Universidad de Valladolid

Facultad de Ciencias

Trabajo de Fin de Grado

Grado en Matemáticas

Técnicas Vectoriales de Búsqueda de Documentos

Autor: Iván Nieves Stantcheva
Tutor: Ángel Durán Martín
Curso 2024–2025

Dedicado a mi hermana,
que sabe casi tanto de matemáticas como sé yo.

No se ha usado ningún tipo de IA generativa
para la elaboración de este proyecto.

Índice general

Resumen

Pese a que existe una gran cantidad de artículos sobre una gran variedad de temas, las técnicas de búsqueda y catalogación de dichos artículos en función de sobre qué temas tratan en ocasiones resultan ser insuficientes. Una forma de plantear la relación entre artículos y sus ámbitos es expresar cada artículo como un vector cuyas componentes indican su relación un ámbito dado. El proceso de búsqueda por ámbitos se basa en la expresión de la petición como otro vector con el mismo formato para después realizar operaciones relativamente simples entre él y los vectores correspondientes a los artículos de la base de datos. Aplicando ciertas descomposiciones matriciales es posible disminuir la incertidumbre en los datos. Este trabajo explora cómo aplicar estas técnicas de álgebra lineal al proceso de búsqueda de documentos en colecciones de gran tamaño.

Palabras clave: álgebra lineal, análisis semántico latente, búsqueda de información, descomposición en valores singulares, espacios vectoriales, factorización QR, medidas de similitud.

Abstract

Despite there being an overwhelming amount of papers over an equally vast range of topics, techniques to efficently find relevant articles for a particular subject are sometimes lacking. A way to represent this relationswhip between articles and topic related to it models the article as a vector each of whose components represents its relationswhip with a given topic. A given user query is likewise converted into vector form following the same scheme, turning the search process into algebraic operations between it and the vectors corresponding to the articles stored in the database. Certain matrix decompositions help deal with uncertainty and noise in the original data. This paper explores how to apply these linear algrebra techniques to the search process in large text collections.

Key words: information retrieval, latent semantic analysis, linear algebra, QR decomposition, similarity measures, singular value decomposition, vector spaces.

Capítulo 1

Introducción

1.1. El problema de la catalogación de colecciones de textos de gran tamaño

Tradicionalmente el indexado y búsqueda de artículos científicos se ha basado en información «foránea» al contenido de dichos artículos, a saber: su título, autores, resumen y *abstract*, palabras clave, etc. Estos «metadatos», originalmente elaborados a mano, no suelen resultar útiles para entender el contenido de un artículo, pero sí para facilitar su búsqueda cuando un investigador desea explorar el «estado del arte».

A día de hoy es inviable buscar manualmente información en los más de 175 zettabytes de información contenida en Internet [28, 25]. Incluso reduciéndonos a colecciones más pequeñas, del orden de miles de documentos, resulta evidente que es necesario un sistema de búsqueda semi-automático.

Dejando de lado la cantidad de datos a manejar, el mayor problema que presenta el proceso es la inconsistencia de los datos de búsqueda: la determinación de los conceptos y palabras clave asociadas a un documento dependen de quién lo introduce en el sistema de almacenamiento, habiendo una disparidad media del 20% entre las etiquetas que asignan dos «indexadores» a un mismo documento [3].

Sin embargo, la inconsistencia de datos no es el único obstáculo a sortear: La polisemia (una palabra, múltiples significados) y la sinonimia (múltiples palabras, un significado) también complican el proceso de búsqueda, pues permiten «desacuerdos semánticos» entre el autor de un documento y el buscador. Una solución «tradicional» a estas complicaciones la otorga el sistema SMART [17, 18], que trata de solventar este problema mediante la determinación automática de palabras clave.

En este trabajo nos centraremos en la descripción de técnicas para la determinación de qué documentos de una colección resulta ser más relevante dada una ponderación de etiquetas indicada por un usuario buscador, suponiendo que las relaciones documento-etiqueta son correctas e intencionadas.

1.2. Estructura del TFG

El contenido principal de este trabajo está dividido en dos capítulos.

En el primero de ellos detallamos técnicas básicas de catalogación y búsqueda de documentos aplicando el «modelo de espacios vectoriales». Bajo este modelo, cada documento se expresa como un vector cuyas componentes se corresponden con las distintas palabras clave de la colección; estos vectores se almacenan en forma de matriz, y el proceso de búsqueda se traduce en la realización de ciertas operaciones sobre los vectores documento que los comparan con la petición del usuario.

El segundo capítulo trata sobre el análisis semántico latente (LSA por sus siglas en inglés), una variante del modelo de espacios vectoriales en la que la matriz que contiene los vectores documento es reemplazada por una aproximación de menor rango, lo que reduce el «ruido» contenido en la base de datos. Exploraremos cómo reducir el rango usando factorización QR y usando descomposición en valores singulares, algoritmos que también describimos y demostramos. Concluiremos el trabajo detallando algunas técnicas para la introducción de nuevos datos y la modificación de los ya existentes en relación con la reducción de rango mediante descomposición en valores singulares.

A lo largo del trabajo usaremos dos ejemplos ficticios para ilustrar los algoritmos que describimos; tras cada sección se incluye un apartado en el que se presenta una implementación en Mathematica de estos algoritmos. Al final del trabajo, el apéndice D reúne un listado de estos fragmentos de código.

Se incluye también otros tres apéndices que describen conceptos mencionados en y relevantes para el cuerpo del trabajo pero cuyo contenido resulta fuera de lugar dentro de él.

Capítulo 2

El modelo basado en espacios vectoriales

En este capítulo explicaremos el conocido como «modelo de espacios vectoriales», usado para representar de manera matricial una colección de documentos, y posteriormente detallaremos técnicas para la búsqueda de documentos según la relevancia que poseen en base a los contenidos buscados.

2.1. Representación vectorial de la información

En el modelo basado en espacios vectoriales [3], cada elemento de una colección (un «documento») se representa por un vector columna. Cada componente de dicho vector representa un posible concepto, etiqueta o palabra clave (un «término») con el que puede estar relacionado un documento, y la magnitud que posee una componente en un vector indica la importancia del término correspondiente dentro del documento.

Consideraremos únicamente magnitudes no negativas. Si bien se podrían usar valores negativos para indicar que un documento no trata en absoluto sobre un cierto término, depender de diferencias relativas y, sobre todo, usar cero como valor centinela (léase usar matrices dispersas) reduce el uso de memoria de los algoritmos que describimos más adelante.

Pongamos un ejemplo: si buscamos asignar etiquetas a una lista de videojuegos, una posible etiqueta (término) sería «puzle» y su valor en el vector correspondiente al juego (documento) «Ara Fell» sería 0.25, pues no es un juego de puzles aunque sí tiene pequeños puzles necesarios para avanzar, mientras que en el caso del videojuego «DotA 2» esa componente sería cero, pues no involucra puzles en absoluto.

La forma obvia de representar en este modelo una colección con D documentos y T términos es mediante una matriz de tamaño $T \times D$, esto es, con D vectores columna (uno por cada documento), cada uno con T elementos (uno por cada término). Denotaremos A a esta matriz y a_{td} al peso ponderado correspondiente al término t dentro del vector correspondiente al documento d. Hay varias formas de realizar esa

ponderación, siendo una de las más comunes asignar a cada término t un peso global g_t según su relevancia en la colección (reduciendo el peso de aquellos términos que menos información aporten, como por ejemplo el término «ordenador» dentro de una colección de artículos sobre informática) y multiplicar ese peso global por un peso local l_{td} que refleje la importancia del término t dentro del documento d. A su vez hay múltiples formas de especificar ese peso local, desde ajustes logarítmicos de frecuencia de aparición hasta simples factores booleanos (unos y ceros), siendo estos últimos los que usaremos en ejemplos posteriores a fin de simplificar las explicaciones. En ocasiones se aplica también un paso de normalización para restringir el rango posible de pesos (típicamente, al intervalo $[0, 1]$).

Experimentalmente se ha determinado que una determinación de pesos de alta calidad [16] viene dada por

$$a_{td} = \frac{f_{td} \times \log(D/n_t)}{\sqrt{\sum_{s=1}^{T} (f_{sd} \times \log(D/n_s))^2}}, \tag{2.1}$$

donde f_{td} es la frecuencia de aparición del término t en el documento d y n_t es el número de documentos que contienen el término t. Con la notación anterior tendríamos que $g_t = \log(D/n_t)$ y $l_{td} = f_{td}$ y que se ha aplicado un paso de normalización que hace que los vectores documento tengan norma euclídea 1.

Desde el punto de vista geométrico, estamos representando los documentos de la base de datos como puntos en $\mathbb{R}^T$ cuyas coordenadas vienen dadas por los vectores documento correspondiente. En líneas generales, la coordenada de mayor valor de un vector documento corresponde al término más prominente con el que se relaciona y documentos similares en cuanto a contenido se hallan más cercanos en $\mathbb{R}^T$. Del mismo modo, los documentos más relevantes para un término dado son aquellos para los que la componente correspondiente toma mayor valor. Aquellos documentos que traten sobre un solo término yacen sobre el semieje correspondiente y aquellos con mayor contenido semántico total se hallan más lejos del origen.

Un caso frecuente es que haya más documentos que términos (esto es, la matriz A tiene más columnas que filas; $D > T$), aunque si la colección de documentos abarca contenido muy variado suele ocurrir lo contrario ($T > D$) y además puede haber documentos «enciclopédicos» que traten muchos términos; pondremos ejemplos explicando ambos casos más adelante. En general, un documento individual suele tratar únicamente sobre unos pocos términos, por lo que la mayoría de entradas a_{td} son ceros. Como se mencionó previamente, esto último hace que la manera ideal y más común de almacenar en tiempo de ejecución los elementos de A sea mediante matrices dispersas, aunque es posible que, «en reposo», A admita formas mejores de compresión (como puede ser *run-length encoding*—véase el apéndice A).

También cabe observar que, aunque toda la información semántica (en cuanto a la relación entre términos y documentos) de A viene dada por el espacio vectorial generado por sus columnas, no todas las posibles combinaciones lineales de documentos tienen sentido si las consideramos como documentos en sí: si estamos clasificando géneros de libros, ¿cómo podemos sumar la mitad de los géneros de «Winning Ways» a tres veces los de «Words of Radiance»?

