta oración podría tener la intención de solicitar una acción, como recordar al dueño del perro que vigile al perro.

2.1.1 Desafíos y ambigüedades

Si bien es posible derivar una representación semántica clara para una oración simple como la que se muestra en la figura II, en general es muy cimplicado hacer este tipo de análisis para texto en lenguaje natural no restringido. La razón principal de esta dificultad es que el lenguaje natural está diseñado para hacer la comunicación humana eficiente; esto contrasta con un lenguaje de programación, que está diseñado para facilitar la comprensión por parte del ordenador. Específicamente, hay dos razones por las cuales el NLP es muy difícil.

- Se omite mucho conocimiento de sentido común en la comunicación en lenguaje natural porque se asumie que el receptor posee dicho conocimiento (por lo tanto, no hay necesidad de comunicarlo explícitamente).
- Se mantienen muchas ambigüedades, que se asumen que el receptor sabe cómo resolver (por lo tanto, no hay necesidad de gastar palabras para aclararlas). Como resultado, el texto en lenguaje natural está lleno de ambigüedad, y resolverla generalmente implicaría razonar con una gran cantidad de conocimiento de sentido común, lo cual es un desafío en inteligencia artificial.

En este sentido, el NLP es "completo en IA", es decir, tan difícil como cualquier otro problema complicado en inteligencia artificial.Algunos tipos de ambigüedades a las que hay que enfrentarse en NLP son:

- Ambigüedad a nivel de palabra: Una palabra puede tener múltiples categorías sintácticas o significados. Por ejemplo, “diseño” como sustantivo o verbo, (POS ambiguo) o palabras polisémicas (sentido ambiguo).
- Ambigüedad sintáctica: Las oraciones pueden tener múltiples estructuras sintácticas. Por ejemplo, procesamiento de lenguaje natural puede tener dos interpretaciones diferentes: “procesado del lenguaje natural” o “procesado natural del lenguaje” (modificación ambigua). “Un hombre vio a un niño con un telescopio,” que tiene interpretaciones diferentes que conducen a significados distintos (adjunción ambigua de frase preposicional (PP)).

- Resolución de anáforas: Determina a qué se refiere un pronombre, lo cual puede ser poco claro. "*John persuaded Bill to buy a TB for himself*", donde "himself" puede referirse a John o a Bill.
- Presuposición: Implica suposiciones, como en "Él ha dejado de fumar" implicando que fumaba antes. Hacer este tipo de inferencias resulta generalmente complicado.

2.1.2 Historia y estado del arte en NLP

La investigación en NLP se remonta al menos a la década de 1950, cuando los investigadores eran muy optimistas sobre tener ordenadores capaces de entender el lenguaje humano, particularmente con el propósito de la traducción automática. Sin embargo, pronto quedó claro que la traducción de alta calidad completamente automática no podría lograrse sin conocimiento. Un diccionario resultaba insuficiente; en su lugar, haría falta una enciclopedia.

Al darse cuenta de que la traducción automática podría ser demasiado ambiciosa, los investigadores abordaron aplicaciones menos ambiciosas de NLP a finales de la década de 1960 y 1970 con cierto éxito, aunque las técnicas desarrolladas no lograron escalar, teniendo así un impacto limitado en las aplicaciones. Por ejemplo, la gente probó aplicaciones de reconocimiento de voz donde el objetivo es transcribir un discurso. Esta tarea requiere solo una comprensión limitada del lenguaje natural, por lo tanto, es más realista. Se desarrollaron dos proyectos que demostraron la capacidad del ordenador para entender el lenguaje natural: uno es el proyecto Eliza, donde se utilizan reglas superficiales para permitir que un ordenador juegue el papel de un terapeuta para entablar un diálogo en lenguaje natural con un humano; el otro es el proyecto del mundo de bloques, que demostró la viabilidad de la comprensión semántica profunda del lenguaje natural cuando el lenguaje se limita a un dominio de juguete con solo bloques como objetos.

En las décadas de 1970 y 1980, se prestó atención al procesamiento de datos textuales en lenguaje natural del mundo real, particularmente a la comprensión de historias. Se desarrollaron muchos formalismos para la representación del conocimiento y reglas heurísticas de inferencia. Sin embargo, la conclusión general fue que incluso las historias simples son bastante difíciles de entender por un ordenador, confirmando la necesidad de una representación del conocimiento a gran escala e inferencias bajo incertidumbre.

Después de la década de 1980, los investigadores comenzaron a alejarse de los enfoques simbólicos tradicionales (basados en lógica) para el procesamiento del lenguaje natural, que en su mayoría habían demostrado no ser robustos para aplicaciones reales, y prestaron más atención a los enfoques estadísticos, que dieron más éxito; inicialmente en el reconocimiento de voz, y posteriormente en prácticamente el resto de tareas de NLP. A diferencia de los enfoques simbólicos, los enfoques estadísticos tienden a ser más robustos porque dependen menos de reglas generadas por humanos; en su lugar, a menudo aprovechan las regularidades y patrones en los usos empíricos del lenguaje, y se basan únicamente en datos de entrenamiento etiquetados por humanos y en la aplicación de técnicas de aprendizaje automático.

Si bien el conocimiento lingüístico siempre es útil, hoy en día, las técnicas de procesamiento del lenguaje natural más avanzadas tienden a depender en gran medida del uso intensivo de técnicas de aprendizaje automático estadístico, con el conocimiento lingüístico jugando solo un papel secundario. Estas técnicas de NLP estadístico son exitosas para algunas de las tareas de NLP.

2.1.3 Procesamiento de lenguaje natural estadístico

El NLP estadístico se basa en la probabilidad y en la estadística para resolver problemas de lenguaje natural. Algunas tareas comunes son:

- El etiquetado de partes del discurso (POS) es una tarea relativamente fácil, y los etiquetadores de POS en el estado del arte pueden tener una precisión muy alta (por encima del 97% en datos de noticias).
- El análisis sintáctico (*parsing*) es más difícil, aunque el análisis sintáctico parcial se puede hacer con una precisión razonablemente alta (por ejemplo, por encima del 90% para el reconocimiento de frases nominales).

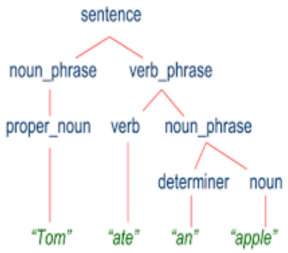

FIGURE III: Ejemplo de análisis sintáctico.

- Análisis completo de estructura (*full structure parsing*): es una tarea muy complicada debido a las ambigüedades en el lenguaje natural.
- Análisis semántico: asigna un significado a una oración. Es una tarea aún más complicada, con éxito limitado. Extracción de información notable(se reconocen entidades nombradas como nombres de personas y organizaciones, y relaciones entre entidades como quién trabaja en qué organización), la desambiguación del sentido de las palabras (distinguir diferentes sentidos de una palabra en diferentes contextos de uso) y el análisis de sentimientos (reconocer opiniones positivas sobre un producto en una reseña de producto) son áreas de interés. Inferencias y habla.
- Análisis de actos: determina la intención detrás de la comunicación. Generalmente, solo es factible en dominios muy limitados.

Procesamiento de lenguaje natural superficial y profundo

En textos arbitrarios, solo el análisis superficial del lenguaje natural se puede hacer de forma robusta. El análisis profundo no escala bien, y no es robusto para textos no restringidos. Este último, además, requiere una cantidad significativa de datos de entrenamiento (etiquetados por humanos) para obtener una precisión razonable.

2.2 Representación de texto

Las técnicas de NLP permiten diseñar muchos tipos diferentes de características informativas para los objetos textuales. Como ejemplo, la oración *A dog is chasing a boy on the playground* en la figura V. Se puede representar esta oración de muchas formas distintas. Primero, siempre se puede representar dicha oración como una cadena de caracteres. Esto es cierto para todos los idiomas, y es quizás la forma más general de representar texto, ya que siempre puede usarse este enfoque para representar cualquier dato textual. Desafortunadamente, la desventaja de esta representación es que no

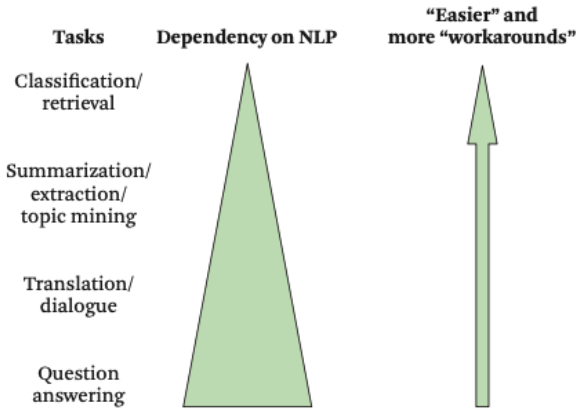

FIGURE IV: Dificultad de varias aplicaciones de NLP.

permite realizar análisis semántico, que a menudo es necesario en muchas aplicaciones de minería de texto. No se están reconociendo palabras, que son la unidad básica de significado para cualquier idioma.

La siguiente versión de la representación del texto es realizar la segmentación de palabras para obtener una secuencia de palabras. En la oración de ejemplo, se obtienen características como "*dog*" y "*chasing*". Con este nivel de representación se tiene mucha más libertad. Al identificar palabras, se puede, por ejemplo, descubrir fácilmente las palabras más frecuentes en un documento o en toda la colección. Estas palabras luego se pueden usar para formar temas. Por lo tanto, representar datos textuales como una secuencia de palabras abre muchas posibilidades de análisis interesantes.

Sin embargo, este nivel de representación es ligeramente menos general que una cadena de caracteres. En algunos idiomas, como el chino, no es tan fácil identificar todos los límites de las palabras. Para resolver este problema, se confia en algunas técnicas especiales para identificar palabras y realizar una segmentación más avanzada que no se base solo en los espacios en blanco (lo que no siempre es 100% preciso). Por lo tanto, la representación de la secuencia de palabras no es tan robusta como la representación de la cadena de caracteres. En inglés, es muy fácil obtener este nivel de representación, por lo que puede usarse todo el tiempo.

Avanzando más en el procesamiento del lenguaje natural, podemos agregar etiquetas de partes del discurso (POS) a las palabras. Esto permite contar, por ejemplo, los sustantivos más frecuentes, o determinar qué tipo de sustantivos están asociados con qué tipo de verbos. Esto abre más oportunidades para un análisis más profundo. Nótese en la figura V se usa un signo más en las características adicionales porque al representar el texto como una secuencia de etiquetas de partes del discurso, no necesariamente se reemplaza la secuencia de palabras original. En su lugar, se agrega esto como una forma adicional de representar datos textuales.

Representar el texto tanto como palabras como etiquetas de POS enriquece la representación de los datos textuales, permitiendo un análisis más profundo y fundamentado. Si se avanza más, entonces se estaría analizando la oración para obtener una estructura sintáctica. Esto abre más análisis de, por ejemplo, los estilos de escritura o la corrección de errores gramaticales.

Avanzando más en el análisis semántico, se podría reconocer "*dog*" como un animal. También podemos reconocer "*boy*" como una persona, y "*playground*" como una ubicación y analizar sus relaciones. Una deducción podría ser que el perro estaba persiguiendo al niño, y el niño está en el

parque. Esto agregará más entidades y relaciones, a través del reconocimiento de relaciones entre entidades. Ahora, se puede contar la persona más frecuente que aparece en toda esta colección de artículos de noticias. Estos tipos de patrones repetidos pueden potencialmente dar muy buenas características.

Esta representación de alto nivel es aún menos robusta que la secuencia de palabras o las etiquetas de POS, pero es muy útil. No siempre es fácil identificar todas las entidades con los tipos correctos y se pueden cometer errores. Las relaciones son aún más difíciles de encontrar. Si se mueve hacia una representación lógica, entonces existen predicados y reglas de inferencia. Con reglas de inferencia se pueden inferir hechos derivados interesantes del texto. No se puede hacer eso todo el tiempo para todo tipo de oraciones, ya que puede llevar un tiempo de computación significativo o una gran cantidad de datos de entrenamiento.

Finalmente, los actos de habla agregarían otro nivel de representación de la intención de esta oración. En este ejemplo, podría ser una solicitud. Saber eso permitiría analizar cosas aún más interesantes sobre el emisor de esta oración. ¿Cuál es la intención de decir eso? ¿Qué escenarios o qué tipos de acciones ocurrirán?

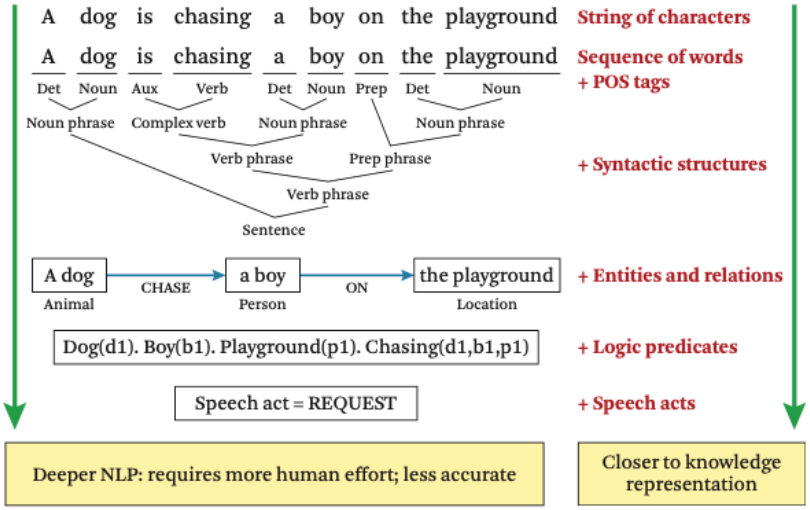

FIGURE V: Diferentes niveles de representación de texto.

Las técnicas de NLP más sofisticadas requieren más esfuerzo humano, y generalmente son menos robustas ya que intentan resolver un problema mucho más difícil. Si se analiza el texto en niveles que representan un análisis más profundo del lenguaje, entonces hay que tolerar posibles errores. Eso también significa que aún es necesario combinar dicho análisis profundo con análisis superficial basado en, por ejemplo, secuencias de palabras. A medida que se avanza, la representación del texto está más cerca de la representación del conocimiento en la mente humana; ese es el propósito de la minería de texto.

En la representación de textos hay que buscar un balance, un compromiso entre un análisis profundo, que puede dar errores pero dará un conocimiento directo que puede ser extraído del texto, y un análisis superficial, que es robusto pero no aportará una represención del conocimiento con el detalle adecuado.

Diferentes representaciones de texto tienden a permitir diferentes análisis, como se muestra en la figura VI. En particular, se agregan gradualmente resultados de análisis más profundos para representar datos textuales que abrirían más oportunidades de representación y capacidades de análisis.

Text Rep	Generality	Enabled Analysis	Examples of Application
String		String processing	Compression
Words		Word relation analysis; topic analysis; sentiment analysis	Thesaurus discovery; topic- and opinion-related applications
+ Syntactic structures		Syntactic graph analysis	Stylistic analysis; structure-based feature extraction
+ Entities & relations		Knowledge graph analysis; information network analysis	Discovery of knowledge and opinions about specific entities
+ Logic predicates		Integrative analysis of scattered knowledge; logic inference	Knowledge assistant for biologists

FIGURE VI: Representación de texto y análisis permitido.

2.3 Modelos de lenguaje estadísticos

Un modelo de lenguaje estadístico proporciona una distribución de probabilidad sobre secuencias de palabras, por ejemplo,

$$p(\textit{Today is Wednesday}) = 0.001$$
$$p(\textit{Today Wednesday is}) = 0.000000001$$
$$p(\textit{The equation has a solution}) = 0.000001$$

Un modelo de lenguaje puede depender del contexto. En el modelo de lenguaje mostrado anteriormente, la secuencia "*The equation has a solution*" tiene una probabilidad menor que "*Today is Wednesday*". Esto puede ser un modelo de lenguaje razonable para describir conversaciones generales, pero puede ser inexacto para describir conversaciones que ocurren en una conferencia de matemáticas, donde la secuencia "*The equation has a solution*" puede ocurrir con más frecuencia que "*Today is Wednesday*".

Sea un modelo de lenguaje, se pueden muestrear secuencias de palabras de acuerdo con la distribución para obtener una muestra de texto. En este sentido, podemos usar dicho modelo para "generar" texto. Por lo tanto, un modelo de lenguaje también se denomina a menudo un modelo generativo para texto.

2.3.1 Usos de modelos del lenguaje

- Reconocimiento de voz: Predice secuencias probables de palabras. Si se escucha, por ejemplo, *John feels*, se puede predecir que la siguiente palabra será *happy*, aunque haya palabras similares acústicamente, como *habit*. Esto es debido a que una es más probable que la otra.
- Categorización de texto: Determina la probabilidad de temas. Por ejemplo, si se tien *baseball* tres vces y *game* una en un artículo, ¿cómo de probable es que el tema sea "deportes"?

- Recuperación de información: Mejora la relevancia en búsquedas. Si un usuario está interesado en noticias de deportes, ¿cómo de probable es que se use la palabra *baseball* en una consulta?

2.3.2 Modelos de lenguaje

Si se enumeran todas las posibles secuencias de palabras y se asigna una probabilidad a cada secuencia, el modelo sería demasiado complejo de estimar, ya que el número de parámetros es potencialmente infinito al tener un número potencialmente infinito de secuencias de palabras. Es decir, nunca se dispondría de suficientes datos como para estimar estos parámetros. Por lo tanto, se deben hacer suposiciones para simplificar el modelo.

El modelo de lenguaje más simple es el modelo de lenguaje unigrama, en el cual se asume que una secuencia de palabras resulta de generar cada palabra de manera independiente. Por lo tanto, la probabilidad de una secuencia de palabras sería igual al producto de la probabilidad de cada palabra. Formalmente, sea V el conjunto de palabras en el vocabulario, y $w_1, \ldots, w_n$ una secuencia de palabras, donde $w_i \in V$ es una palabra. Así, la probabilidad de la secuencia de palabras sería

$$p(w_1, \ldots, w_n) = \prod_{i=1}^{n} p(w_i) \tag{2.1}$$

Dado un modelo de lenguaje unigrama θ, habrá tantos parámetros como palabras en el vocabulario, y estos satisfacen la restricción $\sum_{w \in V} p(w) = 1$. Tal modelo esencialmente especifica una distribución multinomial sobre todas las palabras.

Dado un modelo de lenguaje θ, en general, las probabilidades de generar dos documentos diferentes D_1 y D_2 serían diferentes, es decir, $p(D_1|\theta) \neq p(D_2|\theta)$. Intuitivamente, los documentos con mayor probabilidad serían aquellos que contienen muchas ocurrencias de las palabras de alta probabilidad según $p(w|\theta)$. En este sentido, las palabras de alta probabilidad de θ pueden indicar el tema capturado por θ.

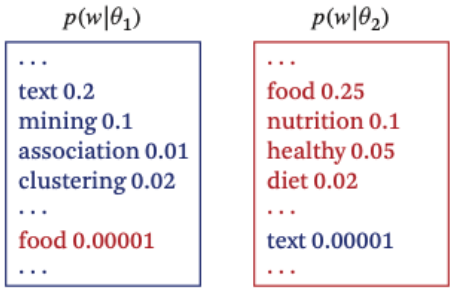

FIGURE VII: Dos ejemplos de modelos de lenguaje unigrama, representando dos temas distintos.

Por ejemplo, los dos modelos de lenguaje unigrama ilustrados en la Figura VII sugieren un tema sobre "minería de texto" y un tema sobre "salud", respectivamente. Intuitivamente, si D es un artículo sobre minería de texto, se esperaría que $p(D|\theta_1) > p(D|\theta_2)$, mientras que si D' es un artículo de blog que discute el control de la dieta, se esperaría lo contrario: $p(D'|\theta_1) < p(D'|\theta_2)$. También se puede esperar que $p(D|\theta_1) > p(D'|\theta_1)$ y $p(D|\theta_2) < p(D'|\theta_2)$.

Sea ahora un documento D que se asume que ha sido generado utilizando un modelo de lenguaje unigrama θ, y se quiere inferir el modelo subyacente θ (es decir, estimar las probabilidades de cada

palabra w, $p(w|\theta)$) basado en el documento observado D. Este es un problema estándar en estadística llamado estimación de parámetros y puede resolverse utilizando muchos métodos diferentes.

Un método popular es el estimador de máxima verosimilitud (ML), que busca un modelo $\hat{\theta}$ que daría a los datos observados la mayor verosimilitud (es decir, que mejor explique los datos):

$$\hat{\theta} = \arg\max_{\theta} p(D|\theta) \tag{2.2}$$

Es fácil demostrar que la estimación ML de un modelo de lenguaje unigrama da a cada palabra una probabilidad igual a su frecuencia relativa en D. Esto es,

$$p(w|\hat{\theta}) = \frac{c(w,D)}{|D|} \tag{2.3}$$

donde $c(w,D)$ es el conteo de la palabra w en D y $|D|$ es la longitud de D, o el número total de palabras en D.

Esta estimación es óptima en el sentido de que maximizaría la probabilidad de los datos observados, pero si realmente es adecuada para una aplicación sigue siendo cuestionable. Por ejemplo, si el objetivo es estimar el modelo de lenguaje en la mente de un autor de un artículo de investigación, y usamos el estimador de máxima verosimilitud para estimar el modelo basado solo en el resumen de un artículo, entonces claramente no es correcto, ya que el modelo estimado asignaría una probabilidad cero a cualquier palabra no vista en el resumen, lo que haría que todo el artículo tuviera una probabilidad cero a menos que solo use palabras del resumen. En general, la estimación de máxima verosimilitud asignaría una probabilidad cero a cualquier *token* o evento no observado en los datos; esto es así porque asignar una probabilidad no nula a dicho *token* quitaría masa de probabilidad que podría haberse asignado a una palabra observada (ya que todas las probabilidades deben sumar 1), reduciendo así la verosimilitud de los datos observados. Para mejorar el estimador de máxima verosimilitud se usan técnicas de suavizado.

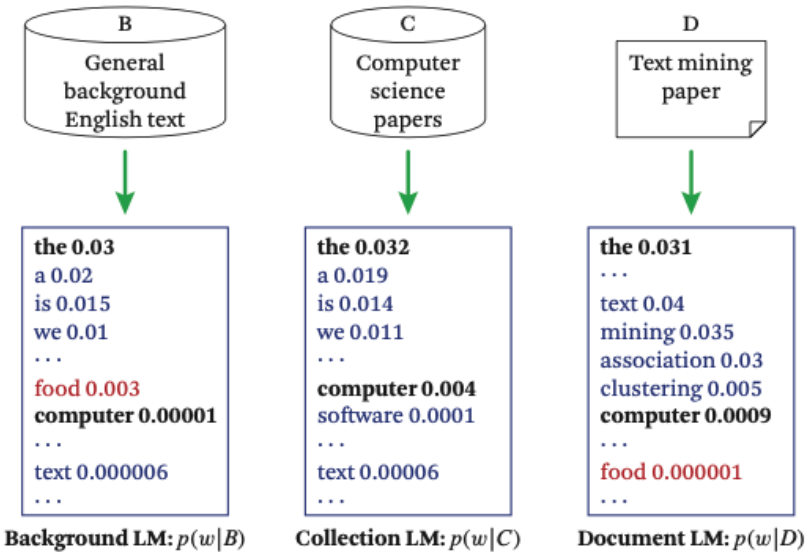

FIGURE VIII: Tres modelos de lenguaje diferentes representando tres temas distintos.

Aunque extremadamente simple, un modelo de lenguaje unigrama es muy útil para el análisis de texto. Por ejemplo, la figura VIII muestra tres modelos de lenguaje unigrama diferentes, estimados en

tres muestras distintas de datos textuales: una base de datos de texto en inglés general, una base de datos de artículos de investigación en ciencias de la computación y un artículo de investigación sobre minería de texto. En general, las palabras con las probabilidades más altas en los tres modelos son aquellas palabras funcionales en inglés, porque tales palabras se usan frecuentemente en cualquier texto. Bajando más en la lista de palabras, se verían más palabras con contenido y palabras temáticas. Las palabras de contenido serán completamente distintas dependiendo de los datos utilizados para la estimación y, por lo tanto, pueden usarse para discriminar los temas en diferentes muestras de texto.

Los modelos de lenguaje unigrama también pueden usarse para realizar análisis semántico de relaciones entre palabras. Por ejemplo, se pueden usarlos para encontrar qué palabras están asociadas semánticamente con una palabra como “computadora”. La idea principal para hacer esto es ver qué otras palabras tienden a co-ocurrir con esa palabra. Específicamente, primero se puede obtener una muestra de documentos (u oraciones) donde se menciona “computadora”. Luego se estima un modelo de lenguaje basado en esta muestra para obtener $p(w|\text{computadora})$. Este modelo dice qué palabras ocurren frecuentemente en el contexto de “computadora”. Sin embargo, las palabras más frecuentes según este modelo probablemente serían palabras funcionales en inglés o palabras que simplemente son comunes en los datos, sin una fuerte asociación con "computadora". Para filtrar las palabras comunes, se necesita un modelo para las mismas, que luego indique qué palabras deben ser filtradas.

Es fácil ver que el modelo de lenguaje inglés general (es decir, un modelo de lenguaje de fondo) serviría bien para el propósito. Se puede usar el modelo de lenguaje de fondo para normalizar el modelo $p(w|\text{computadora})$ y obtener una razón, un *ratio* de probabilidad para cada palabra. Las palabras con valores de razón altos pueden entonces asumirse como asociadas semánticamente con “computadora”, ya que tienden a ocurrir frecuentemente en su contexto, pero no en general. Esto se ilustra en la figura IX.

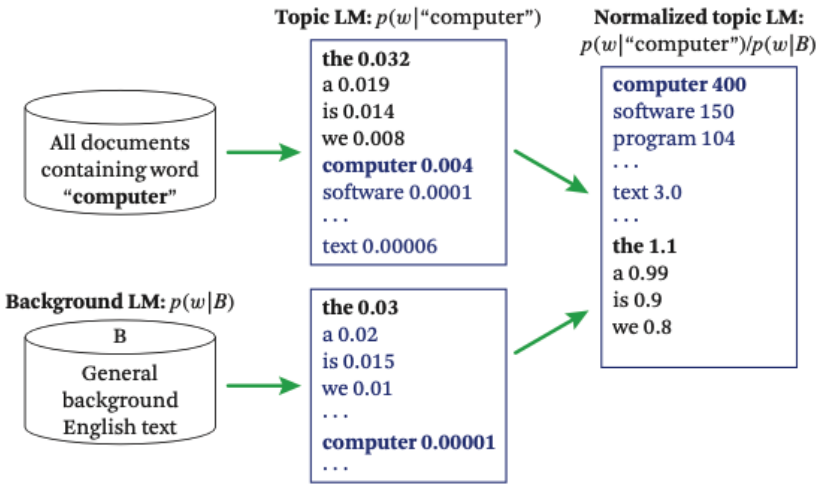

FIGURE IX: Uso de modelos de lenguaje temáticos y modelos de fondo para encontrar palabras semánticamente relacionada.

3 Visión general del acceso a datos textuales

El acceso a datos textuales es la base para el análisis de texto. La tecnología de acceso a texto desempeña dos roles importantes en las aplicaciones de gestión y análisis de texto. Primero, permite la recuperación de los datos textuales más relevantes para un problema de análisis particular, evitando así el procesamiento innecesario de una gran cantidad de datos no relevantes. Segundo, permite la interpretación de cualquier resultado de análisis o conocimiento descubierto en el contexto adecuado y proporciona la procedencia de los datos (origen).

El objetivo general del acceso a datos textuales es conectar a los usuarios con la información correcta en el momento adecuado. Esta conexión se puede realizar de dos maneras: *pull*, donde los usuarios toman la iniciativa de extraer información relevante del sistema, y *push*, donde el sistema toma la iniciativa de ofrecer información relevante a los usuarios.

3.1 Modos de acceso: *pull* y *push*

Dado que los datos textuales son creados para ser consumidos por humanos, estos últimos juegan un papel importante en las aplicaciones de análisis y gestión de datos textuales. Específicamente, los humanos pueden ayudar a seleccionar los datos más relevantes para un problema particular, lo cual es beneficioso al permitir evitar procesar la gran cantidad de datos textuales crudos (lo cual sería ineficiente) y centrarse en analizar la parte más relevante. Seleccionar datos textuales relevantes de una gran colección es la tarea básica del acceso a texto. Esta selección generalmente se basa en una especificación de la necesidad de información de un analista (un usuario), y se puede hacer en dos modos: pull y push. La figura X describe cómo estos modos se ajustan junto con la consulta y la navegación.

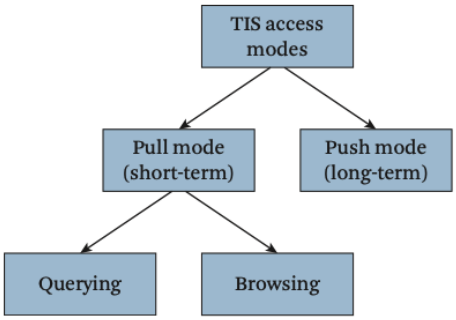

FIGURE X: La dicotomía de los modos de acceso a la información textual.

Pull

En el modo *pull*, el usuario inicia el proceso de acceso para encontrar los datos textuales relevantes, típicamente utilizando un motor de búsqueda. Este modo de acceso a texto es esencial cuando un usuario tiene una necesidad de información *ad hoc*, es decir, una necesidad de información temporal que podría desaparecer una vez que se satisfaga la necesidad. Por ejemplo, un usuario puede tener la necesidad de comprar un producto y, por lo tanto, estar interesado en recuperar todas las opiniones relevantes sobre productos candidatos; después de que el usuario haya comprado el producto, generalmente ya no necesitará dicha información. Otro ejemplo es que durante el proceso de análisis de datos de redes sociales para entender opiniones sobre un evento emergente, el analista también puede decidir explorar información sobre una entidad particular relacionada con el evento (por ejemplo, una persona), lo que también puede desencadenar una actividad de búsqueda.

Si bien la consulta es la forma más común de acceder a datos textuales en el modo *pull*, la navegación es otra forma complementaria de acceder a datos textuales en el modo *pull*, y puede ser muy útil cuando un usuario no sabe cómo formular una consulta efectiva, o encuentra inconveniente ingresar una consulta de palabras clave (por ejemplo, a través de un teléfono inteligente), o simplemente quiere explorar un tema sin un objetivo fijo. De hecho, al buscar en la *web*, los usuarios tienden a mezclar la consulta y la navegación (por ejemplo, al atravesar hipervínculos). En general, se puede considerar la consulta y la navegación como dos formas complementarias de encontrar información relevante en el espacio de información y, cuando la consulta no funciona bien, la navegación puede ser muy útil.

Push

En el modo *push*, el sistema inicia el proceso para recomendar un conjunto de elementos de información relevantes al usuario. Este modo de acceso a la información es generalmente más útil para satisfacer una necesidad de información a largo plazo de un usuario o analista, como pueden ser los intereses de investigación de un investigador, que pueden considerarse relativamente estables a lo largo del tiempo. En comparación, el flujo de información (es decir, los artículos de investigación publicados) es dinámico. Aunque un usuario puede buscar regularmente información relevante con consultas, es más deseable que un sistema de recomendación (también llamado sistema de filtrado) monitoree el flujo de información dinámico y "empuje" cualquier artículo relevante al usuario basado en la coincidencia de los artículos con los intereses del usuario (por ejemplo, en forma de un correo electrónico).

Otro escenario del modo *push* es la recomendación iniciada por el productor (difusión selectiva de información, SDI). En este escenario, el productor de información tiene interés en difundir la información entre los usuarios relevantes, y empujaría un elemento de información a dichos usuarios. La publicidad de información de productos en las páginas de resultados de búsqueda es un ejemplo de esto. La recomendación puede entregarse a través de notificaciones por correo electrónico o recomendarse a través de una página de resultados de un motor de búsqueda.

Necesidad de información

En términos generales, hay dos tipos de necesidades de información: a corto y a largo plazo. Las necesidades a corto plazo están a menudo asociadas con el modo *pull*, y las necesidades a largo plazo están más asociadas con el modo *push*. Una necesidad de información a corto plazo es temporal y generalmente se satisface a través de la búsqueda o navegación en el espacio de información, mientras que una necesidad de información a largo plazo puede satisfacerse mejor a través del filtrado o la

recomendación, donde el sistema tomaría la iniciativa de empujar la información relevante a un usuario.

La recuperación *ad hoc* es extremadamente importante porque las necesidades de información *ad hoc* aparecen con mucha más frecuencia que las necesidades de información a largo plazo. Las técnicas efectivas para la recuperación *ad hoc* generalmente pueden reutilizarse para el filtrado y la recomendación también. Además, en el caso de necesidades de información a largo plazo, es posible recopilar comentarios de los usuarios (*feedback*), que pueden ser explotados. En este sentido, la recuperación *ad hoc* es mucho más difícil, ya que no existe mucha información de retroalimentación de un usuario (es decir, pocos datos de entrenamiento para una consulta particular). Debido a la disponibilidad de datos de entrenamiento, el problema del filtrado o la recomendación generalmente puede resolverse utilizando técnicas de aprendizaje automático supervisado.

3.2 Acceso interactivo multimodal

Idealmente, el sistema debería proporcionar soporte para que los usuarios tengan acceso interactivo multimodal a datos textuales relevantes, de modo que los modos *push* y *pull* estén integrados en el mismo entorno de acceso a la información, y la consulta y la navegación también estén integradas sin problemas. Esto proporciona la máxima flexibilidad a los usuarios y les permitir consultar y navegar a voluntad.

FIGURE XI: Ejemplo de interfaz de navegación con un mapa de temas donde la navegación y la consulta están integradas de manera natural.

En la figura XI se tiene un sistema prototipo donde se ha añadido un mapa de temas construido automáticamente basado en un conjunto de consultas recopiladas en un motor de búsqueda comercial a una interfaz de motor de búsqueda regular para permitir que un usuario navegue por el espacio de información de manera flexible. Con esta interfaz, un usuario puede hacer cualquiera de las siguientes acciones en cualquier momento:

- Consulta (salto de largo alcance). Cuando un usuario envía una nueva consulta a través del cuadro de búsqueda, los resultados de un motor de búsqueda se muestran en el panel derecho. Al mismo tiempo, la parte relevante de un mapa de temas también se muestra en el panel izquierdo para facilitar la navegación si el usuario lo desea.

- Navegación en el mapa (caminata de corto alcance). El panel izquierdo de la interfaz permite que un usuario navegue por el mapa. Cuando un usuario hace clic en un nodo del mapa, este panel se actualizará y se mostrará una vista local con el nodo clicado como el foco actual. En la

vista local, mostramos los padres, los hijos y los vecinos horizontales del nodo actual en foco (etiquetado como "centro" en la interfaz). Un usuario puede así hacer *zoom* en un nodo hijo, hacer *zoom* hacia afuera a un nodo padre, o navegar a un nodo vecino horizontal. El número adjunto a un nodo es una puntuación para el nodo que usamos para clasificar los mismos. Dicho mapa permite al usuario "caminar" en el espacio de información para navegar por documentos relevantes sin necesidad de reformular consultas.

- Visualización de una región temática. El usuario puede hacer doble clic en un nodo temático en el mapa para ver los documentos cubiertos en la región temática. El panel de resultados de búsqueda se actualizaría con nuevos resultados correspondientes a los documentos en la región temática seleccionada.
- Visualización de un documento. Dentro del panel de resultados, un usuario puede seleccionar cualquier documento para verlo como en una interfaz de búsqueda estándar.

En la figura XII, se muestra un ejemplo de traza de navegación en la que el usuario comenzó con una consulta "*dining table*", hizo *zoom* en "*asian dining table*", hizo *zoom* hacia afuera de nuevo a "*dining table*", navegó horizontalmente primero a "*dining chair*" y luego a "*dining furniture*", y finalmente hizo *zoom* hacia afuera al tema general "*furniture*" donde el usuario tendría muchas opciones para explorar diferentes tipos de muebles. Si este usuario siente que se necesita un "salto largo", puede usar una nueva consulta para lograrlo. Dado que el mapa puede ocultarse y solo mostrarse cuando el usuario lo necesita, dicha interfaz es una extensión muy natural de la interfaz de búsqueda actual desde la perspectiva del usuario. Así, se puede ver cómo un sistema de acceso a texto puede combinar múltiples modos de acceso a la información para adaptarse a las necesidades actuales de un usuario.

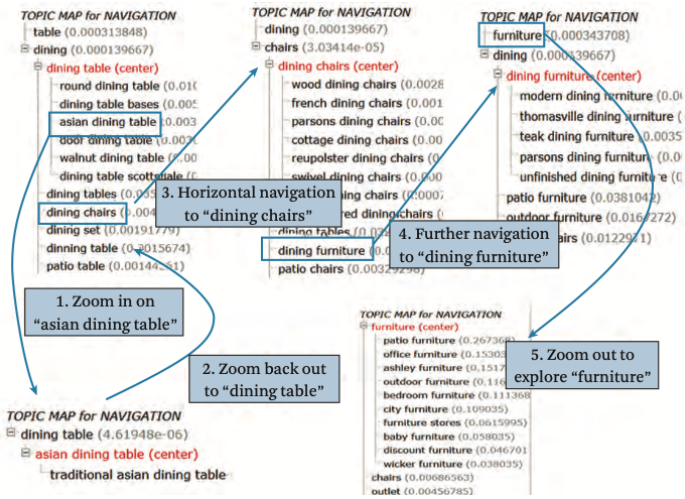

FIGURE XII: Ejemplo de traza de navegación en un mapa de temas, sin necesidad de hacer consultas.