Volviendo a la búsqueda de información, a partir de las filas de la matriz A podemos estudiar el paralelismo entre los distintos términos usados en la colección analizada y considerando las columnas, extraer y modelizar propiedades sobre el contenido de la colección en sí. Apoyándonos en esto último podemos modelizar también el proceso de búsqueda semántica de documentos: Solicitamos al usuario que ponderе qué términos considera más relevantes para su propósito y construimos un nuevo vector, como si de un nuevo documento se tratase, y determinamos aquellos vectores documento que sean más cercanos (para alguna definición de «cercano») a su petición. La implementación de este proceso de búsqueda de se denomina *query matching* y la exploraremos en detalle en la siguiente sección.

2.1.1. Ejemplo: Etiquetas de Videojuegos

Pongámonos en el supuesto de que queremos clasificar videojuegos según ciertas etiquetas para generar recomendaciones según los videojuegos jugados recientemente, de manera similar a la cola de recomendaciones que ofrece la plataforma Steam. Este mismo ejemplo puede extrapolarse de manera obvia a la clasificación de libros o películas según su género, o colecciones de artículos según los temas que tratan etc.

En este supuesto, y según lo que hemos detallado antes, cada documento sería un videojuego (por ejemplo, «The Witness») y cada término sería una etiqueta posible (por ejemplo, «Difícil»). Para facilitar los cálculos y simplificar las explicaciones nos limitaremos al caso más simple, en el que no se realiza una ponderación global de los términos ($g_t = 1$) y el peso local de los términos es booleano ($l_{td} \in \{0, 1\}$).

Supondremos que ya hemos decidido qué y cuántos juegos y etiquetas vamos a clasificar (en secciones posteriores abordaremos las técnicas para añadir más términos y documentos a una base de datos ya existente). En este caso asignaremos 10 posibles etiquetas a un total de 20 juegos.

La sección (matriz) de la tabla 2.1 que contiene los ceros y unos es lo que hemos denominado base de datos A; esta matriz tiene rango diez. Leyendo por columnas, el «vector documento» correspondiente al juego «The Witness» contiene unos únicamente en las posiciones correspondientes a las etiquetas «Difícil», «Mundo Abierto» y «Puzles», lo que nos indica que ese juego posee esas etiquetas (y sólo esas). Leyendo por filas, el «vector término» correspondiente a la etiqueta «Cartas» nos indica que los juegos de nuestra base de datos que tienen esa etiqueta son únicamente «Chroma: Bloom and Blight» y «Slay the Spire».

De cara a la implementación de los algoritmos posteriores, por ahora únicamente almacenaremos la matriz A y asignaremos a sus filas y columnas los términos y documentos correspondientes. El primer argumento a `IntegerDigits` es la representación en base 36 de los elementos de la matriz A; podemos considerar esto como una forma primitiva de compresión de la base de datos: en vez de almacenar la matriz en sí almacenamos un número que permite recomponer la base de datos en tiempo de ejecución.

	Ara Fell	Baba is You	Brogue	Chroma: Bloom and Blight	Divinity: Original Sin 2	DotA 2	Fell Seal: Arbiter's Mark	Fire Emblem: Heroes	Iconoclasts	Islands of Insight	Octopath Traveller	One Step From Eden	Slay the Spire	Slice & Dice	Symphony of War	Taiji	Terraria	The Elder Scrolls V: Skyrim	The Witness	Wargroove
Cartas	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
Difícil	0	1	1	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	1	0
Estrategia	1	0	1	1	1	1	1	1	0	0	1	0	1	1	1	0	0	0	0	1
Multijugador	0	1	0	0	1	1	0	0	0	1	0	0	0	0	0	0	1	1	0	0
Mundo Abierto	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	1	1	0
Por Turnos	1	1	1	1	1	0	1	1	0	0	1	0	1	1	1	0	0	0	0	1
Puzles	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1	0
PvP	0	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
RPG	1	0	0	0	1	0	1	1	1	0	1	0	0	0	1	0	0	1	0	1
Roguelike	0	0	1	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0

Tabla 2.1: Documentos (columnas), términos (filas) y base de datos del primer ejemplo.

```
database = IntegerDigits[
    36^^q9vwuqqx9w4b32pojqzyj1uozz3zuxmv7x630g, 2, 10*20] \
    // ArrayReshape[#, {10, 20}]& \
    // Map[AssociationThread[{
        "Cartas",       "Difícil",  "Estrategia",   "Multijugador", "Mundo Abierto",
        "Por Turnos",   "Puzles",   "PvP",          "RPG",          "Roguelike"
    }, #] &] \
    // AssociationThread[{
        "Ara Fell",                 "Baba is You",              "Brogue",
        "Chroma: Bloom and Blight", "Divinity: Original Sin 2", "DotA 2",
        "Fell Seal: Arbiter's Mark", "Fire Emblem: Heroes",     "Iconoclasts",
        "Islands of Insight",       "Octopath Traveller",       "One Step From Eden",
        "Slay the Spire",           "Slice & Dice",             "Symphony of War",
        "Taiji",                    "Terraria",        "The Elder Scrolls V: Skyrim",
        "The Witness",              "Wargroove"
    }, #] &
```

2.1.2. Ejemplo: Tipos de Criatura*

En el ejemplo anterior tenemos que $D > T$, pues hay más documentos que términos; sin embargo, como mencionamos anteriormente, también puede darse el caso contrario: que haya más términos que documentos. Puesto que más adelante explicaremos algoritmos en los que hay que distinguir estos dos casos, vamos a poner un ejemplo en el que se tiene $D < T$.

Vamos a ponernos en el supuesto de un juego en el que se premia al jugador por conseguir múltiples «criaturas» que compartan ciertos «tipos». Ahora el objeti-

*Los contenidos de este ejemplo están inspirados en un prototipo de juego de mesa creado por Philipp "Baumi" Baumgart, aún por publicar. Agradecemos que nos haya permitido usarlo.

vo de nuestras búsquedas es encontrar aquellas criaturas que proporcionen mayor sinergia (esto es, que compartan más tipos) con las que el jugador ya tiene. Los resultados de estas búsquedas se podrían usar como una inteligencia artificial primitiva que juegue automáticamente, o como asistente en tiempo real durante las partidas.

Al igual que en el ejemplo anterior vamos a usar pesos locales booleanos sin ponderación global de términos, aunque en este ejemplo consideraremos un total de 12 criaturas y 16 tipos de criatura, haciendo que en este caso la matriz tenga más filas que columnas—justo al contrario que la matriz del primer ejemplo. También a diferencia del otro ejemplo, la matriz de la base de datos no tiene rango máximo (tiene rango once). La tabla 2.2 recoge los datos de que usaremos para este ejemplo.

	Acquisitions Expert	Ardent Electromancer	Cascade Seer	Changeling Outcast	Farisght Adept	Joraga Bard	Mul Daya Channelers	Nimble Trapfinder	Skyclave Cleric	Tajuru Paragon	Vine Gecko	Werefox Bodyguard
Ally	0	0	0	1	0	1	0	0	0	0	0	0
Bard	0	0	0	1	0	1	0	0	0	0	0	0
Cleric	0	0	0	1	0	0	0	0	1	1	0	0
Druid	0	0	0	1	0	0	1	0	0	0	0	0
Elf	0	0	0	1	0	1	1	0	0	1	0	1
Elemental	0	0	0	1	0	0	0	0	0	0	1	0
Fox	0	0	0	1	0	0	0	0	0	0	0	1
Human	1	1	0	1	0	0	0	1	0	0	0	0
Knight	0	0	0	1	0	0	0	0	0	0	0	1
Kor	0	0	0	1	1	0	0	0	1	0	0	0
Lizard	0	0	0	1	0	0	0	0	0	0	1	0
Merfolk	0	0	1	1	0	0	0	0	0	0	0	0
Rogue	1	0	0	1	0	1	0	1	0	1	0	0
Shaman	0	0	0	1	0	0	1	0	0	0	0	0
Warrior	0	0	0	1	0	0	0	0	0	1	0	0
Wizard	0	1	1	1	1	0	0	0	0	1	0	0

Tabla 2.2: Documentos (columnas), términos (filas) y base de datos del segundo ejemplo.

Observamos que «*Changeling Outcast*» es un documento «enciclopédico»: se relaciona con una gran cantidad de términos (en este caso, con todos). Aunque parezca poco intuitivo, este hecho puede hacer que el proceso de búsqueda lo considere menos relevante que otros documentos con menos términos, como veremos más adelante.

Al igual que para el ejemplo anterior, por ahora nos conformaremos con almacenar la matriz A y los términos y documentos para usarlos en secciones posteriores.

```
database2 = IntegerDigits[
    36^^4lzfzxpizuw6yr2w3dde738byf9o1nfzi5eo4, 2, 16*12] \
    // ArrayReshape[#, {16, 12}]& \
    // Map[AssociationThread[{
        "Acquisitions Expert", "Ardent Electromancer", "Cascade Seer",
        "Changeling Outcast" , "Farisght Adept"      , "Joraga Bard",
        "Mul Daya Channelers", "Nimble Trapfinder"   , "Skyclave Cleric",
        "Tajuru Paragon"     , "Vine Gecko"          , "Werefox Bodyguard"
    }, #] &] \
    // AssociationThread[{
        "Ally"  , "Bard", "Cleric", "Druid"  , "Elf"  , "Elemental","Fox"    ,"Human",
        "Knight", "Kor" , "Lizard", "Merfolk", "Rogue", "Shaman"   ,"Warrior","Wizard"
    }, #] & \
    // Transpose
```

2.2. *Query Matching*

2.2.1. Medidas de similitud

En la sección anterior mencionamos que el *query matching* [3] es la búsqueda de documentos cuyo vector correspondiente se acerque más a otro dado (el vector petición). Antes de empezar el proceso de búsqueda debemos definir la métrica de éxito: ¿en base a qué criterio son los vectores encontrados los más cercanos al vector petición? Como también mencionamos, aquellos documentos más cercanos tienen vectores documento con coordenadas más cercanas en $\mathbb{R}^T$; ¿cómo podemos expresar esto más precisamente?