3.3 Recuperación de texto

La herramienta más importante para apoyar el acceso a datos textuales es un motor de búsqueda (SE). Estos proporcionan soporte directo para la consulta y pueden extenderse fácilmente para proporcionar recomendaciones o navegación. Además, las técnicas utilizadas para implementar un motor de búsqueda efectivo a menudo también son útiles para la implementación de un sistema de recomendación, así como para muchas funciones de análisis de texto.

Desde la perspectiva del usuario, el problema de la TR es usar una consulta para encontrar documentos relevantes en una colección de documentos de texto. Esta es una tarea necesaria de forma frecuente, ya que los usuarios a menudo tienen necesidades de información *ad hoc* temporales para varias tareas, y necesitan encontrar la información relevante de inmediato. El sistema para apoyar la TR es un sistema de recuperación de texto, o un motor de búsqueda.

Aunque la TR a veces se usa indistintamente con el término más general "recuperación de información" (IR), este último también incluye la recuperación de otros tipos de información, como imágenes o videos. Sin embargo, las técnicas de recuperación para otros datos no textuales son menos maduras y, como resultado, la recuperación de esta tiende a depender del uso de técnicas de recuperación de texto para hacer coincidir una consulta de palabras clave con datos textuales acompañados de un elemento de datos no textuales. Por ejemplo, los motores de búsqueda de imágenes actuales en la *web* son esencialmente un sistema de TR donde cada imagen está representada por un documento de texto que consiste en cualquier dato textual asociado con la imagen (por ejemplo, título, leyenda, etc).

La tarea de la TR puede ser más o menos complicada dependiendo de consultas específicas y colecciones específicas. Por ejemplo, durante una búsqueda *web*, encontrar páginas de inicio generalmente es fácil, pero encontrar opiniones de personas sobre algún tema (por ejemplo, la política exterior de EE. UU.) sería más difícil. Hay varias razones por las cuales la TR es complicada:

- Una consulta suele ser bastante corta e incompleta (no es un lenguaje formal como SQL).
- La necesidad de información puede ser difícil de describir con precisión, especialmente cuando el usuario no está familiarizado con el tema.
- La comprensión precisa del contenido del documento es difícil. En general, dado que lo que cuenta como la respuesta correcta es subjetivo, incluso cuando los expertos humanos juzgan la relevancia de los documentos, pueden no estar de acuerdo entre sí.

Debido a la falta de estructuras semánticas claras y la dificultad en la comprensión del lenguaje natural, a menudo es un reto recuperar con precisión información relevante para la consulta de un usuario. De hecho, aunque los motores de búsqueda *web* actuales pueden parecer suficiente, aún puede ser complicado para un usuario localizar y recolectar rápidamente toda la información relevante para una tarea. En general, los motores de búsqueda actuales funcionan muy bien para consultas de navegación y consultas informativas simples y populares, pero en el caso de que un usuario tenga una necesidad de información compleja, como analizar opiniones sobre productos para comprar o investigar información médica sobre algunos síntomas, a menudo funcionan mal. Además, los motores de búsqueda actuales generalmente proporcionan poco o ningún soporte para ayudar a los usuarios a digerir y explotar la información recuperada. Como resultado, incluso si un motor de búsqueda puede recuperar la información más relevante, un usuario aún tendría que revisar una larga lista de documentos y leerlos en detalle para digerir completamente el conocimiento enterrado en los datos textuales para realizar su tarea.

3.4 Recuperación de texto vs recuperación de bases de datos

Es útil hacer una comparación del problema de la recuperación de texto (TR) y el problema de la recuperación de bases de datos. Ambas tareas de recuperación son para ayudar a los usuarios a encontrar información relevante, pero debido a la diferencia en los datos gestionados por estas dos tareas, hay muchas diferencias importantes.

Primero, los datos gestionados por un motor de búsqueda y un sistema de bases de datos son diferentes. En las bases de datos, los datos están estructurados, cada campo tiene un significado claramente definido según un esquema. Por lo tanto, los datos pueden verse como una tabla con columnas bien especificadas. Por ejemplo, en un sistema de base de datos de un banco, un campo puede ser nombres de clientes, otro puede ser la dirección, y otro más puede ser el saldo de cada tipo de cuenta. En contraste, los datos gestionados por un motor de búsqueda son texto no estructurado que puede ser difícil de entender para los ordenadores. Así, incluso si una oración dice que una persona vive en una dirección particular, sigue siendo difícil para el ordenador responder una consulta sobre la dirección de una persona en respuesta a una consulta de palabras clave, ya que no hay una estructura definida simple para el texto libre. Aquí es el sistema el que debe ver qué es relevante en cada búsqueda.

En segundo lugar, una consecuencia de la diferencia en los datos es que las consultas que pueden ser soportadas por los dos también son diferentes. Una consulta de base de datos especifica claramente las restricciones en los campos de la tabla de datos y, por lo tanto, los resultados de recuperación esperados (respuestas a la consulta) están muy bien especificados sin ambigüedad. En un motor de búsqueda, sin embargo, las consultas son generalmente palabras clave, que son solo una especificación vaga de qué documentos deben ser devueltos. Incluso si el ordenador puede entender completamente la semántica del texto en lenguaje natural, a menudo resulta que la necesidad de información del usuario es vaga debido a la falta de conocimiento completo sobre la información que se debe encontrar (lo cual es a menudo la razón por la que el usuario quiere encontrar la información en primer lugar). Por ejemplo, en el caso de buscar literatura relevante para un problema de investigación, es poco probable que el usuario pueda especificar clara y completamente qué documentos deben ser devueltos.

Finalmente, los resultados esperados en las dos aplicaciones también son diferentes. En la búsqueda en bases de datos, podemos recuperar elementos de datos muy específicos (por ejemplo, columnas específicas); en la TR, generalmente solo podemos recuperar un conjunto de documentos relevantes. Con pasajes o campos identificados en un documento de texto, un motor de búsqueda también puede recuperar pasajes, pero generalmente es difícil recuperar entidades específicas o valores de atributos como podemos en una base de datos. Esta diferencia no es tan esencial como la diferencia en la especificación vaga de cuál es exactamente la "respuesta correcta" a una consulta, pero es una consecuencia directa de la necesidad de información vaga en la TR.

Debido a estas diferencias, los desafíos en la construcción de una base de datos útil y un motor de búsqueda útil también son algo diferentes. En las bases de datos, dado que los elementos que deben ser devueltos están claramente especificados, no hay desafío en determinar qué elementos de datos satisfacen la consulta del usuario y, por lo tanto, deben ser devueltos; un desafío importante es cómo encontrar las respuestas lo más rápido posible, especialmente cuando hay muchas consultas siendo emitidas al mismo tiempo. Aunque el desafío de la eficiencia también existe en un motor de búsqueda, un desafío más importante es averiguar qué documentos deben ser devueltos para una consulta antes de preocuparse por cómo devolver las respuestas rápidamente. En las aplicaciones de bases de datos, también es muy importante mantener la integridad de los datos, es decir, asegurar que no ocurra ninguna inconsistencia debido a una falla de energía. En la TR, modelar la necesidad de información del usuario y las tareas de búsqueda es importante, nuevamente debido a la dificultad para un usuario de especificar claramente las necesidades de información y la dificultad en el procesamiento del lenguaje natural.

Dado que lo que cuenta como la mejor respuesta a una consulta depende del usuario, en la TR, el usuario es en realidad parte de nuestra entrada (junto con la consulta y el conjunto de documentos). Por lo tanto, no hay una manera matemática de probar que una respuesta es mejor que otra o probar

que un método es mejor que otro. En cambio, siempre tenemos que confiar en la evaluación empírica utilizando algunas colecciones de prueba y usuarios. En contraste, en la investigación de bases de datos, dado que el problema principal es la eficiencia, uno puede probar que un algoritmo es mejor que otro analizando la complejidad computacional o realizando algún estudio de simulación. Sin embargo, al realizar un estudio de simulación (para determinar qué algoritmo es más rápido), también se enfrenta el mismo problema que en la recuperación de texto: la simulación puede no reflejar con precisión las aplicaciones reales. Por lo tanto, un algoritmo que se demuestre más rápido con la simulación puede no ser realmente más rápido para una aplicación particular. De manera similar, un algoritmo de recuperación que se demuestre más efectivo con una colección de prueba puede resultar ser menos efectivo para una aplicación particular o incluso otra colección de prueba. Cómo evaluar de manera confiable los algoritmos de recuperación es en sí mismo un tema de investigación desafiante.

Debido a la diferencia, los dos campos han sido tradicionalmente estudiados en diferentes comunidades con una base de aplicación diferente. Las bases de datos han tenido aplicaciones generalizadas en prácticamente todos los dominios con una industria bien establecida y fuerte. La comunidad de IR que estudia la recuperación de texto ha sido una comunidad interdisciplinaria que involucra la ciencia de la información y la informática, pero no había tenido una base industrial fuerte hasta que nació la *web* a principios de la década de 1990. Desde entonces, la industria de los motores de búsqueda ha dominado, y a medida que más y más información en línea está disponible, las tecnologías de motores de búsqueda (que incluyen TR y otros componentes técnicos como el aprendizaje automático y el procesamiento del lenguaje natural) continuarán creciendo.

3.5 Selección y clasificación de documentos

Sea un colección de documentos (un conjunto de documentos de texto desordenados), la tarea de recuperación de texto (TR) puede definirse como el uso de una consulta de usuario (es decir, una descripción de la necesidad de información del usuario) para identificar un subconjunto de documentos que puedan satisfacer la necesidad de información del usuario.

Formalmente, sea $V = \{w_1, ..., w_N\}$ un conjunto de vocabulario de todas las palabras en un idioma natural particular donde w_i es una palabra. La consulta de un usuario $q = q_1, q_2, ..., q_m$ es una secuencia de palabras, donde $q_i \in V$. De manera similar, un documento $d_i = d_{i1}, ..., d_{im}$ también es una secuencia de palabras donde $d_{ij} \in V$. En general, una consulta es mucho más corta que un documento, ya que la consulta a menudo es escrita por un usuario utilizando un sistema de motor de búsqueda, y los usuarios generalmente no quieren hacer mucho esfuerzo para escribir muchas palabras. Sin embargo, esto no siempre es el caso (por ejemplo, en una búsqueda en *Twitter*, cada documento es un tweet.

La colección de textos $C = \{d_1, ..., d_M\}$ es un conjunto de documentos de texto. En general, se puede asumir que existe un subconjunto de documentos en la colección, es decir, $R(q) \subset C$, que son relevantes para la consulta del usuario q; es decir, son documentos relevantes o documentos útiles para el usuario que escribió la consulta. Naturalmente, este conjunto relevante depende de la consulta q. Sin embargo, qué documentos son relevantes generalmente es desconocido; la consulta del usuario es solo una "pista" de qué documentos deberían estar en el conjunto $R(q)$. Además, diferentes usuarios pueden usar la misma consulta para intentar recuperar conjuntos de documentos relevantes algo diferentes. Esto significa que es irreal esperar que un ordenador devuelva exactamente el conjunto $R(q)$, a diferencia del caso en la búsqueda en bases de datos, donde esto es factible. Por lo tanto, lo mejor que un ordenador puede hacer es devolver una aproximación de $R(q)$, $R'(q)$.

A un alto nivel, hay dos estrategias alternativas para obtener $R'(q)$: selección de documentos vs. clasificación de documentos.

En la selección de documentos, se implementa un clasificador binario para clasificar un documento como relevante o no relevante con respecto a una consulta particular. Es decir, se diseña una función de clasificación binaria, o una función indicadora, $f(q,d) \in \{0,1\}$. Si $f(q,d) = 1$, se asumirá que d es relevante, mientras que si $f(q,d) = 0$, no será relevante. Por lo tanto, $R'(q) = \{d|f(q,d) = 1, d \in C\}$. Usando esta estrategia, el sistema debe estimar la relevancia absoluta, es decir, si un documento es relevante o no.

Una estrategia alternativa es clasificar los documentos y dejar que el usuario decida un punto de corte. Es decir, se implementa una función de clasificación $f(q,d) \in \mathbb{R}$ y se calsifican todos los documentos en valores descendentes de esta función de clasificación. Un usuario navegará por la lista clasificada y se detendrá cuando lo considere apropiado. En este caso, el conjunto $R'(q)$ es en realidad definido en parte por el sistema y en parte por el usuario, ya que el usuario elegiría implícitamente un umbral de puntuación θ basado en la posición de clasificación donde se detuvo. En este caso, $R'(q) = \{d|f(q,d) \geq \theta\}$. Usando esta estrategia, el sistema solo necesita estimar la relevancia relativa de los documentos: qué documentos son más probablemente relevantes.

Dado que la estimación de la relevancia relativa es intuitivamente más fácil que la de la relevancia absoluta, se espera que sea más fácil implementar la estrategia de clasificación. De hecho, la clasificación generalmente se prefiere a la selección de documentos por múltiples razones:

- Debido a la dificultad para un usuario de prescribir los criterios exactos para seleccionar documentos relevantes, es poco probable que el clasificador binario sea preciso. A menudo, la consulta está sobre-restringida o sub-restringida.
  - En el caso de una consulta sobre-restringida, puede que no haya documentos relevantes que coincidan con todas las palabras de la consulta, por lo que forzar una decisión binaria puede resultar en no entregar ningún resultado de búsqueda.
  - Si la consulta está sub-restringida (demasiado general), puede haber demasiados documentos que coincidan con la consulta, lo que resulta en una sobre-entrega.

A menudo es muy difícil para un usuario conocer el nivel "correcto" de especificidad de antemano antes de explorar la colección de documentos. Incluso si el clasificador puede ser preciso, un usuario aún se beneficiaría de la priorización de los documentos relevantes coincidentes para el examen, ya que un usuario solo puede examinar un documento a la vez y algunos documentos relevantes pueden ser más útiles que otros (grados de relevancia). Por todas estas razones, clasificar documentos apropiadamente se convierte en un desafío técnico principal en el diseño de un sistema de recuperación de texto efectivo.

La estrategia de clasificación se muestra además como óptima teóricamente bajo dos suposiciones basadas en el principio de clasificación por probabilidad (Robertson, 1997), que establece que devolver una lista clasificada de documentos en orden descendente de relevancia predicha es la estrategia óptima bajo las siguientes dos suposiciones:

- La utilidad de un documento para un usuario es independiente de la utilidad de cualquier otro documento.
- Un usuario navegará por los resultados secuencialmente.

Entonces, el problema es el siguiente: se tiene una consulta que es una secuencia de palabras, y un documento que también es una secuencia de palabras, y se quiere definir la función $f(q,d)$, capaz

de calcular una puntuación basada en la consulta y el documento. El desafío principal es diseñar una buena función de clasificación que pueda clasificar todos los documentos relevantes por encima de los no relevantes.

Ahora, esto significa que la función debe ser capaz de medir la probabilidad de que un documento d sea relevante para una consulta q. Eso también significa que debe haber alguna forma de definir la relevancia.

4 Modelos de recuperación

4.1 Visión general

Durante muchas décadas, los investigadores han diseñado varios tipos diferentes de modelos de recuperación que se dividen en diferentes categorías. Primero, una familia de los modelos se basa en la idea de similitud. Básicamente, se asume que si un documento es más similar a la consulta que otro documento, se diría que el primer documento es más relevante que el segundo. Entonces, en este caso, la función de clasificación se define como la similitud entre la consulta y el documento. Un ejemplo bien conocido de este caso es el modelo de espacio vectorial (Salton et al. 1975).

El segundo conjunto de modelos se llama modelos de recuperación probabilística. En esta familia de modelos, se sigue una estrategia muy diferente. Se asume que las consultas y los documentos son todas observaciones de variables aleatorias, y se asume que hay una variable aleatoria binaria llamada R (con un valor de 1 o 0) para indicar si un documento es relevante para una consulta. Luego se define la puntuación de un documento con respecto a una consulta como la probabilidad de que esta variable aleatoria R sea igual a 1 dado un documento y una consulta particulares. Hay diferentes casos de esta idea general. Uno es el modelo probabilístico clásico, que se remonta a trabajos realizados en las décadas de 1960 y 1970, otro es el enfoque de modelado de lenguaje, y otro más es el modelo de divergencia de la aleatoriedad.

Uno de los modelos de recuperación más efectivos derivados del marco de recuperación probabilística clásica es BM25 [Robertson y Zaragoza 2009], pero dado que la función de recuperación de BM25 es tan similar a un modelo de recuperación de espacio vectorial, se cubre como una variante del modelo de espacio vectorial.

El tercer tipo de modelo es la inferencia probabilística. Aquí la idea es asociar incertidumbre a las reglas de inferencia. Luego se puede cuantificar la probabilidad de poder demostrar que la consulta se deriva del documento. Esta familia de modelos es teóricamente atractiva, pero en la práctica, a menudo se reducen a modelos esencialmente similares al modelo de espacio vectorial o a un modelo de recuperación probabilística regular.

Finalmente, también hay una familia de modelos que utilizan el pensamiento axiomático. La idea es definir un conjunto de restricciones que se espera que satisfaga una buena función de recuperación. En este caso, el problema es encontrar una buena función de clasificación que pueda satisfacer todas las restricciones deseadas.

Aunque todos estos modelos se basan en diferentes pensamientos, al final las funciones de recuperación tienden a ser muy similares e involucran variables similares. El marco de recuperación axiomática ha demostrado ser efectivo para diagnosticar deficiencias de un modelo de recuperación y desarrollar modelos de recuperación mejorados en consecuencia (por ejemplo, BM25+ [Lv y Zhai 2011]).

4.2 Forma general de la función de recuperación

Antes de introducir modelos específicos, primero se verá la forma común de un modelo de recuperación de última generación y algunas de las ideas comunes utilizadas en todos estos modelos. Esto se ilustra en la figura XIV. Primero, estos modelos se basan en la suposición de usar una representación de bolsa de palabras (*bag-of-words*) del texto. Una representación de bolsa de palabras sigue siendo la principal representación utilizada en todos los motores de búsqueda. Con esta suposición, la puntuación de una consulta como "noticias de campaña presidencial", con respecto a un documento d, se basaría en las puntuaciones calculadas en cada palabra individual de la consulta. Eso significa que la puntuación dependería de la puntuación de cada palabra, como "presidencial", "campaña" y "noticias".

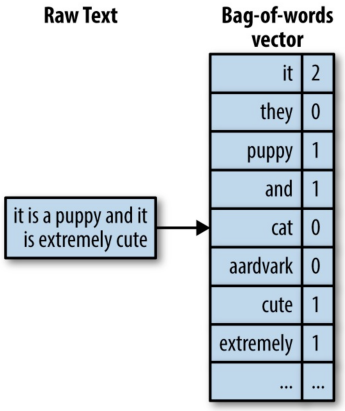

FIGURE XIII: Bolsa de palabras

Hay tres componentes diferentes, cada uno correspondiente a cómo de bien el documento coincide con cada una de las palabras de la consulta. Dentro de estas funciones, vemos una serie de heurísticas.

- Un factor que afecta la función g es cuántas veces ocurre la palabra "presidencial" en cada documento. Esto se llama frecuencia de término (TF). También podríamos denotar esto como $c(\text{presidencial}, d)$. En general, si la palabra ocurre con más frecuencia en el documento, el valor de esta función sería mayor.

- Otro factor es la longitud del documento (DL). En general, si un término ocurre muchas veces en un documento largo, no es tan significativo como si ocurriera el mismo número de veces en un documento corto (ya que se espera que cualquier término ocurra con más frecuencia en un documento largo).

- Finalmente, hay un factor llamado frecuencia de documento (DF). Esto observa con qué frecuencia "presidencial" ocurre al menos una vez en cualquier documento de toda la colección. Esto la frecuencia de documento, o DF, de "presidencial". DF intenta caracterizar la popularidad del término en la colección. En general, coincidir con un término raro en la colección contribuye más a la puntuación general que coincidir con un término común.

TF, DF y DL capturan algunas de las ideas principales utilizadas en prácticamente todos los modelos de recuperación de última generación. En algunos otros modelos, también se usa una probabilidad para caracterizar esta información.

Muchos modelos funcionan igual de bien, pero estos son los cuatro modelos principales que generalmente se consideran de última generación:

- Normalización de longitud pivotada (Singhal et al. 1996).

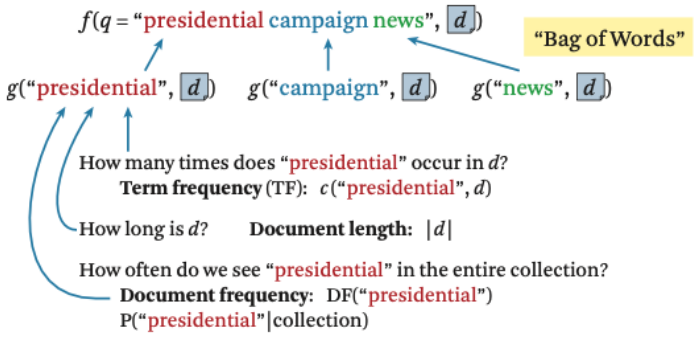

FIGURE XIV: Ideas comunes para la puntuación con una representación de bolsa de palabras.

- Okapi BM25 (Robertson y Zaragoza 2009).
- Verosimilitud de consulta (*query likelihood*) (Ponte y Croft 1998).
- PL2 (Amati y Van Rijsbergen 2002).

En resumen, los puntos principales son los siguientes. Primero, el diseño de una buena función de clasificación requiere una definición computacional de relevancia, y se consigue este objetivo diseñando un modelo de recuperación adecuado. En segundo lugar, muchos modelos son igualmente efectivos, pero no existe un único ganador. Finalmente, las funciones de clasificación de última generación tienden a basarse en las siguientes ideas:

- Representación de bolsa de palabras
- TF y la frecuencia de documento de las palabras.

Esta información es utilizada por una función de clasificación para determinar la contribución general de la coincidencia de una palabra, con un ajuste para la longitud del documento. Estas ideas a menudo se combinan.

4.3 Modelos de recuperación de espacio vectorial

4.4 Modelos de recuperación de espacio vectorial

El modelo de recuperación de espacio vectorial (VS) es un método simple pero efectivo para diseñar funciones de clasificación para la recuperación de información. Es un caso especial de los modelos basados en similitud, donde se asume que la relevancia está aproximadamente correlacionada con la similitud entre un documento y una consulta. En este proceso, inevitablemente hay que hacer una serie de suposiciones. Aquí se asume que si un documento es más similar a una consulta que otro documento, entonces se asumiría que el primer documento es más relevante que el segundo. Esta es la base para clasificar documentos en el modelo de espacio vectorial. Sin embargo, esta no es la única manera de formalizar la relevancia.

Sea un hiperespacio donde cada dimensión corresponde a un término; se pueden trazar los documentos en este espacio ya que están representados como vectores de magnitudes de términos.

En la figura XV, se tiene un espacio tridimensional con tres palabras: programación, biblioteca y presidencial; cada término define una dimensión. Se pueden considerar vectores en este espacio

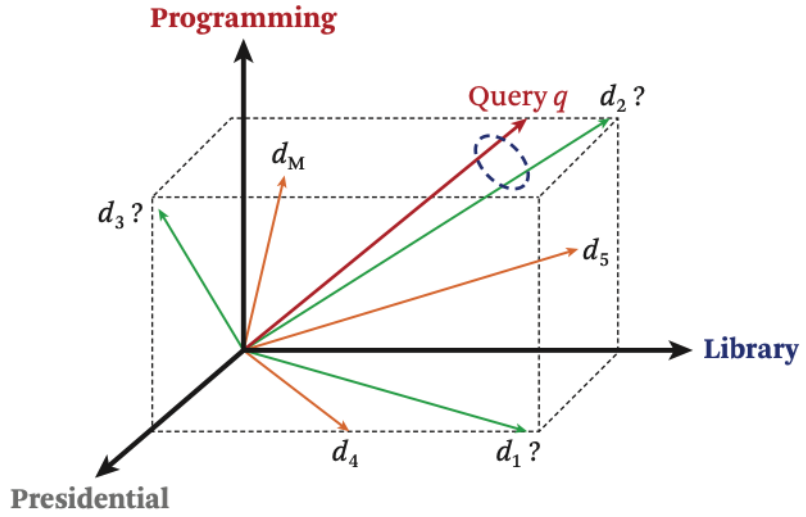

FIGURE XV: Espacio tridimensional con tres palabras: programación, biblioteca y presidencial.

tridimensional, y se asume que todos los documentos y la consulta estarán ubicados en este espacio vectorial (es decir, es espacio es cerrado). Por ejemplo, el vector d_1 representa un documento que probablemente cubre los términos biblioteca y presidencial sin realmente hablar de programación. Para representar esl documento original se usa únicamente este vector, ignorando todo lo demás, incluyendo, por ejemplo, el orden de las palabras. No es una representación óptima, pero a menudo es suficiente para muchos problemas de recuperación.

Al usar esta representación de espacio vectorial, se pueden capturar intuitivamente las diferencias entre los temas de los documentos. De manera similar, se puede colocar la consulta en este espacio como otro vector y medir la similitud entre el vector de la consulta y cada vector de documento. En este caso, por ejemplo, podemos ver fácilmente que d_2 parece ser el más cercano al vector de la consulta y, por lo tanto, d_2 se clasificará por encima de los demás. Esta es la idea principal del modelo de espacio vectorial.

Para ser más precisos, el modelo VS es un marco. En este marco, se hacen algunas suposiciones. Una suposición es que se representará cada documento y consulta mediante un vector de términos. Aquí, un término puede ser cualquier concepto básico como una palabra o una frase, o incluso n-gramas de caracteres o cualquier otra representación de características. Se asume que cada término define una dimensión. Por lo tanto, dado que existen $|V|$ términos en el vocabulario, se define un espacio de $|V|$ dimensiones. Un vector de consulta consistiría en una serie de elementos correspondientes a los pesos de diferentes términos. Cada vector de documento es también similar; tiene una serie de elementos y cada valor de cada elemento indica el peso del término correspondiente. La relevancia en este caso se mide en función de la similitud entre los dos vectores. Por lo tanto, la función de recuperación también se define como la similitud entre el vector de la consulta y el vector del documento.

Para sugerir realmente una función particular que pueda implementarse, hay que refinar los conceptos anteriores.

- Se asume que los conceptos son ortogonales, de lo contrario habrá redundancia (por ejemplo, si dos sinónimos se distinguen de alguna manera como dos conceptos diferentes, se definirían en dos dimensiones diferentes, causando una redundancia o sobreénfasis de la coincidencia de este

concepto, ya que sería como si coincidieras con dos dimensiones cuando en realidad coincidiste con un solo concepto semántico).

- No se especifica cómo definir los pesos de los términos. El peso del término en el vector de la consulta indica la importancia de un término; dependiendo de cómo se asigne, se puede preferir que algunos términos coincidan sobre otros. De manera similar, el peso del término en el documento también es muy significativo: indica qué tan bien caracteriza el término al documento.
- Finalmente, cómo definir la medida de similitud también es poco claro.

4.4.1 Modelo de espacio vectorial simple

En la figura XVI, se ilustra la instanciación más simple del modelo de espacio vectorial. Aquí, se usa cada palabra del vocabulario para definir una dimensión, lo que da $|V|$ dimensiones: esta es la instanciación de bolsa de palabras. Para colocar los vectores en este espacio, la estrategia más simple es usar un vector de bits para representar tanto una consulta como un documento, y eso significa que cada elemento x_i y y_i tomaría un valor de cero o uno: cuando es uno, significa que la palabra correspondiente está presente en el documento o consulta; cuando es cero, está ausente. El vector de documento en general tendría más unos que el vector de consulta, pero aún habrá muchos ceros al ser el vocabulario muy extenso (de forma general).

$$q = (x_1, \ldots, x_N) \qquad x_i, y_i \in \{0, 1\}$$
$$\textbf{1: word } W_i \textbf{ is present}$$
$$d = (y_1, \ldots, y_N) \qquad \textbf{0: word } W_i \textbf{ is absent}$$
$$Sim(q, d) = q.d = x_1 y_1 + \cdots + x_N y_N = \sum_{i=1}^{N} x_i y_i$$

FIGURE XVI: Cálculo de similitud entre una consulta y un vector de documento usando una representación de vector de *bits* y el producto escalar como similitud.

Una medida de similitud comúnmente utilizada es el producto escalar (cuenta el número de términos que coinciden, cuantos términos únicos de la consulta están presentes en el documento). Finalmente, se puede implementar una función de clasificación o *ranking* usando un lenguaje de programación y luego clasificar documentos en un corpus dada una consulta particular.

Para evaluar si este modelo de espacio vectorial más simple realmente funciona bien, se muestra un ejemplo en la figura XVII.

Esta figura muestra algunos documentos de prueba y una consulta simple. La consulta es “noticias sobre la campaña presidencial”. Para este ejemplo, se verán cinco documentos del corpus que cubren diferentes términos en la consulta. La clasificación ideal es la que haría el propio usuario, $R'(q)$. La mayoría de los usuarios estarían de acuerdo en que d_4 y d_3 probablemente son mejores que los otros, ya que estos dos realmente cubren bien la consulta. Coinciden con “noticias”, “presidencial” y “campaña”, por lo que deberían clasificarse en la parte superior. Los otros tres, d_1, d_2 y d_5, no son relevantes.

Ahora hay que ver si el modelo de espacio vectorial podría hacer lo mismo o algo cercano a la clasificación ideal. En la figura XVIII, se muestran dos documentos, d_1 y d_3, junto con la consulta. En

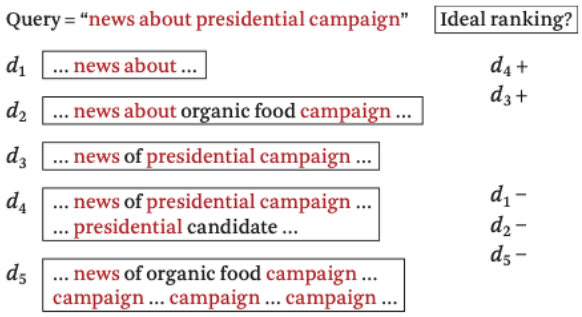

FIGURE XVII: Aplicación de modelo VS de vector de bits e un ejemplo simple.

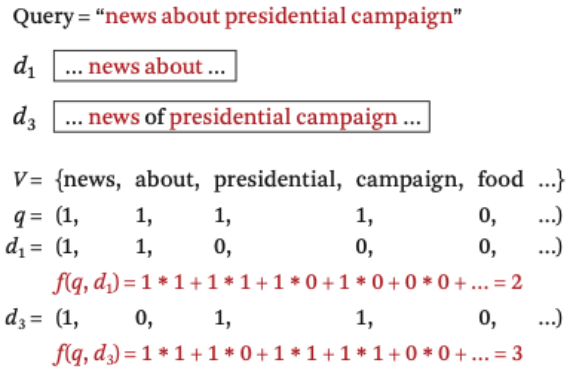

FIGURE XVIII: Cálculo del modelo de recuperación de vector de *bits* en una consulta y el corpus.

el modelo de espacio vectorial, primero se calculan los vectores para estos documentos y la consulta. La consulta tiene cuatro palabras, por lo que para estas cuatro palabras, habría un uno y para el resto habrá ceros. El documento d_1 tiene dos unos, "noticias" y "sobre", mientras que el resto de las dimensiones son ceros. Con los dos vectores, se puede calcular la similitud con el producto escalar, multiplicando los elementos correspondientes en cada vector. Cada par de vectores forma un producto, que representa la similitud entre los dos elementos. En este caso, el resultado será dos. Eso significa que este número es el valor de esta función de puntuación; es simplemente el recuento de cuántos términos únicos de la consulta coinciden en el documento. Esto es cómo interpretamos la puntuación. Ahora también podemos echar un vistazo a d_3. En este caso, puedes ver que el resultado es tres porque d_3 coincidió con las tres palabras de consulta distintas: *news*, *presidential* y *campaign*, mientras que d_1 solo coincidió con dos. Basándonos en esto, d_3 está clasificado por encima de d_1. Eso se ve bastante bien.

Sin embargo, examinando este modelo en detalle, se encuentran algunos problemas. La función de puntuación cuenta el número de términos únicos de la consulta que coinciden en cada documento. Si un documento coincide con más términos únicos de la consulta, entonces se asumirá que el documento es más relevante; eso parece tener sentido. El único problema es que hay tres documentos, d_2, d_3 y d_4, que están empatados con una puntuación de tres. Al inspeccionar más de cerca, parece que d_4 debería estar justo por encima de d_3 ya que d_3 solo mencionó "presidencial" una vez mientras que d_4 lo mencionó muchas más veces. Otro problema es que d_2 y d_3 también tienen la misma puntuación, ya que para d_2, "noticias", "sobre" y "campaña" coincidieron. En d_3, coincidieron "noticias", "presidencial" y "campaña". Intuitivamente, d_3 es más relevante y debería tener una puntuación más alta que d_2. Coincidir con "presidencial" es más importante que coincidir con "sobre" aunque "sobre" y

"presidencial" estén ambos en la consulta. Pero este modelo no hace eso, y eso significa que tenemos que resolver estos problemas.

Para resumir, hablamos sobre cómo instanciar un *vector space model*. Necesitamos hacer tres cosas:

i. Definir las dimensiones (el concepto de lo que es un documento);

ii. Decidir cómo colocar documentos y consultas como vectores en el *vector space*

iii. Definir la similitud entre dos vectores.

Basándonos en esta idea, discutimos una forma muy simple de instanciar el *vector space model*. De hecho, probablemente sea el modelo de espacio vectorial más simple que podemos derivar. Usamos cada palabra para definir una dimensión, con un vector de bits de cero y uno para representar un documento o una consulta. En este caso, solo nos importa la presencia o ausencia de palabras, ignorando la frecuencia. Para una medida de similitud, usamos el *dot product* y demostramos que esta función de puntuación puntúa un documento basado en el número de palabras de consulta distintas que coinciden en él. También demostramos que un modelo de espacio vectorial tan simple todavía no funciona bien y necesitamos mejorarlo.

Modelo VSP mejorado

En esta sección, mejoraremos la representación de este modelo a partir del *bit vector model*. Vimos que la representación de *bit vector* esencialmente cuenta cuántos términos únicos de la consulta coinciden con el documento. En la figura **??** nos gustaría que d_4 se clasifique por encima de d_3, y d_2 realmente no es relevante. El problema aquí es que esta función no puede capturar las siguientes características.

Primero, nos gustaría darle más crédito a d_4 porque coincide con *presidential* más que d_3.

Segundo, coincidir con *presidential* debería ser más importante que coincidir con *about*, porque *about* es una palabra muy común que ocurre en todas partes; no aporta tanto contenido.

Vale la pena pensar en este punto sobre por qué tenemos estos problemas. Si volvemos a las suposiciones que hicimos al instanciar el *VS model*, nos daremos cuenta de que el problema realmente proviene de algunas de esas suposiciones. En particular, tiene que ver con cómo colocamos los vectores en el *vector space*. Naturalmente, para solucionar estos problemas, tenemos que revisar esas suposiciones. Un pensamiento natural es considerar múltiples ocurrencias de un término en un documento en lugar de una representación binaria; deberíamos considerar el *TF* en lugar de solo la ausencia o presencia. Para considerar la diferencia entre un documento donde un término de la consulta ocurrió múltiples veces y uno donde el término de la consulta ocurrió solo una vez, tenemos que considerar la frecuencia del término, el conteo de un término en un documento. La forma más simple de expresar el TF de una palabra w en un documento d es

$$TF(w,d) = \text{count}(w,d). \tag{4.1}$$

Con el *bit vector*, solo capturamos la presencia o ausencia de un término, ignorando el número real de veces que un término ocurrió. Agreguemos la información de conteo de nuevo: representaremos un documento mediante un vector con el peso de cada dimensión. Es decir, los elementos tanto del *query vector* como del *document vector* no serán ceros y unos, sino que serán los conteos de una palabra en la consulta o en el documento, como se ilustra en la figura XIX.

$q = (x_1, \ldots, x_N)$ $\boxed{x_i = \text{count of word } W_i \text{ in query}}$

$d = (y_1, \ldots, y_N)$ $\boxed{y_i = \text{count of word } W_i \text{ in doc}}$

$$Sim(q, d) = q.d = x_1 y_1 + \ldots + x_N y_N = \sum_{i=1}^{N} x_i y_i$$

FIGURE XIX: Representación de un documento con un vector de términos.

5 Motores de búsqueda

5.1 Componentes de los motores de búsqueda

Sea un buscador de proposito general o especializado (vertical o corporativo) los motores de búsqueda tienen una serie de componentes comunes que se pueden identificar en todos ellos. A continuación se describen los componentes más comunes de un motor de búsqueda:

- Tokenizador. Convierte las cadenas de texto curdas en *tokens*. Esto se debe hacer bien y de forma homogenea en todas las partes del sistema, o de lo contrario afectará a las mismas.
- Índice. Indexa los documentos en estructuras de datos apropiadas contra las que se puede buscar. Esto se puede hacer offline, pero debe ser rapido con una cantidad limitada de memoria. En colecciones dinamicas se debe poder borrar, añadir y modificar elementos.
- *Scorer*/*Ranker*. Recibe una consulta y, yendo contra el índice, devuelve una lista rankeada de documentos. Este si es online y debe ser rapido.
- En algunos casos hay *feedback* del usuario. Si se tiene, ese ciclo de *feedback* puede retroalimentar al sistema para hacer una búsqueda mejor. El *relevance feedback* no es muy realista, al ser los usuarios poco propensos a participar. Para eso se tiene el *pseudofeedback*. Aqui el buscador asume que el top 5-10 son muy relevantes. Busca y mejora la búsqueda en base a minería en esos documentos (en general, mejora el rendimiento aunque a veces falla en gran medida).

Desde el punto de vista de la eficiencia, hay que mejorar la velocidad y el uso del disco. Para ello, los sistemas incorporan una estructura de índice invertido (se llama así porque en origen se buscan documentos y en ellos palabras, pero al buscar en una consulta se quiere de palabras a documentos).