Quizá en este punto al lector se le haya ocurrido usar la distancia euclídea para medir la diferencia entre los vectores documento y petición, pero incluso en ejemplos simples no funciona correctamente. En efecto, consideremos los vectores documento $\begin{pmatrix}0 & 1\end{pmatrix}^\intercal$ y $\begin{pmatrix}5 & 0\end{pmatrix}^\intercal$ y el vector petición $\begin{pmatrix}1 & 0\end{pmatrix}^\intercal$. Usando la distancia euclídea, el vector documento más cercano al vector petición resulta ser el primero, a pesar de que ninguno de sus términos coincide con los buscados. Es más, el mismo ejemplo pone en evidencia que ni la distancia de Manhattan ni la correspondiente a la norma infinito son buenas métricas de éxito para el *query matching*.

Existen múltiples «medidas» (campos escalares) que no presentan este problema, como detallaremos a continuación. Antes de entrar en detalles, vamos a definir el concepto de «contorno de iso-similitud», que proporciona una forma geométrica de estudiar el comportamiento de cada medida. Si representamos por $s_{\mathbf{d}}(\mathbf{q})$ a la similitud siguiendo alguna medida entre el vector documento $\mathbf{d}$ y un vector petición $\mathbf{q}$ que suponemos constante*, llamamos contorno de iso-similitud correspondiente a un valor de similitud $\nu \in \mathbb{R}$ al «conjunto de nivel» dado por

$$\left\{\mathbf{d} \in \mathbb{R}^T \middle| s_{\mathbf{d}}(\mathbf{q}) = \nu\right\}.$$

Los contornos de iso-similitud particionan $\mathbb{R}^T$ y dependen no solo de ν, sino también de la medida de similitud considerada y del vector petición.

*En la expresión $s_{\mathbf{d}}(\mathbf{q})$ ponemos $\mathbf{d}$ como subíndice en vez de argumento simplemente para representar que se «conoce antes» que el vector petición.

También es posible considerar el contorno de iso-similitud «asociado» a un vector documento $\mathbf{e}$ dado sin más que tomar $v = s_{\mathbf{e}}(\mathbf{q})$, obteniéndose entonces el contorno de iso-similitud

$$\left\{\mathbf{d} \in \mathbb{R}^T \,\middle|\, s_{\mathbf{d}}(\mathbf{q}) = s_{\mathbf{e}}(\mathbf{q})\right\}.$$

Medida del producto escalar

Una de las «medidas» más simples viene dada por el producto escalar [10]: si $\mathbf{q} = (q_1, q_2, \ldots, q_T)^\intercal$ es el vector petición, podemos calcular su similitud con el d-ésimo vector documento de la base de datos aplicando

$$s_d^{\text{dot}}(\mathbf{q}) := \sum_{t=1}^{T} a_{td} q_t, \tag{2.2}$$

donde s_d^{dot} representa la similitud* del documento d según la medida del producto escalar (*dot product* en inglés) y a_{td} es la componente del vector documento d asociada al término t para $d = 1, 2, \ldots, D$, y $t = 1, 2, \ldots, T$.

Para esta medida es fácil determinar los conjuntos de iso-similitud asociados a un vector petición dado: puesto que $\mathbf{x} \cdot \mathbf{y} = (\mathbf{x} + \mathbf{y}^\perp) \cdot \mathbf{y}$ para cualquier $\mathbf{y}^\perp$ ortogonal a $\mathbf{y}$, cada conjunto de iso-similitud es un hiperplano ortogonal al vector petición. En dos dimensiones, estos hiperplanos se reducen a rectas perpendiculares a la recta que pasa por el origen con dirección el vector petición (las líneas punteadas de la figura 2.1).

Además de facilidad de implementación y de paralelización, medir la similitud mediante el producto escalar (2.2) presenta ciertas propiedades notables, heredadas de las del producto escalar (véase también 2.1):

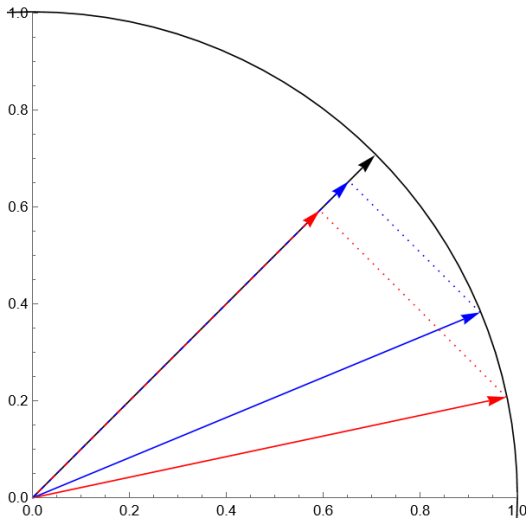
- Monotonía en cuanto a ángulos: Fijada una petición, dados dos documentos cuyos vectores correspondientes tienen igual norma euclídea se considera más similar aquel que forme un ángulo menor con el vector petición.

- Monotonía radial: En el caso de que vectores documento formen el mismo ángulo con el vector petición se considera más similar aquel que tenga mayor norma.

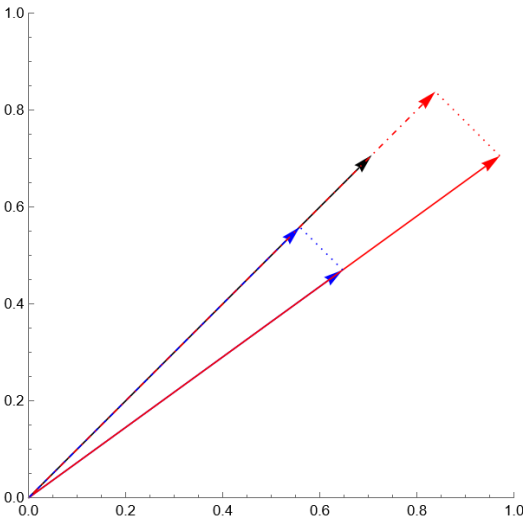
- Monotonía componente a componente: Aumentar una cierta componente de un vector documento nunca disminuye (pero puede no aumentar) la similitud con el vector petición, sea cual sea este.

- Influencia no limitada de cada componente: Valores arbitrariamente grandes de una componente en un vector documento pueden hacer que dicho documento se considere similar a toda petición que involucre a esa componente.

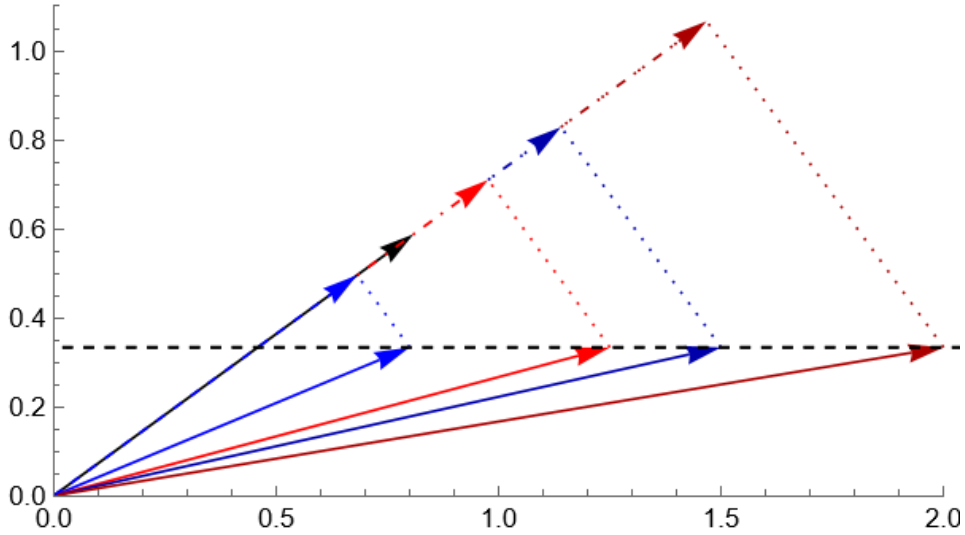
*Hemos cambiado ligeramente la notación para evitar sobrecargarla: el subíndice representa el índice del vector documento en la base de datos y no dicho vector en sí.



(a) Monotonía en cuanto a ángulos.



(b) Monotonía Radial.



(c) Influencia no limitada de cada componente.

Figura 2.1: Propiedades de la medida (2.2). En línea sólida negra: vector petición. En línea sólida azul y roja: vectores documento. Las líneas punteadas representan conjuntos de nivel asociados a los vectores documento. En todos los casos la similitud es la longitud de las flechas rayadas y/o punteadas, que son las proyecciones de los vectores documento sobre el vector petición.

- Similitud no acotada superiormente: Dado que las coordenadas de los vectores involucrados en el producto escalar son no negativos, el producto escalar tampoco lo es, luego la medida (2.2) está acotada inferiormente por cero. No obstante, de acuerdo a lo anterior, el producto escalar sí puede tomar valores tan grandes como se quiera, manipulando las coordenadas del vector petición y/o alguno de los vectores documento.

Las dos últimas propiedades de esta medida son desventajosas y es fácil ver por

qué: Consideremos una base de datos cuyos vectores documento sean $\begin{pmatrix}1 & 0\end{pmatrix}^\intercal$ y $\begin{pmatrix}1 & 5\end{pmatrix}^\intercal$, y un vector petición $\begin{pmatrix}5 & 1\end{pmatrix}^\intercal$. De acuerdo a la fórmula anterior, las similitudes del vector petición a los vectores documento resultan ser 5 y 10 respectivamente, de modo que se consideraría más relevante el segundo, a pesar de que el buscador presumiblemente estaba interesado más en el primer término, que es sobre el que trata exclusivamente el primer documento.

Obviando el problema de la influencia ilimitada anterior, el hecho de que la similitud indicada por la ecuación (2.2) tome valores arbitrariamente grandes hace imposible fijar un umbral para separar los documentos en relevantes y no relevantes que sirva para gran parte de las bases de datos y vectores petición posibles. Siguiendo con el ejemplo anterior, si multiplicamos todos los elementos del vector petición por 2, las similitudes también se duplican, haciendo que algunos umbrales que separan los dos documentos en el supuesto original no lo hagan en el duplicado.