Desde el punto de vista de la eficiencia, hay que Por supuesto, uso de el uso de ahorrar lo más posible en disco entonces por ejemplo sistemas incorporados en la estructura de índice invertido (es la forma de buscar de los motores de busqueda,

no se guardan en origen sino comprimidos. En el caso de la web, el posting list(PL) es muy grande y es el gran peso de la estructura de datos. Esta lista se comprime y gano dos cosas: espacio en disco (el posting list debe estar en disco) y auqnue haya que descomprimir, es mas rapido que haberlo guardado todo en disco.

Otro elemenot importante es el tema de los caches para por ejemplo guardar terminos muy frecuentes, y guardar en cache el posting list de ese termino o incluso los resultados de la busqueda.

5.2 Tokenizadores

ademas de tokenizar puede ir acumulando el numero de apariciones de cada token, lo que puede servir para fases posteriores de scoring. No mete pesos TFIDf. El IDF es una metrica global del corpus, por lo que al procesar el documento no se puede calcular la IDF de un termino. La forma del TF depende del modelo de busqueda que se vaya a usar. Es posterior porque si lo hace el tokenizar, este solo serviria para ese modelo explicito.

5.2.1 Doc and term IDs

como el tokenizador hace un parseado basico del texto, tambien puede asignar ids de documentos (los numeros que van al pl). Es mas eficiente guardar los numeros en el indice que todo el texto de una URL. Las palabras del vocabulario tendran un identificador cada una. A medida que detecta nuevas palabras asignara nuevos numeros. La mayoria de las dimensiones tendran cuenta de cero al ser todo el vocabulario. REpresentacion sparse muchos ceros y pocos numeros

Vectorización Representacion numerico de los textos

5.3 Indexador

Produce el indice invertido con sus dos partes, la posting list (lista de ocurrencia) con ocurrencias y el vocabulario o lexicon. Indexa datos mas grandes que la memoria del ordenador que la esta haciendo . Por tanto, no cabe todo el indice en memoria, se deberan guardar subindices por ejemplo y luego mezclar. El indice esta pensado para recuperar informacion o estadisticas de los terminos de forma rapida. Seria inviable ir por cada documento buscando una palabra.

Lexicon: una tabla de información específica de cada termino.

Por defecto, se guarda la posting list por orden de docID para facilitar la compresión. La PL es mucho mayor que el vocabulario. Se asume que el lexicon puede caber en memoria. Lo bueno de este proceso es que es muy paralelizable. Paralalizamos el parseado, cada una de ellas emite informacion de clave valor (termino, docID) (proceso de map) y luego se recoge y se da la posting list ordenada por docID en un proceso de reduce. El tamaño lo conoce porque sabe cuantos se han emitido para cada clave.

SORTING-BASED INDEXING

este proceso en esencia lo que hace es meter cosas en un indice parcial (invertido) hasta llenar la memoria y, al alcanzar ese limite, se vuelca a disco el indice incompleto y sigue con el siguiente. De este modo, se tiene gran cantidad de indices invertidos incompletos en disco. PAra mezclarlo se usa un algoritmo de merge que los mezcla dos a dos. Esto es relativamente rapido.

FORWAR INDEX

El indice invetido es la Estructura de datos basica para que las busquedad vayan rapido.

Tambien existe sin embargo el indice directo, de documentos a palabras. Este indice se usa para mostrar los resultados de la busqueda (un titulo y un snippet) al usuario. El snippet es dinamico, depende de la consulta. PAra hacer el snippet rapido se necesita el indice directo, ya que se debe hacer online.

Scorer

El scorer solo tira contra el indice . Los docuemntos que no tienen ninguna palabra de la query no entran en el juego de aparecer en el ranking (en los metodos clasicos, los neuronales son otro rollo).

Para hacer este ranking, una forma es term ar a time. Si la query es a, b, c, primero hago a luego b y luego c. Cualquier formula de scoring clasica de las vistas, se tiene un sumatorio de los matching terms y dentro un peso que puede variar de mas a menos sofisticado. Sumar el score de a, el de b y el de c, para cada w en la query. Pregunto al indice los documentos que tiene con w, y devuelve las entradas de la posting list para w, en que documentos esta y cuantas veces en cada uno. Se mira el score de cada documento por tener esa palabra y se obtiene el score parcial de los documentos en cuanto a su contribucion al score parcial de una parte de la query (acumuladores). Al final se tiene un score de cada documento y se ordena en un top k. Esto tiene el problema de que hay muchos documentos que entren a puntuar , y se deben guardar hasta el final.

APra paliar esto de tiene DOC AT A time

Cada vez que encuentro un documento, se calcula el score total de ese documento. Esto permite mantener guardados solo los k mejores acumuladores y uno nuevo solo entra si es mejor que el peor de los k.

FILTERING DOCUMETNS

Solo documentos que cumplan un criterio un filtrado booleano

INDEX SHARDING

En realidad el indice esta troceado en varias maquina.

CACHE

Pueden servir para muchas cosas. En la realidad, los buscadores web personalizan los rankings en funcion del usuario, por lo que en sistemas de gran personalizacion es dificil reusar una query anterior, aunque sea de hace 2 segundos.

La cache no puede ser infinita y para manejar a quien se echa de la cache se usan algoritmos tipo menor recientemente usado (LRU). En un caso ideal se puede reusar toda la query. A veces preguntan algo parecido, por lo que si me quee la posting list, me ahorro buscar en disco y la descompresión de los terminos de antes.

La mayoria de las queries son unicas y algunas se repiten mucho, lo que hace que guardarlas en cache sea recomendable (Zipf law)

TEMA 6 - Evaluacion de sistemas de busqueda

Tradicionalmente se viene haciendo de varias formas con medidas que reflejan la utilidad de personas reales en la aplicacion. PAra ello se pueden hacer estudios de usuario (user studies).

QUE VAMOS A MEDIR

ALgo fundamental para estos sistemas, que da como de relevante son los resultados para la busqueda. REsume un bechmark de retrieval (CRANFIEL EVALUATIOM) debe tener un banco de documentos sobre el que buscar, tener casos de prueba y, el cuello de botella, hay que saber cuales son los documentos relevantes del corpus para cada caso de busqueda. Es el cuello de botella para construir el benchmark porque es facil conseguir documentos y consultas, haciendo crawling en la web, pero es dificil saber cuales son relevantes.

Una vez que tenemos los tres ingredientes, necesitamos metricas que cuantfiquen como de bien el sistema responde a esas busquedas. eSto configura una comparación justa. Construir el bencharm puede ser costoso, pero una vez hecho, se puede usar indefinidamente (reusabilidad).

FIgure 9.1. Imaginemos una query Q1. Para un sistema tipo web, que los usuarios no quieren ver muchos documents, queremos un sistema preciso. Sin embargo, para un sistema de busqueda de patentes por ejemplo, queremos ver muchos de los relevantes.

SET RETRIEVAL EVALUTATION

Metricas basadas en conjuntos (set based): la primera de ellas es la precision, cuantos con rlevantes de los que me das? La segunda es el recall: de los relevantes en el corpus para esa consulta, cuantos me has dado? Un recall perfecto es trivial y no inteeresa, ya que devuelve gran cantidad de documentos.

Estas dos metricas estas en tension la una con la otra. Gran precisio bajo recall. Si optimizo recall caera la precision. ESto era valido antes, pero actualmetne es complicado conocer el denominador. Las busquedas web necesitan precision at k (Pk), que es la precision en los primeros k documentos (este no lo usan google y tal).

Medida F es parametrizada por un parametro beta, es una agregacion de preciison y recal. Se suele combinar con la media armonica de ambas (beta=1). Intenta tener ambas notables en vez de una muy buena y otr mala.

EVALUATION OF A RANDER list

No tiene que ser monotono decreciente. De sistema A y B, el mejor de la derecha apra buscadores es B porque nos interesa la zona de bajo recall

AP

(1/1 + 2/2 + 3/5 + 4/8)

Si la ubicacion de los buenos encontrados varia, el AP varia. El unico AP perfecto es el que tenga todos los relevantes primero, queremos que penalice si los relevantes van hacia abajo.

MAP

Media de los AP de cada query, con m todas las queries que tenga en el benchmark.

KNOW ITEM SEARCH - buscamos por una pagina que ya sabemos cual es pero no queremos guardarla. Me pone esa pagina en la que ya estuve lo mas arriba posible

Otra metrica de rendimiento. MRR: casos de busqueda donde solo importa la posicion del primer relevante. Si esta el primero, 1, si esta el segundo 1/2, si esta el k, 1/k.

NDCG - Normalized Discounted Cumulative Gain

La que optimizan continuamente google y bing. Esta medida no tiene la opcion binaria de relevancia de las anteriores, permite opciones no binarias de relvancia. Acumula ganancias de arriba a abajo en funcion de los que se van dando.

No debe ser ganancia lineal, la ganancia de un 3 por ejemplo en la posicion 1 y otro en la 10 no es la misma, costo mas esfuerzo llegar a la 10. Un tipico discount que se aplica es uno logaritmico. Porque queremos algoritmos orientados a tener los muy relevantes arriba. Esto se calcula en un determinado corte (at k). A google o bing le interesa un NDCG@10, que son los que se muestran en la primera pagina. Para darle un significado, se normaliza dividiendo por el DCG ideal, es decir, el top k ideal de la query. El ideal en la web en principio no se sabe

Los documentos deben ser representativos del corpus sobre el que busca la gente, así como las queries deben ser realistas respecto a las que hara la gente. Querriamos juiciios completos de relevancias, pero es inviable. Queremos medida que tengan asociacion con la utilidad de los usuarios. No se puede usar un P 5 para la web, porque cualquier permutacion de esos dara el mismo valor.

STATISTICAL SIGNIFICANCE TESTS

Supongamos que tenemos una variante A y otra B, con NDCG10 son 0.75 y 0.77. ¿Es significativo? No, porque vemos en la tabla por ejemplo que al ser una media, puede haber fluctuaciones muy grandes. Queremos que haya mejora estadistica transversal (para todos los casos). Para eso hay que hacer test de significacion estadistica para ver si uno es consistentemente mejor que otro. No nos metemos en cuales se suelen aplicar. Nos miden si hay consistencia estadistica en la diferencia de medias.

BUILDING INCOMPLETE relevance

Es imposible etiquetar un corpus grande hoy en dia. Los juicios de relevancia deben ser necesariamente incompletos. Hay que hacer un pool, coger una muestra de la web y etiquetarla. Si sacamos un numero aleatorio de TODA la web, seran todos irrelevantes, por lo que no srive. Sesgamos la muestra de lo que etiquetamos a documentos que potencialmente sean relevantes.

Ver ejemplo obama. Hago N sistemas de busqueda, cuanto mas variados sean mejor. CAda uno recibe la query del benchmark y hace un ranking. Cortamos estos rankings a una profundidad dada. Habra documentos en comun en los rankings. Todos estos los uno en un pool y los etiqueto usando humanos. Esto funciona bien si hay diversidad en los sistemas de busqueda. Se pierden relevantes porque habra documentos que no salgan en ningun ranking. Con esto ya se hace el NDCG.

LEER EL LIBRO LOS CAPS QUE DICE EN EL CAMPUS

6 Representaciones avanzadas de texto

HAce 10 años empezo una revolucion en el campo de la representacion de texto con Word2vec. Pasamos de representaciones sprse a representaciones densas de mebeddngs que capturan mejor la semantica y significado del texto.

CHATGPT tiene un impacto social enorme

Esta revolucion que cambia y por que cambia. Lo tradicional en claisifcacion automatica, el paradigma, se tiene un traing data etiqueta, se lo doy al sistema aprende y puedo poner en produccion. El cuello de botella es que necesito etiquetas en el training data. Da capacidades limitadas de compresion de lenguaje, ya que para ello tendria que meter gran cantidad de formas de decir lo mismo, incluir toda la casuistica.

Primera vuelta de tuerca, preentrenar y finetune. LOs modelos se preentrenan: cogemos una gran cantidad de datos (no toda la web pero casi) y jugando al juego de ocultar palabras (haciendo masking) y predecir la palabra oculta. Si lo dice mal, la penalizo. Si lo hace bien, consigo un modelo que entiende bien el lenguaje, la semantica, otros tipos de conocimientos sobre el mundo. Esto es autoentrenamiento sin etiquetado.

Una vez entrenado, lo ajusto y personalizo para mi tarea con el fine tune (adaptation).

La idea clave de los embbeding es construir automaticamente un representacion de las palabras donde las palabras que son similares conceptualemente estan cerca en el espacio vectorial. Los alrededores de una palabra (izquierda y derecha) son los que determinan el significado de la palabra (ejemplo de tezgüino). Las palabras parecidad van a tener contextos izquierdo y derecho parecidos. Esta es la base teórica para la integración semántica de la representación de las palabras.

Un word embedding es una lista, un vector denso de longitud fija. PAra construirlas se puede hacer de muchas formas: por ejemplo, estadísticas de coocurrencia. Esto es un espacio de bajas dimensiones. Muchas veces se aprenden con redes de neuronas.

Para que sirve esto? si yo tengo los embeddings de todas las palabras se pueden buscar las palabras cercanas en el espacio embebido para buscar sinonimos, por ejemplo.

Una estrategia posible es que los embeddings sea la entrada a una red de neuronas para entrenamiento supervisada. Tambien podemos representar el corpus con los embeddings y ver los posibles clusters.

Los vectores de las palabras admiten operaciones !

Los redes de neuronas. Se coge una red de neuronas y la hago jugar a un juego predictivo de ir escondiendo palabras y que vaya adivinando. Aqui el proceso de backpropagation juega un papel fundamental. Queremos que palabras que significan lo mismo vayan a la misma representación. Otras alternativas son mas algebraicas, usando cuentas de coocurrencia

Predecir la palabra j-esima dadas las anteriores (tipo gpt). Una red de neuronas que hago esto bien es util. En el ambito de los LLM da un gran conocimiento del mundo, tanto semantico como sintactico y gramatical.

estos ebeddings no es que se produjera la red de neuronas para darlos, se encontro que era un subproducto de la red de neuronas.

VEamos varios ejemplos

NNLM: Contruyo una red de neuronas para predecir una palabra i esima dada las anteriores y el output layer es la probabilidad de cada palabra en el vocabulario dadas las anteriores. Internamente representa los embeddings del texto. Los mebeddigs son los pesos de la capa de entrada a la capa oculta. (la e simboliza embedding) x coge la representacion de las embeddings, lo mete en una capa interna (h).

WORD2VEC. Embeddings independietnes del contexto, es decir, palabras polisemicas tienen un solo embedding para todos los significados, aquí estas palabras sufren. Esto lo hace leyendo un corpus masivo y de ese (auto)aprendizaje, porque no etiquetamos nada, aprende las palabras. Una forma de predecir es CBOW , predecir una palabra dando las anteriores y posteriores.... Da una representacion distribuida. DEF.

BERT (la B de Bert es de bidireccional). Embeddings dependiente del contexto, para palabras polisemicas, da un embedding diferente a cada significado.

BERT solo hace encoding (hace todo lo necesario para crear una representación interna del texto), GPT solo decoding (me pasas la representacion al lenguaje que te pido) y otros tienen ambas, por ejemplo T5. CHATGPT ademas esta tuneado para seguir instrucciones; luego se adapta a conversar y genera una representacion de la isntruccion y producirla en el lenguaje que se pida.

T5: sequence-to-sequence. en funcion e cierto patron de entrada produce cierto aprametro de salida. T5 se utilizan para reranking (ranking hecho con un BM25 o algo clasico), pero con T5 es muy costoso. El T5 aprende de un proceso de enmascaramiento (preeentraenamiento). Mono T5 esta finetuneado con MSMarco, coleccion de pasajes que puso bing.

count-based models. LSA No basados en redes neuronales pero igualmente son capaces de generar representaciones densas. La idea del LSA: sabemos el problema de las representaciones sparse (dos sinonimos no guardan relacion) idealmente no querria indexar por palabras, sino por conceptos. El problema es que los conceptos no son explicitos. LSA usa la matriz originales y transformarla a un espacio de conceptos usando una estrategia de coocurrencia cpn metodos algebraicos.

HAL: busca todos los contextos (ventanas de aparicion de las palabras en los corpus) y analiza la cuenta de coocurrencia entre esa palabra y los contextos.

COAL: poco detalle

LR-MVL: un poco mas moderno. Si tiene en cuenta contexto izquierdo y derecho.

GLOVE: mejora sobre estrategias como Word2Vec (problema mira las apariciones de las palabras en el corpus, no usa informacion global). GLOVE tiene en cuenta informacion de estadísticas globales, no en las coocurrencias individuales sino globales.

FINAL REMARKS. CHALLENGES.

OOV- muchos modelos de embeddings no gestionan palabras no vistas anteriormente.

7 Web search

Los motores de búsqueda web son una de las aplicaciones más importantes de la recuperación de texto. Aunque muchos algoritmos de recuperación de información se desarrollaron antes del nacimiento de la web, esta creó la mejor oportunidad para aplicar esos algoritmos a un problema de aplicación importante que a todos les importa. Naturalmente, hubo que hacer algunas extensiones adicionales de los algoritmos de búsqueda clásicos para abordar completamente los nuevos desafíos encontrados en la búsqueda web.

Primero, este es un desafío de escalabilidad. ¿Cómo podemos manejar el tamaño de la web y asegurar la cobertura completa de toda su información (ya sea textual o no)? ¿Cómo podemos servir a muchos usuarios rápidamente respondiendo todas sus consultas? Antes del nacimiento de la web, la escala de búsqueda era relativamente pequeña, generalmente enfocada en bibliotecas, por lo que estas preguntas no eran serias.

El segundo problema es que hay mucha información de baja calidad conocida como spam. La optimización de motores de búsqueda es el intento de aumentar el rango de una página en particular aprovechando cómo se puntúan las páginas, por ejemplo, agregando muchas palabras que no son necesariamente relevantes para el contenido real o creando muchos enlaces falsos a una página en particular para hacer que parezca más popular de lo que realmente es. Se han diseñado muchos enfoques diferentes para detectar y prevenir tales prácticas de spam [Spirin y Han 2012].

El tercer desafío es la naturaleza dinámica de la web. Se crean y actualizan nuevas páginas muy rápidamente. Esto hace que sea más difícil mantener el índice fresco con el contenido más reciente. Estos son solo algunos de los desafíos que tenemos que resolver para construir un motor de búsqueda web de alta calidad. A pesar de estos desafíos, también hay algunas oportunidades interesantes que podemos aprovechar para mejorar los resultados de búsqueda. Por ejemplo, podemos imaginar que usar enlaces entre páginas puede mejorar la puntuación.

Los algoritmos de los que hablamos, como el modelo de espacio vectorial, son generales: pueden aplicarse a cualquier aplicación de búsqueda. Por otro lado, tampoco aprovechan las características especiales de las páginas o documentos en aplicaciones específicas como la búsqueda web. Debido a estos desafíos y oportunidades, hay nuevas técnicas que se han desarrollado específicamente para la búsqueda web. Una de estas técnicas es la indexación y búsqueda paralela. Esto aborda el problema de la escalabilidad; en particular, el marco MapReduce de Google es muy influyente.