Medida del coseno

Consideremos otra alternativa, en parte basada en la anterior: comparemos el ángulo que forman los vectores documento con respecto al vector petición y busquemos maximizar el coseno de ese ángulo. Siguiendo con la notación anterior, podemos calcular el coseno del ángulo que forma con cada uno de los vectores documento usando

$$s_d^{\cos}(\mathbf{q}) := \frac{\sum_{t=1}^{T} a_{td} \times q_t}{\sqrt{\sum_{t=1}^{T} a_{td}^2} \times \sqrt{\sum_{t=1}^{T} q_t^2}} = \frac{a_d \cdot \mathbf{q}}{\|a_d\|_2 \times \|\mathbf{q}\|_2}, \tag{2.3}$$

con $\mathbf{q} = (q_1, q_2, \ldots, q_T)^\intercal$ el vector petición y $a_d = (a_{1d}, a_{2d}, \ldots, a_{Td})^\intercal$ el d-ésimo vector documento para $d = 1, 2, \ldots, D$.

Puesto que todas las componentes de los vectores involucrados son no negativas, el ángulo que forman será de a lo sumo $\pi/2$ radianes, luego su coseno está entre cero y uno. Se podría tomar el arcocoseno de la expresión anterior y considerar el ángulo directamente como la medida de similitud, pero esto es computacionalmente caro y no otorga ningún beneficio sobre el uso de su coseno.

Observamos que la magnitud (norma euclídea) tanto de los vectores documento como petición no altera el valor del coseno, por lo que si normalizamos dichos vectores cuando entran en el sistema evitamos tener que realizar la división de la expresión anterior para calcular la similitud entre cada par de vectores documento y petición. Una vez hecha esta normalización, la fórmula (2.3) se convierte en (2.2) y por tanto resultan ser igual de costosas computacionalmente.

Asimismo podemos comparar las propiedades de esta medida con las correspondientes la medida (2.2) (véase también la figura 2.2):

- Monotonía en cuanto a ángulos: Al igual que en el caso del producto escalar, la similitud aumenta conforme el ángulo entre vector petición y vector documento disminuye.

- Ausencia de monotonía radial: Puesto que los vectores involucrados son normalizados (a priori o a posteriori), dos vectores que forman el mismo ángulo tienen igual similitud, independientemente de su norma. Los contornos de iso-similitud son por tanto las semirrectas que parten del origen.

- Ausencia de monotonía componente a componente: Ahora sí es posible disminuir la similitud aumentando los valores de una componente de un vector documento: En el ejemplo anterior, considérese aumentar la segunda componente del segundo vector documento.

- Influencia limitada de cada componente: La presencia de una componente muy elevada en un vector documento lo aleja (en cuanto a ángulo) de cualquier vector petición que tenga más de una componente no nula, haciendo que la similitud disminuya (véase la figura 2.2c). Lo anterior también ocurre si el vector petición tiene una única componente no nula y la componente aumentada no es dicha componente, por lo que podemos decir que en general, la medida (2.3) es resistente a los efectos de aumentar desmesuradamente las componentes de los vectores involucrados.

- Similitud acotada: La medida del coseno únicamente toma valores entre cero y uno, haciendo posible establecer umbrales que funcionen para la mayoría de bases de datos y vectores petición. El umbral más común es 0.9; exploraremos los efectos de variar este umbral en ejemplos posteriores.

Si bien las medidas de similitud mediante productos escalares y cosenos son las más simples y comunes, existen otras que también cabe destacar.

Medida del pseudo-coseno

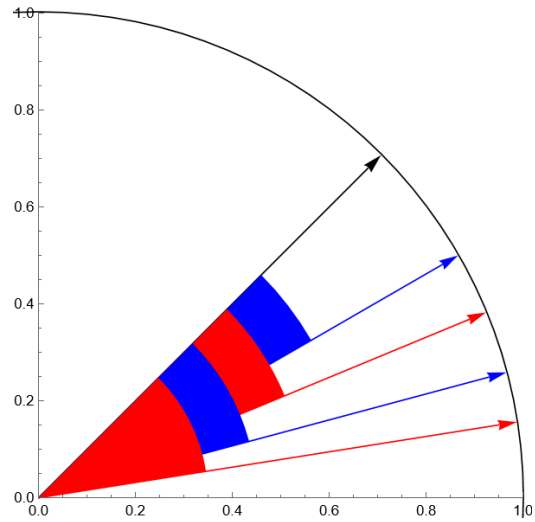
Una variante de la medida (2.3) es la conocida como «medida del pseudo-coseno», que, en vez de normalizar la norma euclídea, normaliza la norma asociada a la distancia de Manhattan y viene dada por

$$s_d^{\text{psc}}(\mathbf{q}) := \frac{\sum_{t=1}^{T} a_{td} \times q_t}{\left(\sum_{t=1}^{T} a_{td}\right) \times \left(\sum_{t=1}^{T} q_t\right)} = \frac{a_d \cdot q}{\|a_d\|_1 \times \|\mathbf{q}\|_1}, \tag{2.4}$$

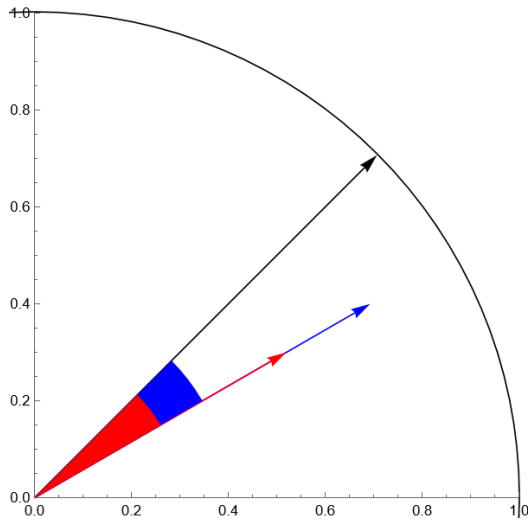
con $\mathbf{q} = (q_1, q_2, \ldots, q_T)^\intercal$ el vector petición y $a_d = (a_{1d}, a_{2d}, \ldots, a_{Td})^\intercal$ el d-ésimo vector documento para $d = 1, 2, \ldots, D$.

De acuerdo a los mismos argumentos usados para comparar las medidas (2.2) y (2.3), calcular la similitud mediante (2.4) se resume en hacer un producto escalar, luego es igual de costosa que esas dos medidas.

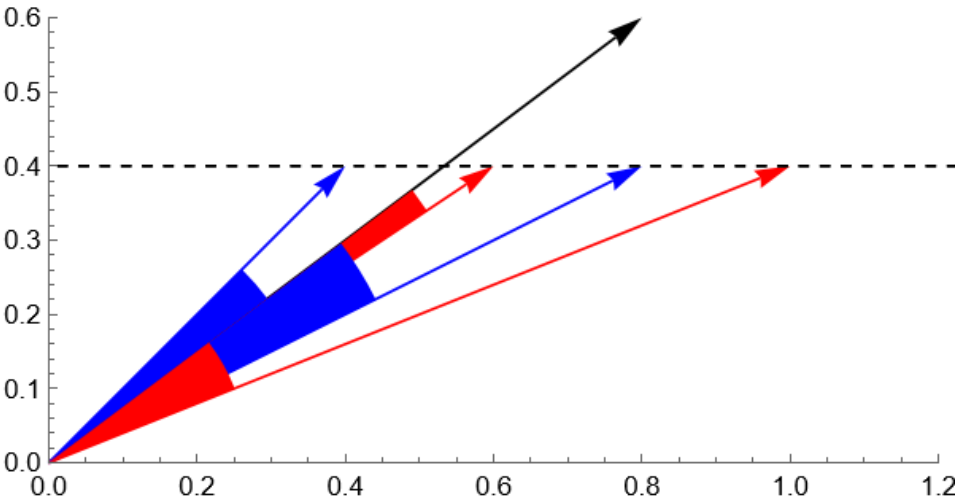
Al igual que para la medida del coseno, los conjuntos de iso-similitud son las semirrectas que parten del origen. A diferencia de la medida del coseno, esas semirrectas son «colapsadas» al hiperplano que pasa por los puntos correspondientes a los vectores de la base canónica de $\mathbb{R}^T$ en vez de a la parte de la esfera (hueca) unidad que queda en el primer hiperoctante.



(a) Monotonía en cuanto a ángulos.



(b) Monotonía Radial.



(c) Influencia limitada de cada componente.

Figura 2.2: Propiedades de la medida (2.3). En negro: vector petición. En azul y rojo: vectores documento. En todos los casos la similitud es el coseno de los ángulos coloreados; mayor similitud corresponde a un ángulo menor (más cerrado).

También a diferencia de la medida (2.3)—para la que lo verdaderamente importante es la dirección del vector petición,—para la medida del pseudo-coseno lo más relevante a la hora de calcular la similitud es la componente (o componentes) de mayor valor del vector petición (su dirección o direcciones «preferidas»). Más concretamente, aquellos vectores documento que solo tienen por componente no nula la correspondiente a una o más direcciones preferidas son los que se consideran más similares. Si *todas* las direcciones son preferidas (esto es, si el vector petición es de la forma $(\lambda, \lambda, \ldots, \lambda)^\intercal$ para algún $\lambda \in \mathbb{R}$), la medida (2.4) considera igual de similares *todos* los documentos (véase la figura 2.3b).

El resto de propiedades de la medida del pseudo-coseno son también distintas a la correspondiente al coseno:

- Monotonía respecto a las direcciones preferidas: Conforme el ángulo entre un vector documento y la dirección preferida (el ángulo más pequeño con cada una de ellas, si hay varias) aumenta, la similitud disminuye.
- Ausencia de monotonía radial: Al igual que la medida (2.3), se usan vectores normalizados luego la norma de dichos vectores no afecta a la similitud.
- Ausencia de monotonía componente a componente: En este caso, aumentar una componente de un vector documento aumenta la similitud si y solo si hace que el ángulo con la dirección preferida más «cercana en ángulo» disminuya.
- Influencia limitada de cada componente: En general, la medida del pseudocoseno es resistente a los efectos de aumentar una componente cualquiera de un vector documento, exhibiendo el mismo comportamiento que la medida (2.3) en ese aspecto. Véase las figuras 2.3c y 2.3d.
- Similitud acotada: El rango de similitudes posibles depende del vector petición, pero en todos los casos está acotado. Los casos extremos son aquellos en los que el vector petición está sobre un eje, en cuyo caso el rango es $[0, 1]$, y aquellos en los que el vector petición cae sobre la diagonal, donde la similitud es siempre $1/T$.