También hay técnicas que se han desarrollado para abordar el problema del spam. Tendremos que evitar que esas páginas de spam sean clasificadas en los primeros lugares. También hay técnicas para lograr una clasificación robusta a la luz de los optimizadores de motores de búsqueda. Vamos a usar una amplia variedad de señales para clasificar las páginas de modo que no sea fácil engañar al motor de búsqueda con un truco en particular.

La tercera línea de técnicas es el análisis de enlaces; estas son técnicas que pueden permitirnos mejorar los resultados de búsqueda aprovechando información adicional sobre la naturaleza en red de la web. Por supuesto, utilizaremos múltiples características para la clasificación, no solo el análisis de enlaces. También podemos explotar todo tipo de características como el diseño de las páginas web o el texto ancla que describe un enlace a otra página.

El primer componente de un motor de búsqueda web es el rastreador. Este es un programa que descarga el contenido de las páginas web que deseamos buscar. El segundo componente es el indexador, que tomará estas páginas descargadas y creará un índice invertido. El tercer componente es

la recuperación, que responde a la consulta de un usuario hablando con el navegador del usuario. El navegador mostrará los resultados de la búsqueda y permitirá al usuario interactuar con la web. Estas interacciones con el usuario permiten oportunidades para retroalimentación (discutido en el Capítulo 7) y evaluación (discutido en el Capítulo 9). En la siguiente sección, discutiremos el rastreo. Ya hemos descrito todos los pasos de indexación excepto el rastreo en detalle en el Capítulo 8.

Después de nuestra discusión sobre el rastreo, pasamos a los desafíos particulares de la indexación web. Luego, discutimos cómo podemos aprovechar los enlaces entre páginas en el análisis de enlaces. La última técnica que discutimos es el aprendizaje para clasificar, que es una forma de combinar muchas características diferentes para la clasificación.

7.1 Web Crawling

Ejemplo: la USC le pone delay a las peticiones de google

El rastreador también se llama araña o robot de software que rastrea (recorre, analiza y descarga) páginas en la web. Construir un rastreador de juguete es relativamente fácil porque solo necesitas comenzar con un conjunto de páginas semilla, obtener páginas de la web y analizar los nuevos enlaces de estas páginas. Luego los agregamos a una cola y luego exploramos los enlaces de esas páginas en una búsqueda en anchura hasta que estemos satisfechos.

Construir un rastreador real es bastante complicado y hay algunos problemas complejos con los que inevitablemente tenemos que lidiar. Un problema es la robustez: ¿Qué pasa si el servidor no responde o devuelve basura que no se puede analizar? ¿Qué pasa si hay una trampa que genera páginas dinámicamente que atraen a tu rastreador a seguir rastreando el mismo sitio en círculos? Otro problema es que no queremos sobrecargar un servidor en particular con demasiadas solicitudes de rastreo. Eso puede causar que el sitio experimente una denegación de servicio; algunos sitios también bloquearán direcciones IP que creen que los están rastreando o creando demasiadas solicitudes. De manera similar, un rastreador debe respetar el protocolo de exclusión de robots. Un archivo llamado robots.txt en la raíz del sitio le dice a los rastreadores qué rutas no tienen permitido rastrear. También necesitas manejar diferentes tipos de archivos como imágenes, PDFs o cualquier otro tipo de formatos en la web que contengan información útil para tu aplicación de búsqueda. Idealmente, el rastreador debería reconocer páginas duplicadas para no repetirse o quedar atrapado en un bucle. Finalmente, puede ser útil descubrir URLs ocultas; estas son URLs que pueden no estar vinculadas desde ninguna página pero que aún contienen contenido que te gustaría indexar.

Entonces, ¿cuáles son las principales estrategias de rastreo? En general, la búsqueda en anchura es la más común porque equilibra naturalmente la carga del servidor. El rastreo paralelo también es muy natural porque esta tarea es muy fácil de paralelizar. Una variación interesante se llama rastreo enfocado. Aquí, vamos a rastrear algunas páginas sobre un tema en particular, por ejemplo, todas las páginas sobre automóviles. Esto generalmente comenzará con una consulta que usas para obtener algunos resultados. Luego, rastreas gradualmente más. Una versión aún más extrema del rastreo enfocado es (por ejemplo) descargar e indexar todas las publicaciones de un foro en particular. En este caso, podríamos tener una URL como http://www.text-data-book-forum.com/boards?id=3 que se refiere a la tercera publicación en el foro. Al cambiar el parámetro id, podemos iterar a través de todas las publicaciones del foro e indexarlas bastante fácilmente. En este escenario, es especialmente importante agregar un retraso entre las solicitudes para que el servidor no se vea abrumado.

Otro desafío en el rastreo es encontrar nuevas páginas que se hayan creado desde la última vez que se ejecutó el rastreador. Esto es muy desafiante si las nuevas páginas no han sido vinculadas a ninguna página antigua. Si lo están, entonces probablemente puedas encontrarlas siguiendo los enlaces desde las páginas existentes en tu índice.

Finalmente, podríamos enfrentar el escenario de rastreo incremental o rastreo repetido. Supongamos que quieres poder crear un motor de búsqueda web. Claramente, primero rastreas datos de la web.

En el futuro, solo necesitamos rastrear las páginas actualizadas. Esta es una pregunta de investigación muy interesante: ¿cómo podemos determinar cuándo una página necesita ser rastreada nuevamente (o incluso cuándo se ha creado una nueva página)? Hay dos factores principales a considerar aquí, el primero de los cuales es si una página en particular se actualizaría con frecuencia. Si la página es una página estática que no ha cambiado durante meses, probablemente no sea necesario volver a rastrearla todos los días, ya que es poco probable que cambie con frecuencia. Por otro lado, si es (por ejemplo) una página de resultados deportivos que se actualiza con mucha frecuencia, es posible que necesites volver a rastrearla incluso varias veces en el mismo día. El segundo factor a considerar es con qué frecuencia una página en particular es accedida por los usuarios del sistema de motores de búsqueda. Si es una página de alta utilidad, es más importante asegurarse de que esté actualizada. Compáralo con otra página que nunca ha sido buscada por ningún usuario durante un año; aunque esa página impopular haya cambiado mucho, probablemente no sea necesario rastrear esa página, o al menos no es tan urgente, para mantener su frescura.

7.2 Web Indexing

En esta sección, discutiremos cómo crear un índice a escala web. Después de que nuestro rastreador entregue gigabytes o terabytes de datos, el siguiente paso es usar el indexador para crear el índice invertido. En general, podemos usar las técnicas estándar de recuperación de información para crear el índice, pero hay nuevos desafíos que tenemos que resolver para la indexación a escala web. Los dos principales desafíos son la escalabilidad y la eficiencia.

El índice será tan grande que no puede caber en una sola máquina o en un solo disco, por lo que tenemos que almacenar los datos en múltiples máquinas. Además, debido a que los datos son tan grandes, es beneficioso procesar los datos en paralelo para que podamos producir el índice rápidamente. Para abordar estos desafíos, Google ha realizado una serie de innovaciones. Una es el Sistema de Archivos de Google (GFS), que es un sistema de archivos distribuido general que puede ayudar a los programadores a gestionar archivos almacenados en un clúster de máquinas. La segunda es MapReduce, que es un marco de software general para soportar la computación paralela. Hadoop es la implementación de código abierto más conocida de MapReduce, ahora utilizada en muchas aplicaciones.

La Figura 10.1 muestra la arquitectura del Sistema de Archivos de Google (GFS). Utiliza un mecanismo de gestión centralizado muy simple para gestionar todas las ubicaciones específicas de los archivos. Es decir, mantiene un espacio de nombres de archivos y una tabla de búsqueda para saber exactamente dónde se almacena cada archivo. El cliente de la aplicación habla con el nodo maestro de GFS, que obtiene ubicaciones específicas de los archivos para procesar. Este sistema de archivos almacena sus archivos en máquinas en fragmentos de tamaño fijo; cada archivo de datos se separa en muchos fragmentos de 64 MB. Estos fragmentos se replican para garantizar la fiabilidad. Todos estos detalles son algo de lo que el programador no tiene que preocuparse, y todo está a cargo de este sistema de archivos. Desde la perspectiva de la aplicación, el programador vería un archivo normal. El programa no tiene que saber exactamente dónde está almacenado, y puede simplemente invocar operadores de alto nivel para procesar el archivo. Otra característica es que la transferencia de datos es directamente entre la aplicación y los servidores de fragmentos, por lo que también es eficiente en este sentido.

Encima del GFS, Google propuso MapReduce como un marco general para la programación paralela. Esto soporta tareas como la construcción de un índice invertido. Al igual que GFS, este marco oculta características de bajo nivel del programador. Como resultado, el programador puede hacer un esfuerzo mínimo para crear una aplicación que se pueda ejecutar en un gran clúster en paralelo. Algunos de los detalles de bajo nivel ocultos en el marco son las comunicaciones, el balanceo de carga y la ejecución de tareas. La tolerancia a fallos también está incorporada; si un servidor se cae,

algunas tareas pueden no completarse. Aquí, el mecanismo de MapReduce sabría que la tarea no se ha completado y automáticamente asignaría la tarea a otros servidores que puedan hacer el trabajo. Nuevamente, el programador no tiene que preocuparse por esto.

En MapReduce, los datos de entrada se separan en varios pares (clave, valor). Lo que exactamente es el valor dependerá de los datos. Cada par se enviará a una función de mapa que escribe el programador. La función de mapa procesará estos pares (clave, valor) y generará varios otros pares (clave, valor). Por supuesto, la nueva clave suele ser diferente de la clave antigua que se da al mapa como entrada. Todas las salidas de todas las llamadas al mapa se recopilan y ordenan en función de la clave. El resultado es que todos los valores que están asociados con la misma clave se agruparán. Para cada clave única, ahora tenemos un conjunto de valores que están adjuntos a esta clave. Estos son los datos que se envían a la función de reducción. Cada instancia de reducción manejará una clave diferente. Esta función procesa su entrada, que es una clave y un conjunto de valores, para producir otro conjunto de pares (clave, valor) como salida. Este es el marco general de MapReduce. Ahora, el programador solo necesita escribir la función de mapa y la función de reducción. Todo lo demás está a cargo del marco MapReduce. Con un marco así, los datos de entrada se pueden dividir en múltiples partes que se procesan en paralelo primero por el mapa, y luego se procesan nuevamente en paralelo una vez que llegamos a la etapa de reducción.

La Figura 10.3 muestra un ejemplo de conteo de palabras. La entrada son archivos que contienen palabras tokenizadas y la salida que queremos generar es el número de ocurrencias de cada palabra. Este tipo de conteo sería útil para evaluar la popularidad de una palabra en una gran colección o lograr un efecto de ponderación IDF para la búsqueda. Entonces, ¿cómo podemos resolver este problema? Un pensamiento natural es que esta tarea se puede hacer en paralelo simplemente contando diferentes partes del archivo en paralelo y combinando todos los conteos. Esa es precisamente la idea de lo que podemos hacer con MapReduce: podemos paralelizar líneas en este archivo de entrada. Más específicamente, podemos asumir que la entrada a cada función de mapa es un par (clave, valor) que representa el número de línea y la cadena en esa línea.

La primera línea es el par (1, HelloWorldByeWorld). Este par se enviará a una función de mapa que cuenta las palabras en esta línea. En este caso, solo hay cuatro palabras y cada palabra obtiene un conteo de uno. El pseudocódigo del mapa mostrado en la parte inferior de la figura es bastante simple. Simplemente necesita iterar sobre todas las palabras en esta línea, y luego simplemente llamar a una función Collect, lo que significa que luego enviaría la palabra y el contador al colector. El colector intentaría ordenar todos estos pares clave-valor de diferentes funciones de mapa. El programador especifica esta función como una forma de procesar cada parte de los datos. Por supuesto, la segunda línea será manejada por una instancia diferente de la función de mapa, que producirá una salida similar. Como se mencionó, el colector hará el agrupamiento o clasificación interna. En esta etapa, puedes ver que hemos recopilado múltiples pares. Cada par es una palabra y su conteo en la línea. Una vez que vemos todos estos pares, podemos ordenarlos en función de la clave, que es la palabra. Cada palabra ahora está adjunta a varios valores, es decir, varios conteos. Estos nuevos pares (clave, valor) se alimentarán a una función de reducción.

La Figura 10.4 muestra cómo la función de reducción termina el trabajo de contar las ocurrencias totales de esta palabra. Ya tiene estos conteos parciales, por lo que todo lo que necesita hacer es simplemente sumarlos. Tenemos un contador y luego iteramos sobre todas las palabras que vemos en esta matriz, como se muestra en el pseudocódigo en la parte inferior de la figura. Finalmente, salimos la clave y el conteo total, que es precisamente lo que queremos como salida de todo este programa. Como podemos ver, esto ya es muy similar a la construcción de un índice invertido; la salida aquí está indexada por una palabra, y tenemos un diccionario del vocabulario. Lo que falta son los IDs de los documentos y los conteos de frecuencia específicos de las palabras en cada documento en particular. Podemos modificar esto ligeramente para construir realmente un índice invertido en paralelo.

Modifiquemos nuestro ejemplo de conteo de palabras para crear un índice invertido. La Figura

10.5 ilustra este ejemplo. Ahora, asumimos que la entrada a la función de mapa es un par (clave, valor) donde la clave es un ID de documento y el valor denota el contenido de la cadena de todas las palabras en ese documento. La función de mapa hará algo muy similar a lo que hemos visto en el ejemplo anterior: simplemente agrupa todos los conteos de esta palabra en este documento, generando nuevos pares. En los nuevos pares, cada clave es una palabra y el valor es el conteo de esta palabra en este documento seguido del ID del documento. Más tarde, en el índice invertido, nos gustaría mantener esta información del ID del documento, por lo que la función de mapa realiza un seguimiento de ella.

Después de la función de mapa, hay un mecanismo de clasificación que agruparía las mismas palabras y alimentaría estos datos a la función de reducción. Vemos que la entrada de la función de reducción se parece a una entrada de índice invertido. Es solo la palabra y todos los documentos que contienen la palabra y la frecuencia de la palabra en esos documentos. Todo lo que necesitamos hacer es simplemente concatenarlos en un bloque continuo de datos, y esto se puede almacenar en el sistema de archivos. La función de reducción hará un trabajo muy mínimo. El Algoritmo 10.1 se puede usar para esta construcción de índice invertido.

El Algoritmo 10.1, adaptado de Lin y Dyer [2010], describe las funciones de mapa y reducción. Un programador especificaría estas dos funciones para ejecutarse en la parte superior de un clúster de MapReduce. Como se describió antes, el mapa cuenta las ocurrencias de una palabra usando un array asociativo (diccionario), y sale todos los conteos junto con el ID del documento. La función de reducción simplemente concatena toda la entrada que se le ha dado y como una sola entrada para esta clave de ID de documento. A pesar de su simplicidad, esta función de MapReduce nos permite construir un índice invertido a una escala muy grande. Los datos pueden ser procesados por diferentes máquinas y el programador no tiene que preocuparse por los detalles. Así es como podemos hacer la construcción de índices en paralelo para la búsqueda web.

Para resumir, la indexación a escala web requiere algunas nuevas técnicas que van más allá de las técnicas de indexación tradicionales estándar. Principalmente, tenemos que almacenar el índice en múltiples máquinas, y esto generalmente se hace utilizando un sistema de archivos distribuido como el GFS. En segundo lugar, requiere crear el índice en paralelo porque es muy grande. Esto se hace utilizando el marco MapReduce. Es importante tener en cuenta que tanto el GFS como el marco MapReduce son muy generales, por lo que también pueden soportar muchas otras aplicaciones además de la indexación.

7.3 Link Analysis

Si A referencia a B, y A es bueno, suponemos que B es bueno. El texto del enlace que pone A para B (anchor text) es muy importante (page rank): es lo que dicen otros, en resumen, de B.

En esta sección, vamos a continuar nuestra discusión sobre la búsqueda web, particularmente enfocándonos en cómo utilizar los enlaces entre páginas para mejorar la búsqueda. En la sección anterior, hablamos sobre cómo crear un índice.

Primero, en la web tendemos a tener necesidades de información muy diferentes. Por ejemplo, las personas pueden buscar una página web o una página de entrada, lo cual es diferente de la búsqueda tradicional en bibliotecas donde las personas están principalmente interesadas en recopilar información literaria. Este tipo de consultas a menudo se llaman consultas de navegación, donde el propósito es navegar hacia una página específica. Para tales consultas, podríamos beneficiarnos del uso de información de enlaces. Por ejemplo, las consultas de navegación podrían ser facebook o yahoo finance. El usuario simplemente está tratando de llegar a esas páginas sin escribir explícitamente la URL en la barra de direcciones del navegador.

En segundo lugar, los documentos web tienen mucha más información que el texto puro; hay organización jerárquica y anotaciones como el diseño de la página, el título o los hipervínculos a otras páginas. Estas características proporcionan una oportunidad para usar información de contexto

adicional del documento para mejorar la puntuación. Finalmente, la calidad de la información varía mucho. Todo esto significa que debemos considerar muchos factores para mejorar el algoritmo de clasificación estándar, dándonos una forma más robusta de clasificar las páginas y haciendo más difícil para los spammers manipular una señal para mejorar la clasificación de una sola página.

Como resultado de todas estas preocupaciones, los investigadores han realizado una serie de extensiones importantes a los algoritmos de clasificación estándar. Una es explotar los enlaces para mejorar la puntuación, que es el tema principal de esta sección. También hay algoritmos para explotar la información de retroalimentación implícita a gran escala en forma de clics. Por supuesto, eso pertenece a la categoría de técnicas de retroalimentación, y las técnicas de aprendizaje automático se utilizan a menudo allí. En general, los algoritmos de clasificación de búsqueda web se basan en algoritmos de aprendizaje automático para combinar todo tipo de características. Muchos de ellos se basan en modelos estándar como BM25 que discutimos en el Capítulo 6. La información de enlaces es una de las características importantes utilizadas en las funciones de puntuación combinadas en los sistemas modernos de búsqueda web.

La Figura 10.6 muestra una instantánea de una parte de la web. Podemos ver que hay muchos enlaces que conectan diferentes páginas, y en el centro, hay una descripción de un enlace que apunta al documento en el lado derecho. Este texto de descripción se llama texto ancla. En realidad, es increíblemente útil para los motores de búsqueda porque proporciona una descripción adicional de la página a la que se apunta. Por ejemplo, si alguien quiere marcar la página principal de Amazon.com, la persona podría crear un enlace llamado la gran librería en línea que apunta a Amazon. La descripción es muy similar a lo que el usuario escribiría en el cuadro de búsqueda cuando busca una página así. Supongamos que alguien escribe una consulta como librería en línea o gran librería en línea. La consulta coincidiría con este texto ancla en la página. Esto en realidad proporciona evidencia para coincidir con la página a la que se ha apuntado: la página de entrada de Amazon. Por lo tanto, si coincides con el texto ancla que describe el enlace a una página, proporciona una buena evidencia de la relevancia de la página a la que se apunta.