La dependencia de la medida (2.4) de la direcciones preferidas del vector petición, junto con su comparativamente extraña monotonía hacen que esta medida sea poco recomendable para el *query matching*.

Medida de Dice

Existe otra medida que, al igual que (2.4), trata de normalizar los vectores en función de la norma uno: la medida de Dice, basada en el conocido como coeficiente de Dice-Sørensen [24]:

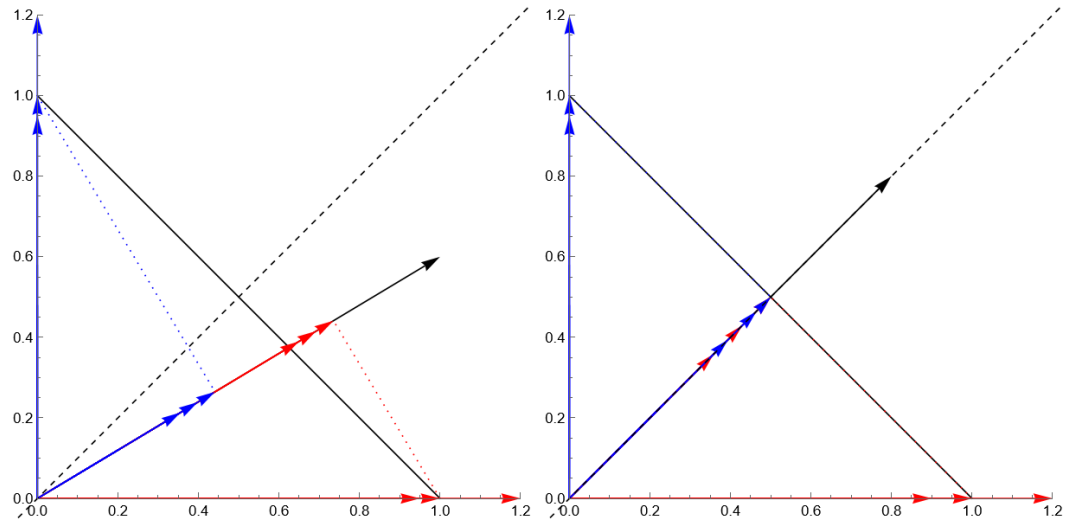
$$s_d^{\text{Dice}}(\mathbf{q}) := \frac{2 \times \sum_{t=1}^{T} a_{td} \times q_t}{\left(\sum_{t=1}^{T} a_{td}\right) + \left(\sum_{t=1}^{T} q_t\right)} = \frac{2 \times a_d \cdot \mathbf{q}}{\|a_d\|_1 + \|\mathbf{q}\|_1}, \tag{2.5}$$

con $\mathbf{q} = (q_1, q_2, \ldots, q_T)^\intercal$ el vector petición y $a_d = (a_{1d}, a_{2d}, \ldots, a_{Td})^\intercal$ el d-ésimo vector documento para $d = 1, 2, \ldots, D$.

A diferencia de (2.3) y (2.4), esta medida no posee un paso previo de normalización, por lo que en este caso el cociente es inevitable en tiempo de búsqueda. Observamos también que esta medida es sensible a la diferencia relativa de las normas respectivas de los vectores documento y petición.

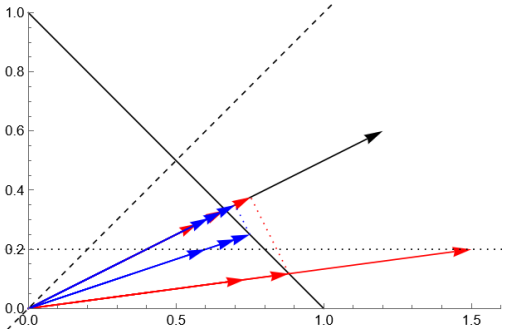
En el caso extremo en que el vector petición tiene norma despreciable respecto a los vectores documento, tenemos que esta medida es proporcional a la (2.4). En el otro caso extremo, en que el vector petición tiene norma mucho mayor, (2.5) se vuelve proporcional a (2.2).

El comportamiento de (2.5) depende por tanto del vector petición considerado y en general sus propiedades son una mezcla de las de las medidas (2.4) y (2.2). Los

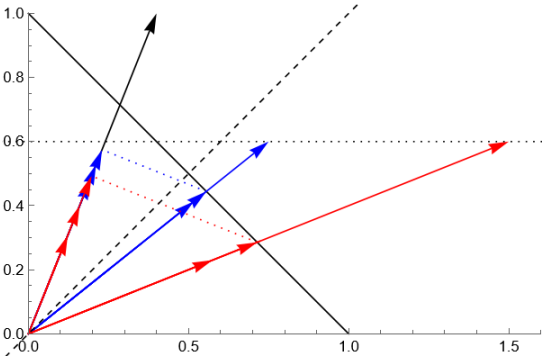


(a) Casos extremos.

(b) Colapso.



(c) Influencia de las direcciones preferidas.



(d) Influencia de las direcciones no preferidas.

Figura 2.3: Propiedades de la medida (2.4). En negro: vector petición (flecha); recta $x + y = 1$ (línea sólida); y recta $x = y$ (línea rallada). Flechas azules y rojas: vectores documento (una cabeza); su normalización a la recta $x + y = 1$ (dos cabezas); y su proyección sobre el vector petición (tres cabezas). En todos los casos la similitud es la longitud de las flechas con tres cabezas (mayor longitud es mejor; en algunos casos dos cabezas se superponen).

conjuntos de iso-similitud son hiperplanos ortogonales al vector petición (como en la medida (2.2)) cerca del origen y se aproximan a «conos» que parten del origen lejos de él.

La dependencia del tamaño relativo de los vectores petición y documento hace que esta medida sea de utilidad limitada para el *query matching*.

Medidas de covarianza y correlación del producto de momentos

Desplazando vectores documento y petición de manera que su promedio (componente a componente; denotados $\overline{a_d}$ y $\overline{\mathbf{q}}$ respectivamente) sea cero y aplicando las fórmulas de similitud del producto escalar y del coseno a los vectores desplazados se obtienen las medidas de similitud de covarianza y correlación del producto

de momentos, respectivamente:

$$s_d^{\text{PMCov}}(\mathbf{q}) := \sum_{t=1}^{T} (a_{td} - \overline{a_d}) \times (q_t - \overline{\mathbf{q}}); \tag{2.6}$$

$$s_d^{\text{PMCorr}}(\mathbf{q}) := \frac{\sum_{t=1}^{T} (a_{td} - \overline{a_d}) \times (q_t - \overline{\mathbf{q}})}{\sqrt{\sum_{t=1}^{T} (a_{td} - \overline{a_d})^2} \times \sqrt{\sum_{t=1}^{T} (q_t - \overline{\mathbf{q}})^2}}; \tag{2.7}$$

con $\mathbf{q} = (q_1, q_2, \ldots, q_T)^\intercal$ el vector petición y $a_d = (a_{1d}, a_{2d}, \ldots, a_{Td})^\intercal$ el d-ésimo vector documento para $d = 1, 2, \ldots, D$.

Ambas medidas descartan parte de la información de los vectores involucrados, pues «los obligan» a tener promedio cero, reduciendo en uno los grados de libertad de los vectores: por ejemplo, ambas asignan la misma similitud a cualquier vector cuando se compara con $\begin{pmatrix}1 & 2\end{pmatrix}^\intercal$ que cuando se hace con $\begin{pmatrix}2 & 3\end{pmatrix}^\intercal$. Su uso se reduce a situaciones en las que son más significativas las desviaciones del promedio que los valores concretos de las componentes correspondientes.

Usando la medida (2.6), el *signo* (no necesariamente la magnitud) de la similitud entre $\mathbf{q}$ y un vector documento $\mathbf{d}$ depende de si $\mathbf{q} - \overline{\mathbf{q}}$ y $\mathbf{d} - \overline{\mathbf{d}}$ (donde la restas son componente a componente) están en el mismo cuadrante (notar que este último par de vectores está en la bisectriz de los cuadrantes segundo y cuarto). Si y solo si ambos están en el mismo cuadrante, la similitud es positiva; si uno de ellos está en el origen, la similitud es cero (lo que implica que si el vector petición está sobre la bisectriz del primer cuadrante la medida (2.6) asigna similitud cero a cualquier vector documento). En dos dimensiones, la medida (2.7) se comporta de manera similar, aunque únicamente toma los valores cero y más y menos uno.

Generalizando a dimensiones más altas, la medida (2.6) asigna similitud positiva únicamente a aquellos vectores documento $\mathbf{d}$ que forman un ángulo de menos de noventa grados con $\mathbf{q} - \overline{\mathbf{q}}$ (donde $\mathbf{q}$ es el vector petición considerado). Sus contornos de iso-similitud son hiperplanos perpendiculares a $\mathbf{q} - \overline{\mathbf{q}}$.

Los contornos de iso-similitud de (2.7) en dimensiones mayores que dos se puede analizar siguiendo los efectos de los dos pasos de normalización que presenta (véase también la figura 2.4): Primero, los vectores involucrados se envían al hiperplano ortogonal a la diagonal «principal» que pasa por el origen (por ejemplo, en dimensión cuatro, este hiperplano tiene ecuación $x + y + z + w = 0$); después, cada semirrecta que parte del origen se envía al punto contenido en ella que está en la esfera unidad (el origen se mantiene fijo); finalmente, la similitud depende únicamente del ángulo con vértice el origen comprendido entre estos últimos puntos correspondientes a los vectores petición y documento (si uno de ellos es el origen, la similitud es cero). Concluimos que los contornos de iso-similitud son haces de semiplanos paralelos a la diagonal que «trazan» un cono alrededor del punto de la esfera unidad correspondiente al vector petición.

El comportamiento de estas medidas en general es bastante distinto del de las medidas (2.2) y (2.3):

- La medida (2.6) presenta monotonía angular respecto del vector $\mathbf{q} - \overline{\mathbf{q}}$. Para la medida (2.7), la monotonía angular es respecto del plano generado por los vectores $\mathbf{q}$ y $\mathbf{q} - \overline{\mathbf{q}}$.

- Ninguna de las dos medidas presenta monotonía radial: la medida de correlación no cambia según la norma de los vectores involucrados, y la medida de covarianza aumenta en magnitud cuando se aumenta la norma, disminuyendo si el valor de la similitud es negativo.

- Del mismo modo, ninguna de las dos medidas presenta monotonía componente a componente. Aumentar una componente solo aumenta la similitud si disminuye el ángulo con el vector $\mathbf{q} - \overline{\mathbf{q}}$ (para la medida (2.6)) o con el plano $\operatorname{span}(\mathbf{q}, \mathbf{q} - \overline{\mathbf{q}})$ (para la medida (2.7)).

- La influencia de valores grandes de una componente es limitada solo en el caso de la medida (2.7). En este sentido las medidas (2.6) y (2.7) imitan a las medidas (2.2) y (2.3), respectivamente.

- A diferencia de el resto de medidas que detallamos, las medidas de covarianza y correlación sí pueden tomar valores negativos. La similitud mediante covarianza puede tomar cualquier valor real (considerar el vector petición $\begin{pmatrix}1 & 2\end{pmatrix}^\intercal$ y vectores documentos de la forma $\begin{pmatrix}\alpha & 0\end{pmatrix}^\intercal$ y $\begin{pmatrix}0 & \omega\end{pmatrix}^\intercal$ con α y ω reales positivos), mientras que mediante correlación está limitada al intervalo $[-1, 1]$.

Medida de solapamiento

Otra medida de similitud, llamada «de solapamiento», no depende del producto escalar entre los vectores petición y documento, sino que usa la función mínimo:

$$s_d^{\text{ovl}}(\mathbf{q}) := \frac{\sum_{t=1}^{T} \min(a_{td}, q_t)}{\min\left(\sum_{t=1}^{T} a_{td}, \sum_{t=1}^{T} q_t\right)}, \tag{2.8}$$

siendo $\mathbf{q} = (q_1, q_2, \ldots, q_T)^\intercal$ el vector petición y $a_d = (a_{1d}, a_{2d}, \ldots, a_{Td})^\intercal$ el d-ésimo vector documento para $d = 1, 2, \ldots, D$.

Los contornos de iso-similitud de esta medida son sustancialmente distintos a los del resto de medidas que estamos detallando (véase la figura 2.5). Esta medida presenta dos contornos con similitud máxima: un paralelotopo con caras paralelas a los planos coordenados y para el que el vector petición es una diagonal mayor; y un hiperoctante desplazado, también con caras paralelas a los planos coordenados y con vértice el extremo del vector petición. En el primer contorno, todas las coordenadas de los vectores son menores o iguales que las correspondientes en el vector petición; en el segundo, son todas mayores o iguales.

En el resto del espacio (las «regiones de solapamiento parcial») se distinguen dos partes, separadas por el hiperplano ortogonal al vector petición que pasa por su extremo. El semiespacio que queda más cercano al origen presenta normalización en función de la norma uno del vector petición; en el otro semiespacio, la normalización

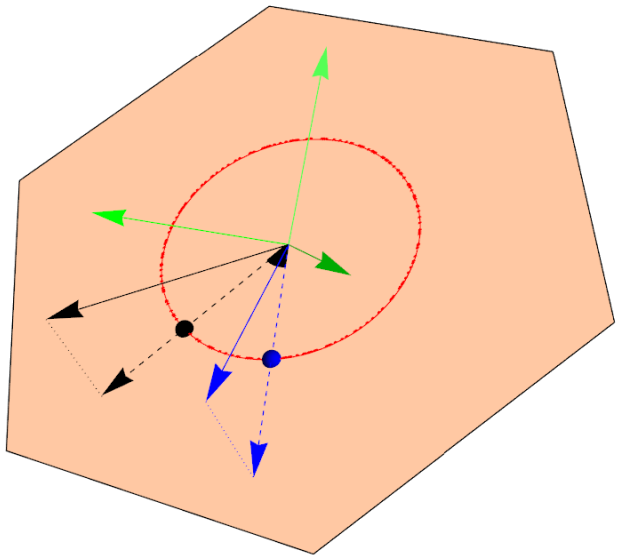


Figura 2.4: Cálculo de la medida (2.7) en tres dimensiones. En verde, los ejes coordenados. En rojo, la restricción de la esfera unidad al plano $x + y + z = 0$; el hexágono naranja marca parte de este último. En negro, vector petición (línea sólida), su proyección al plano $x + y + z = 0$ (línea rallada), y el escalado de este a la esfera unidad (punto sobre esta). En azul, lo mismo para un vector documento. La similitud viene dada por el ángulo marcado en el origen; ángulo más cerrado es mejor.

depende de la norma uno del vector documento considerado y los contornos de isosimilitud son hiperplanos paralelos al plano coordenado más cercano. En ambos casos la medida (2.8) está acotada inferiormente por el valor de la menor componente del vector petición dividida por la norma uno de dicho vector.

Podemos determinar las propiedades de esta medida apoyándonos en las explicaciones anteriores y la figura 2.5:

- Monotonía angular solo en las regiones de solapamiento parcial; fuera de ellas la similitud no varía. El ángulo entre los vectores petición y documento no resulta relevante para (2.8).

- En las regiones de solapamiento parcial la medida (2.8) presenta monotonía radial salvo si el vector tiene una única componente no nula. En el paralelotopo más cercano al origen aumentar la norma terminará por enviar el vector a una región de solapamiento parcial, haciendo que la similitud disminuya temporalmente, hasta llegar al otro paralelotopo, done la norma vuelve a no cambiar la similitud.

- De acuerdo a lo anterior, aumentar una componente de un vector documento puede aumentar o disminuir la similitud respecto de un vector petición dado;

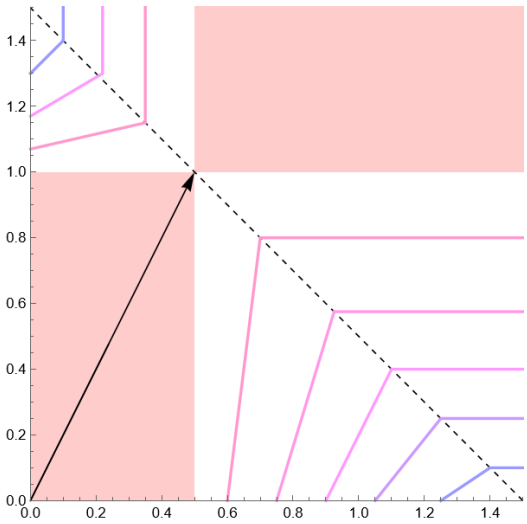


Figura 2.5: Contornos de iso-similitud de la medida (2.8) en dos dimensiones. La flecha representa el vector petición. La línea rallada representa el umbral a partir del cual se divide por la norma uno del vector documento en vez de la del vector petición. Las áreas *no* coloreadas (con fondo blanco) representan las regiones de solapamiento parcial. Las líneas de colores aproximan contornos de iso-similitud.

el comportamiento específico depende de la región en la que se encuentra el vector documento.

- El vector petición impone un límite a la influencia de una componente dada, evitando que la similitud se dispare conforme la componente aumenta.

- Más concretamente, la similitud está acotada superiormente por 1 e inferiormente por $\min_{1 \le t \le T} (q_t) / \|\mathbf{q}\|_1$ donde $\mathbf{q} = (q_1, q_2, \ldots, q_T)^\intercal$ es el vector petición.

Medida de dispersión de activación

Por último, la medida de «dispersión de activación» (*spreading activation measure* en inglés) es, hasta cierto punto, similar al proceso de activación de neuronas (véase el apéndice B) en una red neuronal (usada en el aprendizaje automático y otros sistemas de inteligencia artificial):

Se considera una red con una neurona de entrada, una capa oculta con tantas neuronas como términos hay, y una capa de salida con tantas neuronas como documentos; la función de activación de cada neurona de la capa oculta será la función identidad (o la función ReLU, que para números positivos se comporta igual), dividiendo el resultado por el número de términos. Los pesos de los enlaces en la primera capa son las componentes del vector petición, y los correspondientes a la capa de salida son las componentes de cada vector documento.

En resumen, la «activación» de cada «neurona documento» viene dada por

$$s_d^{\text{spr}}(\mathbf{q}) := \sum_{t=1}^{T} \left(\frac{q_t}{\sum_{s=1}^{T} q_s} \times \frac{a_{td}}{\sum_{e=1}^{D} a_{te}} \right), \tag{2.9}$$

con $\mathbf{q} = (q_1, q_2, \ldots, q_T)^\intercal$ el vector petición y $a_d = (a_{1d}, a_{2d}, \ldots, a_{Td})^\intercal$ el d-ésimo vector documento para $d = 1, 2, \ldots, D$.

Observamos que al igual que la medida (2.4), esta medida normaliza usando la norma uno (asociada a la distancia de Manhattan). Sin embargo, en vez de considerar vectores documento como hace la medida del pseudo-coseno, (2.9) tiene en cuenta la norma de los vectores *término* (notar que la suma en el denominador del segundo factor cambia qué documento se considera cada iteración y no qué término como en (2.4)), lo que reduce la influencia de aquellos términos que aparezcan con mayor valor total en la base de datos.

Los efectos de esta normalización se pueden ver de dos maneras, según como realicemos el cociente en la expresión (2.9). Por un lado, manteniendo el vector petición fijo, los vectores documento son enviados mediante un rescalado al hipercubo con vértice en el origen y lado uno. Por otro lado, podemos reescalar el vector petición, dejando fijos los vectores documento; en este caso el vector petición puede cambiar de dirección ligeramente, apuntando más hacia aquellos términos que aparecen con menos frecuencia o valor en la base de datos. Los contornos de isosimilitud son hiperplanos ortogonales al vector petición, pues tras la normalización de una manera u otra la medida (2.9) se reduce a un producto escalar.