En la parte inferior de la Figura 10.6, hay algunos patrones de enlaces que pueden indicar la utilidad de un documento. Por ejemplo, en el lado derecho puedes ver que una página ha recibido muchos enlaces entrantes, lo que significa que muchas otras páginas están apuntando a esta página. Esto muestra que esta página es bastante útil. En el lado izquierdo puedes ver una página que apunta a muchas otras páginas. Esta es una página central que te permitiría ver muchas otras páginas. Llamamos al primer caso una página de autoridad y al segundo caso una página de concentrador. Esto significa que la información de enlaces puede ayudar de dos maneras; una es proporcionar texto adicional para la coincidencia (en el caso de los anclajes) y la otra es proporcionar algunas puntuaciones adicionales para las páginas web para caracterizar la probabilidad de que una página sea un concentrador o una autoridad.

7.3.1 PageRank

PageRank de Google, una técnica principal que se utilizó originalmente para el análisis de enlaces, es un buen ejemplo de cómo aprovechar la información de enlaces de las páginas. PageRank captura la popularidad de la página, que es otra palabra para autoridad. La intuición es que los enlaces son como citas en la literatura. Piensa en una página que apunta a otra página; esto es muy similar a un artículo que cita a otro artículo. Por lo tanto, si una página es citada con frecuencia, podemos asumir que esta página es más útil. PageRank aprovecha esta intuición y la implementa de manera metódica. En su sentido más simple, PageRank es esencialmente contar citas o contar enlaces entrantes.

Mejora esta idea simple de dos maneras. Una es considerar las citas indirectas. Esto significa que no solo miras el número de enlaces entrantes, sino que también miras los enlaces entrantes de tus enlaces entrantes, de manera recursiva. Si tus enlaces entrantes tienen muchos enlaces entrantes, tu página obtiene crédito por eso. En resumen, si las páginas importantes te están apuntando, tú también

debes ser importante. Por otro lado, si las páginas que te están apuntando no son apuntadas por muchas otras páginas, entonces no obtienes tanto crédito. Este es el concepto de citas indirectas, o citas en cascada.

Nuevamente, podemos entender esta idea considerando artículos de investigación. Si eres citado por diez artículos que no son muy influyentes, eso no es tan bueno como si fueras citado por diez artículos que ellos mismos han atraído muchas otras citas. Claramente, este es un caso en el que nos gustaría considerar enlaces indirectos, que es exactamente lo que hace PageRank. La otra idea es que es bueno suavizar las citas para acomodar posibles citas que aún no se han observado. Supongamos que cada página tiene un conteo de citas pseudo no nulo. Esencialmente, estás tratando de imaginar que hay muchos enlaces virtuales que enlazarán todas las páginas juntas para que realmente obtengas citas pseudo de todos.

Otra forma de entender PageRank es el concepto de un navegante aleatorio visitando cada página web. Veamos este ejemplo en detalle, ilustrado en la Figura 10.7. A la izquierda, hay un pequeño gráfico, donde cada documento d1, d2, d3 y d4 es una página web, y los bordes entre documentos son hipervínculos que los conectan entre sí. Supongamos que un navegante aleatorio o caminante aleatorio puede estar en cualquiera de estas páginas. Cuando el navegante aleatorio decide moverse a una página diferente, puede seguir aleatoriamente un enlace desde la página actual o elegir aleatoriamente un documento para saltar desde toda la colección. Entonces, si el navegante aleatorio está en d1, con cierta probabilidad ese navegante aleatorio seguirá los enlaces a d3 o d4. El modelo de navegación aleatoria también asume que el navegante podría aburrirse a veces y decidir ignorar los enlaces reales, saltando aleatoriamente a cualquier página en la web. Si el navegante toma esa opción, podría llegar a cualquiera de las otras páginas aunque no haya un enlace directo a esa página. Basado en este modelo, podemos hacer la pregunta: "¿Qué tan probable es, en promedio, que el navegante llegue a una página en particular?" Esta probabilidad es precisamente lo que calcula PageRank.

La puntuación de PageRank de un documento di es la probabilidad promedio de que el navegante visite di. Intuitivamente, esto debería ser proporcional al conteo de enlaces entrantes. Si una página tiene un alto número de enlaces entrantes, entonces tendría una mayor probabilidad de ser visitada, ya que habrá más oportunidades de que el navegante siga un enlace allí. Así es como el modelo de navegación aleatoria captura la idea de contar los enlaces entrantes. Pero, también considera los enlaces entrantes indirectos; si las páginas que apuntan a di tienen muchos enlaces entrantes, eso significaría que el navegante aleatorio probablemente llegaría a una de ellas. Esto aumenta la probabilidad de visitar di. Esta es una buena manera de capturar tanto los enlaces directos como los indirectos.

Matemáticamente, podemos representar esta red de documentos como una matriz M, mostrada en el centro de la Figura 10.7. Cada fila representa una página de inicio. Por ejemplo, la primera fila indicaría la probabilidad de ir a cualquiera de las cuatro páginas desde d1. Vemos que solo hay dos entradas no nulas. Cada una es la mitad, ya que d1 apunta solo a otras dos páginas; por lo tanto, si podemos elegir aleatoriamente visitar cualquiera de ellas desde d1, cada una tendría una probabilidad de 1/2. Tenemos ceros para las dos primeras columnas de d1 ya que d1 no se enlaza a sí mismo y no se enlaza a d2. Por lo tanto, Mij es la probabilidad de ir de di a dj. Los valores de cada fila deben sumar uno, porque el navegante tendrá que ir precisamente a una de estas páginas. Ahora, ¿cómo podemos calcular la probabilidad de que un navegante visite una página en particular?

Podemos calcular la probabilidad de llegar a una página de la siguiente manera:

$$p_{t+1}(d_j) = \underbrace{(1-\alpha)\sum_{i=1}^{N} M_{ij}p_t(d_i)}_{\text{reach } d_j \text{ by following a link}} + \underbrace{\alpha\sum_{i=1}^{N}\frac{1}{N}p_t(d_i)}_{\text{reach } d_j \text{ by random jumping}} \tag{7.1}$$

En el lado izquierdo está la probabilidad de visitar la página d_j en el tiempo $t+1$, el siguiente conteo de tiempo. En el lado derecho, podemos ver que la ecuación involucra la probabilidad en la

página d_i en el tiempo t, el paso de tiempo actual. La ecuación captura las dos posibilidades de llegar a una página d_j en el tiempo $t+1$: a través de la navegación aleatoria o siguiendo un enlace.

La primera parte de la ecuación captura la probabilidad de que el navegante aleatorio llegue a esta página siguiendo un enlace. El navegante aleatorio elige esta estrategia con una probabilidad de $1-\alpha$; por lo tanto, hay un factor de $1-\alpha$ antes de este término. Este término suma todas las posibles N páginas en las que el navegante podría haber estado en el tiempo t. Dentro de la suma está el producto de dos probabilidades. Una es la probabilidad de que el navegante estuviera en d_i en el tiempo t. Esa es $p_t(d_i)$. La otra es la probabilidad de transición de d_i a d_j, que sabemos que se representa como M_{ij}. Entonces, para llegar a esta página d_j, el navegante debe estar primero en d_i en el tiempo t y tendría que seguir el enlace para ir de d_i a d_j.

La segunda parte es una suma similar. La única diferencia es que ahora la probabilidad de transición es uniforme: $\frac{1}{N}$. Esta parte captura la probabilidad de llegar a esta página a través de un salto aleatorio, donde α es la probabilidad de un salto aleatorio.

Esto también nos permite ver por qué PageRank captura un suavizado de la matriz de transición. Puedes pensar que este $\frac{1}{N}$ proviene de otra matriz de transición que tiene todos los elementos como $\frac{1}{N}$. Entonces, está claro que podemos combinar las dos partes. Debido a que son de la misma forma, podemos imaginar que hay una matriz diferente que es una combinación de esta M y la matriz uniforme I. En este sentido, PageRank utiliza esta idea de suavizado para asegurar que no haya ninguna entrada 0 en la matriz de transición.

Ahora, podemos imaginar que si queremos calcular las probabilidades promedio, estas satisfarían esta ecuación sin considerar el índice de tiempo. Así que eliminemos el índice de tiempo y asumamos que serían iguales; esto nos daría N ecuaciones, ya que cada página tiene su propia ecuación. De manera similar, también hay precisamente N variables. Esto significa que ahora tenemos un sistema de N ecuaciones lineales con N variables. El problema se reduce a resolver este sistema de ecuaciones, que podemos escribir de la siguiente forma:

$$p(d_j) = \sum_{i=1}^{N} \left[\frac{\alpha}{N} + (1-\alpha)M_{ij}\right] p(d_i) \rightarrow \vec{p} = (\alpha I + (1-\alpha)M)^T \vec{p}, \tag{7.2}$$

donde

$$I_{ij} = \frac{1}{N} \quad \forall i,j \tag{7.3}$$

El vector $\vec{p}$ es igual a la transpuesta de una matriz multiplicada por $\vec{p}$ nuevamente. La matriz transpuesta es, de hecho, la suma de 1 a N escrita en forma de matriz. Recordemos de álgebra lineal que esta es precisamente la ecuación para un vector propio. Por lo tanto, esta ecuación se puede resolver utilizando un algoritmo iterativo. En este algoritmo iterativo, llamado iteración de potencia, simplemente comenzamos con un $\vec{p}$ aleatorio. Luego, repetidamente actualizamos $\vec{p}$ multiplicando la expresión de la matriz transpuesta por $\vec{p}$.

Veamos un ejemplo concreto: establezcamos $\alpha = 0.2$. Esto significa que hay un 20% de probabilidad de saltar aleatoriamente a una página en toda la web y un 80% de probabilidad de seguir aleatoriamente un enlace desde la página actual. Tenemos la matriz de transición original M como antes que codifica los enlaces reales en el gráfico. Luego, tenemos esta matriz de transición de suavizado uniforme I que representa el salto aleatorio. Las combinamos juntas con interpolación a través de α para formar otra matriz que llamamos A:

$$A = (1-0.2)M + 0.2I = 0.8 \begin{bmatrix} 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \end{bmatrix} + 0.2 \begin{bmatrix} \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \end{bmatrix} \tag{7.4}$$

El algoritmo PageRank inicializará aleatoriamente $\vec{p}$ primero, y luego lo actualizará iterativamente utilizando la multiplicación de matrices. Si reescribimos esta multiplicación de matrices en términos de solo A, obtendremos lo siguiente:

$$\begin{bmatrix} p_{t+1}(d_1) \\ p_{t+1}(d_2) \\ p_{t+1}(d_3) \\ p_{t+1}(d_4) \end{bmatrix} = A^T \begin{bmatrix} p_t(d_1) \\ p_t(d_2) \\ p_t(d_3) \\ p_t(d_4) \end{bmatrix} = \begin{bmatrix} 0.5 & 0.85 & 0.05 & 0.45 \\ 0.5 & 0.05 & 0.85 & 0.45 \\ 0.45 & 0.05 & 0.05 & 0.05 \\ 0.45 & 0.05 & 0.05 & 0.05 \end{bmatrix} \begin{bmatrix} p_t(d_1) \\ p_t(d_2) \\ p_t(d_3) \\ p_t(d_4) \end{bmatrix} \tag{7.5}$$

Si deseas calcular el valor actualizado para d_1, multiplicas la fila superior en A por el vector columna de las puntuaciones de PageRank de la iteración anterior. Así es como actualizamos el vector; comenzamos con algunos valores iniciales y multiplicamos iterativamente las matrices, lo que genera un nuevo conjunto de puntuaciones. Repetimos esta multiplicación hasta que los valores en $\vec{p}$ converjan. De álgebra lineal, sabemos que dado que no hay valores cero en la matriz, dicha iteración está garantizada para converger. En ese punto, tendremos las puntuaciones de PageRank para todas las páginas.

Curiosamente, esta fórmula de actualización puede interpretarse como la propagación de puntuaciones a través del gráfico. Podemos imaginar que tenemos valores inicializados en cada una de estas páginas, y si observas la ecuación, combinamos las puntuaciones de las páginas que llevarían a alcanzar una página. Es decir, observamos todas las páginas que apuntan a una página y combinamos sus puntuaciones con la puntuación propagada para obtener la siguiente puntuación para el documento actual. Repetimos esto para todos los documentos, lo que transfiere la masa de probabilidad a través de la red.

En la práctica, el cálculo de la puntuación de PageRank es bastante eficiente porque las matrices son dispersas, lo que significa que si no hay un enlace hacia la página actual, no tenemos que preocuparnos por ello en el cálculo. También es posible normalizar la ecuación, y eso dará una forma algo diferente, aunque la clasificación relativa de las páginas no cambiará. La normalización es para abordar el problema potencial de los enlaces salientes cero. En ese caso, las probabilidades de alcanzar la siguiente página desde la página actual no sumarán 1 porque hemos perdido algo de masa de probabilidad cuando asumimos que hay alguna probabilidad de que el navegante intente seguir enlaces (aunque en este caso no hay enlaces que seguir).

Hay muchas extensiones a PageRank. Una extensión es hacer PageRank específico para consultas, también llamado PageRank Personalizado. Por ejemplo, en este PageRank específico para temas, podemos simplemente asumir que cuando el navegante se aburre, no saltará aleatoriamente a cualquier página en la web. En su lugar, saltará solo a aquellas páginas que son relevantes para la consulta. Por ejemplo, si la consulta es sobre deportes, entonces podríamos asumir que cuando hacemos saltos aleatorios, saltamos aleatoriamente a una página de deportes. Al hacer esto, nuestras puntuaciones de PageRank se alinean con los deportes. Por lo tanto, si sabes que la consulta actual es sobre deportes, podemos usar esta puntuación de PageRank especializada para clasificar los resultados. Claramente, esto sería mejor que usar una puntuación de PageRank genérica para toda la web.

PageRank es un algoritmo general que puede usarse en muchas otras aplicaciones, como el análisis de redes, particularmente en redes sociales. Podemos imaginar que si calculas la puntuación de PageRank de una persona en una red social (donde un enlace indica una relación de amistad), obtendrás puntuaciones significativas para las personas.

7.3.2 HITS

Hemos hablado sobre PageRank como una forma de capturar páginas de autoridad. Al principio de esta sección, también mencionamos que las páginas de concentrador son útiles. Hay otro algoritmo que discutiremos llamado HITS que está diseñado para calcular ambas puntuaciones para cada página.

Las páginas de autoridad capturan la intuición de páginas ampliamente citadas. Las páginas de concentrador son aquellas que apuntan a buenas páginas de autoridad o contienen alguna colección de conocimiento en forma de enlaces. La idea principal del algoritmo HITS es un mecanismo de refuerzo para ayudar a mejorar la puntuación tanto de concentradores como de autoridades. Asumirá que las buenas autoridades son citadas por buenos concentradores. Eso significa que si eres citado por muchas páginas con buenas puntuaciones de concentrador, entonces eso aumenta tu puntuación de autoridad. De manera similar, los buenos concentradores son aquellos que apuntan a buenas autoridades. Así que si estás apuntando a muchas buenas páginas de autoridad, entonces tu puntuación de concentrador aumentará. Al igual que PageRank, HITS también es bastante general y tiene muchas aplicaciones en el análisis de gráficos y redes. Brevemente, describiremos cómo funciona.

La Figura 10.8 muestra lo siguiente. Primero, construimos la matriz de adyacencia A; contiene un 1 en la posición A_{ij} si d_i enlaza a d_j y un cero en caso contrario. Definimos la puntuación de concentrador de una página $h(d_i)$ como la suma de las puntuaciones de autoridad de todas las páginas a las que apunta. En la segunda ecuación, definimos la puntuación de autoridad de una página $a(d_i)$ como la suma de las puntuaciones de concentrador de todas las páginas que apuntan a ella. Esto forma un mecanismo de refuerzo iterativo.

Estas dos ecuaciones también pueden escribirse en forma matricial. El vector de concentrador es igual al producto de la matriz de adyacencia y el vector de autoridad. De manera similar, la segunda ecuación puede escribirse como el vector de autoridad es igual al producto de A^T multiplicado por el vector de concentrador. Estas son solo diferentes formas de expresar estas ecuaciones. Lo interesante es que si observas las formas matriciales, puedes insertar la ecuación de autoridad en la primera. Es decir, puedes eliminar completamente el vector de autoridad, y obtienes la ecuación solo de las puntuaciones de concentrador. Podemos hacer el mismo truco para la fórmula de concentrador. Por lo tanto, aunque enmarcamos el problema como calcular concentradores y autoridades, en realidad podemos eliminar uno de ellos para obtener la ecuación para el otro.

La diferencia entre esto y PageRank es que ahora la matriz es en realidad una multiplicación de la matriz de adyacencia y su transpuesta. Matemáticamente, entonces, estaríamos resolviendo un problema muy similar. En HITS, inicializaríamos los valores a uno y aplicaríamos las ecuaciones matriciales A^TA y AA^T. Todavía necesitamos normalizar la matriz de adyacencia después de cada iteración. Esto nos permitiría controlar el crecimiento de los valores; de lo contrario, crecerían cada vez más.

Para resumir, esta sección ha mostrado que la información de enlaces es muy útil. En particular, el texto ancla es una característica importante en la representación de texto de una página. También hablamos sobre los algoritmos PageRank y HITS como dos algoritmos principales de análisis de enlaces en la búsqueda web. Ambos pueden generar puntuaciones para las páginas que pueden usarse además de las funciones de clasificación estándar de IR. PageRank y HITS son algoritmos muy generales con variantes útiles, por lo que tienen muchas aplicaciones en el análisis de otros gráficos o redes además de la web.

7.4 Aprendizaje para Clasificar

En esta sección, discutiremos el uso del aprendizaje automático para combinar muchas características diferentes en una única función de clasificación para optimizar los resultados de búsqueda. Anteriormente, hemos discutido varias formas de clasificar documentos. Hablamos sobre algunos modelos de recuperación como BM25 o la probabilidad de consulta; estos pueden generar una puntuación basada en el contenido para relacionar el texto del documento con una consulta. También hablamos sobre enfoques basados en enlaces como PageRank que pueden dar puntuaciones adicionales para ayudarnos a mejorar la clasificación.

La pregunta ahora es ¿cómo podemos combinar todas estas características (y potencialmente muchas otras características) para hacer la clasificación? Esto será muy útil para clasificar páginas web no solo para mejorar la precisión, sino también para mejorar la robustez de la función de clasificación de manera que no sea fácil para un spammer simplemente alterar una o algunas características para promover una página.