Es notable que, a diferencia de la mayoría de medidas que estamos considerando, la medida (2.9) presenta inconvenientes a la hora de la introducción y modificación de términos puesto que depende de la ponderación relativa de los términos. Esto hace que dos documentos que en un momento dado se consideran igual de similares para un vector petición dado puedan dejar de serlo cuando se modifica, añade, o elimina un tercer documento.

Dejando de lado este inconveniente, algunas características de la medida (2.9) son similares a las de (2.2):

- Monotonía angular respecto del vector petición normalizado: Siguiendo el segundo «modo» de normalización que hemos descrito, la medida (2.9) presenta monotonía angular respecto del vector petición reescalado y no necesariamente respecto del vector petición original.

- Monotonía radial: Al igual que (2.2), esta medida presenta monotonía radial, aumentando o manteniendo igual la similitud conforme la norma de los vectores documento aumenta.

- Monotonía componente a componente: La similitud mediante la medida (2.9) también aumenta o se mantiene igual conforme una componente de un vector documento aumenta.

- Influencia limitada de cada componente: A diferencia de la medida (2.2), la influencia de cada componente de un vector documento está limitada por la

fracción del vector petición que representa el término correspondiente. Por ejemplo, dado el vector petición $\begin{pmatrix}1 & 3\end{pmatrix}^\intercal$, la aportación de la primera componente de un vector documento a la similitud es de a lo sumo un cuarto.

- Similitud acotada: La medida de similitud (2.9) solo toma valores entre cero y uno.

Resumen

	Correlación con			Alteración	
	$\mathbf{q} \angle a_d$	$\|\mathbf{q}\|_2$	q_t	si $q_t \gg 0$	Rango
Producto Escalar (2.2)	$-$	$0+$	$0+$	Ilimitada, positiva	$[0, +\infty)$
Coseno (2.3)	$-$	0[a]	0	Limitada	$[0, 1]$
Pseudo-coseno (2.4)	0[b]	0[a]	0	Limitada	Depende de $\mathbf{q}$
Dice (2.5)	0	$0+$	0	Ilimitada, cualquiera	Depende de $\mathbf{q}$
Covarianza del Prod. de M. (2.6)	0[c]	0[a]	0	Ilimitada, cualquiera	$(-\infty, +\infty)$
Correlación del Prod. de M. (2.7)	0[d]	0	0	Limitada	$[-1, 1]$
Solapamiento (2.8)	$-$[e]	0[e]	0[e]	Limitada	$\left[\min \mathbf{q} / \|\mathbf{q}\|_1, 1\right]$
Dispersión de Activación (2.9)	0	$0+$	$0+$	Limitada	$[0, 1]$

Tabla 2.3: Comparación de las medidas (2.2) a (2.9).

[a] Este parámetro no afecta a la similitud.
[b] Correlación negativa con la dirección preferida más cercana en ángulo.
[c] Correlación negativa con $\mathbf{q} - \overline{\mathbf{q}} \angle a_d - \overline{a_d}$.
[d] Correlación negativa con el plano $\operatorname{span}(\mathbf{q}, \mathbf{q} - \overline{\mathbf{q}})$.
[e] Solo en las regiones de solapamiento parcial; fuera de ellas no hay cambio.

Podemos resumir las propiedades de las medidas de similitud anteriores en forma de tabla a fin de contrastar sus características. En la tabla 2.3 las correlaciones $-$ y $+$ representan que la similitud disminuye o aumenta, respectivamente, conforme la magnitud considerada aumenta; correlación cero significa que la similitud puede aumentar o disminuir; y correlación $0+$ significa que la similitud aumenta o se mantiene igual.

Las columnas que indican correlación muestran los efectos que pueden tener las manipulaciones del vector petición sobre la similitud indicada por cada medida. Por ejemplo, la medida de solapamiento asigna menor similitud conforme el ángulo entre el vector petición y un vector documento aumenta y aumentar la norma del vector petición o de alguno de sus componentes puede aumentar, mantener igual o incluso disminuir la similitud.

La columna «Alteración si $q_t \gg 0$» indica el comportamiento «asintótico» de la medida conforme una componente del vector petición (o un vector documento) toma valores muy grandes.

2.2.2. Implementación

Seguiremos trabajando con el ejemplo descrito en el apartado 2.1.1, sobre la asignación de etiquetas a videojuegos.

Pongamos que un jugador busca recomendaciones sobre juegos RPG que pueda jugar con amigos. Conforme a lo detallado en esta sección, en primer lugar hemos de determinar el vector petición: $\mathbf{q} = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}^T$, donde los unos corresponden a las etiquetas «Multijugador» y «RPG».

```
query = database \
    // First // Keys // AssociationMap[Boole@MemberQ[{"Multijugador", "RPG"}, #] &]
◁"Cartas" → 0, <<2>>, "Multijugador" → 1, <<4>>, "RPG" → 1, "Roguelike" → 0▷
```

Después, calculamos la similitud de $\mathbf{q}$ con cada uno de los vectores documento de nuestra base de datos. A fin de ilustrar las diferencias entre los distintas medidas de similitud detalladas antes usaremos todas; en la práctica se elegiría una medida y se realizarían todos los cálculos con ella.

```
similarity[aDatabase_Association, OptionsPattern[{Method → "Cosine"}]] \
    := Switch[OptionValue[Method],
        "Dot Product", vQuery ⊢> aDatabase // Map[Values /@ (# . vQuery) &],
        "Cosine", vQuery ⊢> aDatabase // Map[1 - Values /@ CosineDistance[#, vQuery] &],
        "Pseudo-Cosine", vQuery ⊢> aDatabase // Map[
            Values /@ (# . vQuery) / Total[#] / Total[vQuery] &
        ],
        "Dice", vQuery ⊢> aDatabase // Map[
            2 * Values /@ (# . vQuery) / (Total[#] + Total[vQuery]) &
        ],
        "Product-Moment Correlation", vQuery ⊢> aDatabase // Map[
            Standardize[Values[#], Mean, 1&] . Standardize[Values[vQuery], Mean, 1&] &
        ],
        "Product-Moment Covariance", vQuery ⊢> aDatabase // Map[
            1 - CosineDistance[
                Standardize[Values[#], Mean, 1&], Standardize[Values[vQuery], Mean, 1&]
            ] &
        ],
        "Overlap", vQuery ⊢> aDatabase // Map[
            Total[Min /@ Transpose[Values /@ {#, vQuery}]] / Min[Total /@ {#, vQuery}]&
        ],
        "Spreading Activation", vQuery ⊢> aDatabase // Map[
            Total @* KeyValueMap[
                vQuery[[#1]] * #2 / Total@Transpose[aDatabase][[#1]] / Total[vQuery] &
            ]
        ]
    ]
```

Como cabe esperar, cada una de las medidas asigna distintas similitudes a cada documento para una misma petición dada:

```
similarity[database, Method → "Dot Product"][query] // Values
{1, 1, 0, <<14>>, 2, 0, 1}

similarity[database, Method → "Cosine"][query] // Values // N
{0.408248, 0.316228, 0., <<14>>, 0.816497, 0., 0.408248}

similarity[database, Method → "Overlap"][query] // Values // N
{0.5, 0.5, 0., <<14>>, 1., 0., 0.5}

similarity[database, Method → "Spreading Activation"][query] // Values // N
{0.0555556, 0.083333, 0., <<15>>, 0., 0.0555556}
```

En este punto podemos usar las listas anteriores para determinar qué documentos se aproximan mejor a la petición. Para ello aplicamos un umbral (determinado por la opción `Threshold`) para filtrar aquellos elementos con menor relevancia, rechazando aquellos documentos cuya similitud sea menor que dicho umbral:

```
searchQuery[aDatabase_Association,OptionsPattern[{Method→"Cosine",Threshold → 0.8}]]\
    := vQuery ⊢>
        similarity[aDatabase, Method → OptionValue[Method]][vQuery] \
            // Select[# > OptionValue[Threshold] &]

searchQuery[database, Method → "Dot Product", Threshold → 1][query] // Keys
{Divinity: Original Sin 2, The Elder Scrolls V: Skyrim}

searchQuery[database, Method → "Cosine", Threshold → 1/2][query] // Keys
{Divinity: Original Sin 2, Iconoclasts, The Elder Scrolls V: Skyrim}

searchQuery[database, Method → "Overlap", Threshold → 1/2][query] // Keys
{Divinity: Original Sin 2, Iconoclasts, The Elder Scrolls V: Skyrim}

searchQuery[database, Method → "Spreading Activation", Threshold → 1/10][query]//Keys
{Divinity: Original Sin 2, The Elder Scrolls V: Skyrim}
```

Esas serían nuestras sugerencias al jugador, que como vemos, dependen de la medida de similitud consdierada.

A modo de ilustración podemos comparar el efecto de distintos umbrales en los valores retornados usando la medida de comparación de cosenos:

```
Manipulate[
    searchQuery[database, Method → "Cosine", Threshold → threshold][query] // Keys,
    {threshold, 0, 1}
]
⟨ threshold = .3 ⟩ {Ara Fell, <<8>>, Wargoove}
⟨ threshold = .5 ⟩ {Divinity: Original Sin 2, Iconoclasts, The Elder Scrolls V: Skyrim}
⟨ threshold = .8 ⟩ {The Elder Scrolls V: Skyrim}
⟨ threshold = .9 ⟩ {}
```

Un último fragmento de código permite comparar el efecto de las distintas medidas de dispersión para un umbral común dado:

```
Manipuplate[
    searchQuery[database, Method → method, Threshold → threshold][query] // Keys,
    {threshold, 0, 1},
    {similarity, {
        "Dot Product", "Cosine", "Pseudo-Cosine", "Dice", "Product-Moment Correlation",
        "Product-Moment Covariance", "Overlap", "Spreading Activation"
    }}
]
```

Documentos enciclopédicos

Trabajando sobre el ejemplo de la sección 2.1.2 podemos encontrar ejemplos de vectores petición para los que el documento enciclopédico «*Changeling Outcast*» *no* resulta ser considerado relevante según algunas medidas de similitud, pese a que, dada la premisa del ejemplo, siempre debería considerarse útil para el jugador.