La idea general del aprendizaje para clasificar es usar el aprendizaje automático para combinar estas características, optimizando el peso de diferentes características para generar la mejor función de clasificación. Asumimos que dada una pareja consulta-documento (q, d), podemos definir una serie de características. Estas características no tienen que ser necesariamente basadas en el contenido. Podrían ser una puntuación del documento con respecto a la consulta según una función de recuperación como BM25, la probabilidad de consulta, la normalización de longitud pivoteada, PL2, etc. También puede haber una puntuación basada en enlaces como PageRank o HITS, o una aplicación de modelos de recuperación al texto ancla de la página, que son las descripciones de los enlaces que apuntan a d. Todas estas pueden ser pistas sobre si este documento es relevante o no para la consulta. Incluso podemos incluir una característica como si la URL tiene una tilde porque esto podría indicar una página de inicio.

La pregunta es, por supuesto, ¿cómo podemos combinar estas características en una única puntuación? En este enfoque, simplemente hipotetizamos que la probabilidad de que este documento sea relevante para esta consulta es una función de todas estas características. Hipotetizamos que la probabilidad de relevancia está relacionada con estas características a través de una función particular que tiene algunos parámetros. Estos parámetros controlan la influencia de diferentes características en la relevancia final. Esto es, por supuesto, solo una suposición. Si esta suposición realmente tiene sentido sigue siendo una pregunta abierta.

Naturalmente, la siguiente pregunta es cómo estimar esos parámetros. ¿Cómo sabemos qué características deberían tener un alto peso y cuáles deberían tener un bajo peso? Esta es una tarea de entrenamiento o aprendizaje.

En este enfoque, usamos datos de entrenamiento. Estos son datos que han sido juzgados por usuarios, por lo que ya conocemos los juicios de relevancia. Sabemos qué documentos deberían estar altamente clasificados para qué consultas, y esta información puede basarse en juicios reales de usuarios o puede aproximarse simplemente usando información de clics como discutimos en el Capítulo 7. Intentaremos optimizar la precisión de recuperación de nuestro motor de búsqueda (usando, por ejemplo, MAP o NDCG) en los datos de entrenamiento ajustando estos parámetros. Los datos de entrenamiento se verían como una tabla de tuplas. Cada tupla tiene tres elementos: la consulta, el documento y el juicio. Veamos un método específico que se basa en la regresión logística:

$$\log \frac{P(R=1|\, q,d)}{1-P(R=1|\, q,d)} = \beta_0 + \sum_{i=1}^{n} \beta_i X_i \tag{7.6}$$

Queremos muchos inlinks, de paginas buenas mejor, y si esas paginas no tienen muchos outlinks mejor.

Este es uno de muchos métodos diferentes, y en realidad uno de los más sencillos.

En este enfoque, simplemente asumimos que la relevancia de un documento con respecto a la consulta está relacionada con una combinación lineal de todas las características. Aquí tenemos X_i para denotar el valor de la característica i, y podemos tener tantas características como deseemos. Asumimos que estas características pueden combinarse de manera lineal. El peso de la característica X_i está controlado por un parámetro β_i. Un β_i más grande significaría que la característica tendría un peso mayor y contribuiría más a la función de puntuación.

La forma específica de la función también da la siguiente probabilidad de relevancia:

$$P(R=1|\,q,d) = \frac{1}{1+\exp\{-(\beta_0+\sum_{i=1}^{n}\beta_i X_i)\}} \tag{7.7}$$

Sabemos que la probabilidad de relevancia está dentro del rango [0, 1] y asumimos que la función de puntuación es una forma transformada de la combinación lineal de características. Podríamos haber tenido una función de puntuación basada directamente en la combinación lineal de β y X, pero entonces el valor de esta combinación lineal podría fácilmente superar 1. Por lo tanto, la razón por la que usamos regresión logística en lugar de regresión lineal es para mapear esta combinación al rango [0, 1]. Esto nos permite conectar la probabilidad de relevancia (que está entre 0 y 1) con una combinación lineal de coeficientes arbitrarios. Si reescribimos esta combinación de pesos en una función de probabilidad, obtendremos la puntuación predicha.

Si esta combinación de características y pesos nos da un valor alto, entonces el documento es más probable que sea relevante. Esto no es necesariamente la mejor hipótesis, pero es una forma simple de conectar estas características con la probabilidad de relevancia.

La siguiente tarea es ver cómo estimamos los parámetros para que la función pueda aplicarse verdaderamente; es decir, necesitamos estimar los valores de β. Veamos un ejemplo sencillo que se muestra en la Figura 10.9.

En este ejemplo, tenemos tres características. Una es la puntuación BM25 del documento para la consulta. Una es la puntuación PageRank del documento, que podría o no depender de la consulta. También podríamos tener una puntuación PageRank sensible al tema que dependería de la consulta. Por último, tenemos una puntuación BM25 en el texto ancla del documento. Estas son entonces las tres valores de características para una pareja particular (documento, consulta). En este caso, el documento es d_1 y el juicio dice que es relevante. El documento d_2 es otra instancia de entrenamiento con diferentes valores de características, pero en este caso no es relevante. Por supuesto, este es un ejemplo demasiado simplificado donde solo tenemos dos instancias, pero es suficiente para ilustrar el punto.

Usamos el estimador de máxima verosimilitud para estimar los parámetros. Es decir, vamos a predecir el estado de relevancia del documento basado en los valores de las características. La verosimilitud de observar el estado de relevancia de estos dos documentos usando nuestro modelo es

$$\begin{aligned} p(\{q,d_1,R=1\},\{q,d_2,R=0\}) = & \frac{1}{1+\exp\{-(\beta_0+0.7\beta_1+0.11\beta_2+0.65\beta_3)\}} \\ & \times\left(1-\frac{1}{1+\exp\{-(\beta_0+0.3\beta_1+0.05\beta_2+0.4\beta_3)\}}\right) \end{aligned} \tag{7.8}$$

pagerank es query inddependant. HITS no tuvo exito practico porque es muy costoso computacionalmente.

Hipotetizamos que la probabilidad de relevancia está relacionada con las características de esta manera. Vamos a ver para qué valores de β podemos predecir la relevancia de manera efectiva. La expresión para d_1 debería dar un valor más alto que la expresión para d_2; de hecho, esperamos que el valor de d_1 esté cerca de uno ya que es un documento relevante. Veamos cómo se puede expresar esto matemáticamente. Es similar a expresar la probabilidad de un documento, solo que no estamos hablando de la probabilidad de palabras, sino de la probabilidad de relevancia. Necesitamos insertar los valores de X. Los valores de β aún son desconocidos, pero esta expresión nos da la probabilidad de que este documento sea relevante si asumimos tal modelo. Queremos maximizar esta probabilidad para d_1 ya que este es un documento relevante. Para el segundo documento, queremos predecir la probabilidad de que el documento no sea relevante. Esto significa que tenemos que calcular 1 menos la probabilidad de relevancia. Ese es el razonamiento detrás de toda esta expresión; es nuestra probabilidad de predecir estos dos valores de relevancia. Toda la ecuación es nuestra probabilidad de observar un $R = 1$ y un

$R = 0$ para d_1 y d_2 respectivamente. Nuestro objetivo es entonces ajustar los valores de β para que toda la expresión alcance su valor máximo. En otras palabras, veremos la función y elegiremos valores de β para hacer que esta expresión sea lo más grande posible.

Después de aprender los parámetros de regresión, podemos usar esta expresión para cualquier nueva consulta y nuevo documento una vez que tengamos sus características. Esta fórmula se aplica entonces para generar una puntuación de clasificación para una consulta en particular.

Existen muchos algoritmos de aprendizaje más avanzados que los enfoques basados en regresión. Generalmente intentan optimizar teóricamente una medida de recuperación como MAP o NDCG. Nota que el objetivo de optimización que acabamos de discutir no está directamente relacionado con una medida de recuperación. Al maximizar la predicción de uno o cero, no necesariamente optimizamos la clasificación de esos documentos. Se puede imaginar que, aunque nuestra predicción puede no estar tan mal, la clasificación puede ser incorrecta. Podríamos tener una mayor probabilidad de relevancia para d_2 que para d_1. Así, eso no sería bueno desde una perspectiva de recuperación, aunque por verosimilitud la función no esté mal. Enfoques más avanzados intentarán corregir este problema. Por supuesto, el desafío es que el problema de optimización será más difícil de resolver. En contraste, podríamos tener otro caso donde predecimos probabilidades de relevancia alrededor de 0.9 para documentos no relevantes. Aunque la puntuación predicha es muy alta, siempre y cuando los documentos realmente relevantes reciban puntuaciones mayores a 0.9, la clasificación seguirá siendo aceptable para un usuario.

Estos enfoques de aprendizaje para clasificar son en realidad bastante generales. Pueden aplicarse a muchos otros problemas de clasificación además de los problemas de recuperación. Por ejemplo, sistemas de recomendación, publicidad computacional, resumen y muchas otras aplicaciones relevantes pueden resolverse utilizando este enfoque.

Para resumir, hablamos sobre el uso de aprendizaje automático para combinar características y predecir un resultado de clasificación. De hecho, el uso de aprendizaje automático en la recuperación de información comenzó hace muchas décadas. El feedback de Rocchio, discutido en el Capítulo 7, fue un enfoque de aprendizaje automático aplicado para aprender el feedback óptimo. Muchos algoritmos están impulsados por la disponibilidad de grandes cantidades de datos de entrenamiento en forma de clics. Estos datos proporcionan mucho conocimiento útil sobre la relevancia, y por eso se aplican métodos de aprendizaje automático para aprovechar esto. La necesidad de aprendizaje automático también está impulsada por el deseo de combinar muchos tipos diferentes de características para predecir una clasificación precisa. La búsqueda web especialmente impulsa esta necesidad ya que hay más características disponibles en la web que pueden ser aprovechadas para la búsqueda. Usar muchas características diferentes también aumenta la robustez de la función de puntuación, lo cual es útil para combatir el spam. Los motores de búsqueda modernos utilizan todo algún tipo de técnicas de aprendizaje automático para combinar muchas características y optimizar la clasificación, y esto es una característica principal de motores actuales como Google y Bing.

7.5 Futuro del web search

Dado que este capítulo concluye nuestra cobertura de los motores de búsqueda, hablaremos brevemente sobre algunas posibles tendencias futuras de la búsqueda web y los sistemas de recuperación de información inteligentes en general. Para mejorar aún más la precisión de un motor de búsqueda, es importante considerar casos especiales de necesidad de información. Una tendencia particular es tener cada vez más motores de búsqueda especializados y personalizados, que pueden llamarse motores de búsqueda verticales. Se espera que estos motores de búsqueda verticales sean más efectivos que los motores de búsqueda generales actuales porque podrían asumir que un usuario en particular pertenece a un grupo especial que podría tener una necesidad de información común. Debido a esta personalización, también es posible realizar la personalización. La búsqueda puede ser personalizada

porque tenemos una mejor comprensión de los usuarios. Restringir el dominio del motor de búsqueda también puede tener algunas ventajas en el manejo de los documentos, porque tendríamos una mejor comprensión de estos documentos. Por ejemplo, palabras particulares pueden no ser ambiguas en tal dominio, por lo que podemos evitar el problema de la ambigüedad.

Otra tendencia que podemos esperar ver son los motores de búsqueda que son capaces de aprender con el tiempo, una forma de aprendizaje de por vida o aprendizaje continuo. Esto es muy atractivo porque significa que el motor de búsqueda podrá auto-mejorarse. A medida que más personas lo utilicen, el motor de búsqueda se volverá cada vez mejor. Esto ya está sucediendo, porque los motores de búsqueda pueden aprender del feedback de relevancia. Más usuarios lo utilizan, y la calidad del motor de búsqueda permite que las consultas populares que son tipeadas por muchos usuarios recuperen mejores resultados.

Una tercera tendencia podría ser la integración del acceso a la información. La búsqueda, navegación y recomendación podrían combinarse para formar un sistema de gestión de información completo. Al principio de este libro, hablamos sobre el acceso push versus el acceso pull; estos modos pueden combinarse. Por ejemplo, si un motor de búsqueda detecta que un usuario está insatisfecho con los resultados de búsqueda, se puede hacer una "nota". En el futuro, si se rastrea un nuevo documento que coincida con la necesidad de información del usuario registrada en la nota, este nuevo documento podría ser enviado al usuario. Actualmente, la mayoría de los casos de recomendación de información son publicidad, pero en el futuro, puedes imaginar que la recomendación se integra de manera fluida en el sistema con acceso a la información multimodal.

Otra tendencia es que podríamos ver sistemas que intentan ir más allá de la búsqueda para apoyar las tareas de los usuarios. Después de todo, la razón por la cual las personas quieren buscar es para resolver un problema o tomar una decisión para realizar una tarea. Por ejemplo, los consumidores podrían buscar opiniones sobre productos para comprar un producto, por lo que sería beneficioso apoyar todo el flujo de trabajo de compra. Por ejemplo, a veces puedes ver la reseña mostrada directamente en los resultados de búsqueda; si el usuario decide comprar el producto, puede simplemente hacer clic en un botón para ir directamente al sitio de compras y realizar la compra. Aunque hay buen soporte para las compras, los motores de búsqueda actuales no proporcionan buen soporte para las tareas en muchas otras actividades. Los investigadores podrían querer encontrar trabajos relacionados o citas sugeridas. Actualmente, no hay mucho soporte para una tarea como escribir un artículo.

Podemos pensar en cualquier sistema inteligente—especialmente en sistemas inteligentes de información—especificado por tres nodos. Si conectamos estos nodos formando un triángulo, entonces podremos especificar un sistema de información. Podemos llamar a este triángulo el Triángulo de Datos-Usuario-Servicio. Las tres preguntas que planteas en las áreas siguen.

- A quien sirves?
- Que tipo de datos manejas?
- que tipo de servicio proporcionas?

Estas preguntas especifican un sistema de información; hay muchas formas diferentes de conectarlas. Dependiendo de cómo estén conectadas, podemos especificar todo tipo de sistemas diferentes. Consideremos algunos ejemplos.

En la parte superior de la Figura 10.10 hay diferentes tipos de usuarios. En el lado izquierdo, hay diferentes tipos de datos o información, y en la parte inferior, hay diferentes funciones de servicio. Ahora imagina que puedes conectar todo esto de diferentes maneras. Por ejemplo, si conectas a todos con páginas web y soportas la búsqueda y navegación, obtienes una búsqueda web. Si conectamos a los empleados universitarios con documentos de la organización o documentos empresariales y apoyamos la búsqueda y navegación, obtenemos una búsqueda empresarial.

Podríamos conectar a científicos con información de literatura para proporcionar todo tipo de servicios, incluyendo búsqueda, navegación, alertas a nuevos documentos relevantes, minería o análisis de tendencias de investigación, o soporte para tareas y decisiones. Por ejemplo, podríamos ser capaces de proporcionar soporte para generar automáticamente una sección de trabajos relacionados para un artículo de investigación; esto estaría más cerca del soporte de tareas. Luego, podemos imaginar que este sistema de información inteligente sería un tipo de asistente de literatura.

Si conectamos a compradores en línea con artículos de blogs o reseñas de productos, entonces podemos ayudar a estas personas a mejorar su experiencia de compra. Podemos proporcionar capacidades de minería de datos para analizar reseñas, comparar productos y el sentimiento sobre los productos, y proporcionar soporte para tareas o decisiones en la elección de qué producto comprar. O, podemos conectar Personal de servicio al cliente con correos electrónicos de los clientes. Imagina un sistema que pueda proporcionar un análisis de estos correos electrónicos para identificar las principales quejas de los clientes. Podemos imaginar un sistema que podría proporcionar soporte para tareas generando automáticamente una respuesta a un correo electrónico de un cliente adjuntando inteligentemente un mensaje de promoción si es apropiado. Si detectan un mensaje positivo (no una queja), entonces podrían aprovechar esta oportunidad para adjuntar alguna información promocional. Si es una queja, entonces podrías ser capaz de generar automáticamente una respuesta genérica primero y decirle al cliente que puede esperar una respuesta detallada más tarde. Todo esto tiene como objetivo ayudar a las personas a mejorar su productividad.

La Figura 10.10 muestra la tendencia de la tecnología y caracteriza los sistemas de información inteligentes con tres ángulos. En el centro de la figura hay un triángulo que conecta consultas con palabras clave a una representación de bolsa de palabras. Eso significa que los motores de búsqueda actuales básicamente proporcionan soporte de búsqueda a los usuarios y modelan principalmente a los usuarios basándose en consultas de palabras clave, viendo los datos a través de una representación de bolsa de palabras. Los motores de búsqueda actuales realmente no "comprenden" la información en los documentos indexados. Considera algunas tendencias hacia cada nodo hacia una función más avanzada, alejándose del centro. Imagina si podemos ir más allá de las consultas de palabras clave, mirar el historial de búsqueda del usuario y luego modelar aún más al usuario para comprender completamente el entorno de tareas del usuario, el contexto u otra información. Claramente, esto está impulsando la personalización y un modelo de usuario más completo, que es una dirección principal para construir sistemas de información inteligentes. En el lado del documento, también podemos ir más allá de una implementación de bolsa de palabras para tener una representación de entidad-relación. Esto significa que reconoceremos los nombres de las personas, sus relaciones, ubicaciones y cualquier otra información potencialmente útil. Esto ya es factible con las técnicas actuales de procesamiento de lenguaje natural.

Google ha iniciado parte de este trabajo a través de su Knowledge Graph. Una vez que podemos llegar a ese nivel sin mucho esfuerzo manual humano, el motor de búsqueda puede proporcionar un servicio mucho mejor. En el futuro, nos gustaría tener una representación de conocimiento donde quizás podamos agregar algunas reglas de inferencia, haciendo que el motor de búsqueda sea más inteligente. Esto requiere un análisis semántico a gran escala, y quizás esto inicialmente sea más factible para motores de búsqueda verticales. Es decir, es más fácil progresar en un dominio particular.

En el lado de los servicios, vemos que necesitamos ir más allá de la búsqueda para apoyar el acceso a la información en general; la búsqueda es solo una forma de acceder a la información. Más allá del acceso, también necesitamos ayudar a las personas a digerir la información una vez que se encuentra, y este paso es hacerlo mediante el análisis de la información o minería de datos. Tenemos que encontrar patrones o convertir la información textual en conocimiento real que pueda ser usado en aplicaciones o conocimiento accionable que pueda ser utilizado para la toma de decisiones. Además, el conocimiento será utilizado para ayudar al usuario a mejorar la productividad en la finalización de una tarea.

En esta dimensión, anticipamos que los futuros sistemas de información inteligentes proporcionarán

soporte interactivo para tareas. Debemos enfatizar "interactivo" aquí, porque es importante optimizar la inteligencia combinada de los usuarios y el sistema. Podemos obtener algo de ayuda de los usuarios de una manera natural sin asumir que el sistema tiene que hacer todo. Es decir, el usuario y la máquina pueden colaborar de manera inteligente y eficiente. Esta inteligencia combinada será alta y, en general, podemos minimizar el esfuerzo general del usuario en resolver su problema actual.

Esta es la visión general de los futuros sistemas de información inteligentes, y esperamos que pueda proporcionarnos algunas ideas sobre cómo realizar futuras innovaciones sobre lo que tenemos hoy y también motivar las técnicas adicionales que se cubrirán en los capítulos posteriores del libro.

Text analysis hasta pagina