Si consideramos el vector petición que tiene un uno en la posición correspondiente a «*Druid*», un cinco en la correspondiente a «*Elf*» y un dos en la correspondiente a «*Wizard*» (y cero en el resto), únicamente las medidas (2.2), (2.6) y (2.9) consideran a «*Changeling Outcast*» como la mejor opción, como está reflejado en la tabla 2.4.

Medida	Preferencia	Similitud
Producto Escalar (2.2)	1	10
Coseno (2.3)	8	0,39
Pseudo-coseno (2.4)	8	0,06
Dice (2.5)	7	0,77
Covarianza del Prod. de M. (2.6)	1	1
Correlación del Prod. de M. (2.7)	8	0
Solapamiento (2.8)	8	0,3
Dispersión de la activación (2.9)	1	0,22

Tabla 2.4: Orden de preferencia y similitud del documento enciclopédico «*Changeling Outcast*» para las distintas de medidas de similitud. Menor orden de preferencia y mayor similitud es mejor.

Capítulo 3

Análisis Semántico Latente

En este capítulo usaremos la medida de similitud del coseno (2.3). Si bien esta medida es una de las más usados en la práctica, no es perfecta: para ciertos umbrales «no encuentra» documentos enciclopédicos (en el sentido de que incluyen muchos términos) cuando se le pide buscar documentos en base a una pequeña cantidad de términos. A fin de evitar este y otros problemas similares, se han desarrollado múltiples técnicas que alteran la representación de los datos en la matriz A para retornar documentos más relevantes [3, 4, 18], a saber:

- Como ya mencionamos antes, el uso de ponderaciones globales de términos (lo que denotamos g_t en la sección 2.1) ayuda a compensar disparidades en cuanto a la frecuencia de ocurrencia de los distintos términos.

- El uso de un vocabulario controlado y reducido simplifica las búsquedas mediante la disminución de términos posibles a incluir. Una implementación de este control es usada en el conocido como sistema SMART (véase el apéndice C).

- El reemplazo de la matriz base de datos A por una aproximación con rango reducido. En esta variación se basa el método del análisis semántico latente (LSA, por sus siglas en inglés) y es a la que dedicaremos esta segunda parte del trabajo.

Cuando describimos el modelo basado en espacios vectoriales mencionamos que la información semántica de la matriz base de datos A viene determinada por el espacio vectorial generado por las columnas (vectores documento) de dicha matriz. La dimensión de este espacio vectorial es el rango de la matriz A, que no puede ser mayor que el mínimo de T y D, aunque es posible que sea estrictamente menor.

Cabe esperar que, incluso en el caso que haya más términos que documentos, parte de los vectores documento otorguen información redundante que podemos obtener a partir del resto de vectores documento; en el ejemplo 2.1.1, «Taiji» y «The Witness» tienen las mismas etiquetas, luego la información que aportan es la misma.

Experimentalmente se ha determinado que la reducción del rango de la matriz A mediante pequeñas perturbaciones E (de manera que $\operatorname{rg}(A+E) < \operatorname{rg}(A)$) pro-

duce resultados de igual o mejor calidad que las técnicas «directas» de búsqueda detalladas en el capítulo anterior [6, 3].

Estas perturbaciones podrían a primera vista parecer dañinas, pues alteran los datos contenidos en la base de datos, haciendo que no reflejen la realidad. No obstante, en muchos casos la base de datos por sí misma es inexacta—por ejemplo, por errores de medida, incertidumbre, o desacuerdo entre colaboradores—haciendo que considerar la matriz A como verdad fundamental sea una asunción errónea en general.

En nuestro primer ejemplo hemos asignado al juego «Skyrim» la etiqueta «Multijugador», pese a que no se puede jugar así sin modificar el juego externamente [29]; quizá otro clasificador hubiera dicho que este es motivo suficiente para no asignar esa etiqueta. En el caso de valores «continuos» la posibilidad de datos inciertos o erróneos es aún mayor: ¿hasta qué punto es distinguible la diferencia entre los valores 0.499 y 0.5? ¿Y entre 0.49 y 0.5, o 0.4 y 0.5?

La reducción del rango de la matriz A reduce este «ruido», «simplificando» en cierto modo dicha matriz, y además reduce el coste de las operaciones de búsqueda y adición de documentos, como veremos al final de este capítulo.

Antes de llegar a ello, en este capítulo detallaremos dos maneras de determinar y reducir el rango de A: mediante factorización QR y mediante descomposición en valores singulares. Para cada uno de estos métodos comenzaremos enunciando y probando su existencia y a continuación los aplicaremos al proceso de búsqueda de documentos.

3.1. El LSA y la factorización QR

3.1.1. Definiciones y propiedades

Comenzaremos con algunas definiciones y propiedades que usaremos a lo largo de esta sección.

Denotaremos por $\cdot$ al producto escalar usual; será real o complejo según el contexto.

Si A es una matriz, denotaremos A^* a su transpuesta conjugada, $A^\intercal$ a su transpuesta y $\operatorname{rg}(A)$ a su rango. Llamaremos diagonal principal de una matriz (cuadrada o no) a la banda diagonal de elementos que parte del primer elemento de su primera fila.

Matrices unitarias y ortogonales

Decimos que $Q \in \mathbb{C}^{F\times C}$ (una matriz compleja con F filas y C columnas) tiene columnas ortonormales cuando $Q^*Q = I_C$ con I_C la matriz identidad de tamaño C. Decimos que Q tiene filas ortonormales si cumple $QQ^* = I_F$.

Decimos que $Q \in \mathbb{C}^{n\times n}$ es unitaria si tiene columnas y filas ortonormales*, o equivalentemente, si cumple que $QQ^* = Q^*Q = I$, donde I es la matriz identidad de

tamaño n. Si Q es real y cumple la condición anterior (que en este caso se convierte en $QQ^\intercal = Q^\intercal Q = I$), decimos que es ortogonal.

En base a la definición anterior, una matriz unitaria es invertible y su inversa es su transpuesta conjugada.

Premultiplicar por una matriz unitaria conserva la norma euclídea:

$$\|Q\mathbf{v}\|_2 = \sqrt{(Q\mathbf{v})^* Q\mathbf{v}} = \sqrt{\mathbf{v}^* Q^* Q\mathbf{v}} = \sqrt{\mathbf{v}^* I\mathbf{v}} = \sqrt{\mathbf{v}^*\mathbf{v}} = \|\mathbf{v}\|_2,$$

con $\mathbf{v} \in \mathbb{C}^n$ un vector cualquiera y $Q \in \mathbb{C}^{n\times n}$ unitaria. Más generalmente se cumple también que $\mathbf{x}\cdot\mathbf{y} = (Q\mathbf{x})\cdot(Q\mathbf{y})$, con una demostración análoga.

Dado que $\det Q = \det Q^*$, se tiene que el determinante de una matriz unitaria tiene módulo uno. En el caso real, el determinante de una matriz ortogonal es más o menos uno.

Matrices de permutación

Decimos que una matriz es de permutación cuando exactamente un elemento de cada fila y columna es distinto de cero y esos elementos son iguales a uno.

Es inmediato que una matriz de permutación es cuadrada y, de hecho, es ortogonal.

Cada matriz de permutación se construye permutando las columnas (o filas) de la matriz identidad: Si consideramos una permutación de n elementos cualquiera, π, la matriz de permutación correspondiente tiene entradas $([i = \pi(j)])_{1\le j\le n}^{1\le i\le n}$ donde los corchetes son los conocidos como corchetes de Iverson [11] (valen uno si la condición es cierta y cero si no) e i recorre la filas de arriba hacia abajo y j, las columnas de izquierda a derecha. En particular, hay $n!$ matrices de permutación de tamaño n.

A partir de esta biyección entre permutaciones y matrices de permutación podemos deducir algunas propiedades:

- La matriz de permutación correspondiente a la permutación identidad es la identidad.
- El producto de dos matrices de permutación también es una matriz de permutación y su permutación correspondiente es la composición de las permutaciones correspondientes a las dos primeras matrices en el mismo orden.
- Más concretamente, el conjunto de matrices de permutación forma un grupo con la operación de producto matricial, con aplicación inversa la transposición de matrices.
- Postmultiplicar por una matriz de permutación es equivalente a permutar las columnas de acuerdo a la permutación que corresponde a la matriz según la biyección anterior.
- Análogamente, premultiplicar es equivalente a permutar las *filas* de acuerdo a la *inversa* de la permutación correspondiente a la matriz.

*La condición de tener filas y columnas ortonormales es necesaria pero no suficiente para ser unitaria: hace falta que la matriz sea cuadrada también para ser unitaria.

Matrices triangulares y diagonales

Llamamos ancho de banda inferior [9] de una matriz $A \in \mathbb{C}^{F\times C}$ con entradas $\left(a_{ij}\right)_{1\leq j\leq C}^{1\leq i\leq F}$ (con i recorriendo filas de arriba a abajo y j, las columnas de izquierda a derecha) al menor entero no negativo l para el que $i > j + l$ implica que $a_{ij} = 0$. Llamamos ancho de banda superior de A al menor entero no negativo u para el que $j > i + u$ implica que $a_{ij} = 0$.

En otras palabras, el ancho de banda inferior (resp. superior) de una matriz indica cuántas bandas diagonales bajo (resp. sobre) la diagonal principal contienen elementos no nulos.

Por ejemplo, la siguiente matriz tiene ancho de banda inferior uno y superior dos:

$$\begin{pmatrix} 1 & 2 & 3 & 0 \\ 5 & 0 & 0 & 0 \\ 0 & 10 & 11 & 12 \\ 0 & 0 & 0 & 16 \\ 0 & 0 & 0 & 20 \end{pmatrix}$$

Decimos que una matriz $A \in \mathbb{C}^{F\times C}$ (con F no necesariamente igual a C) es triangular superior cuando su ancho de banda inferior es cero. Análogamente, decimos que una matriz es triangular inferior cuando su ancho de banda superior es cero, o equivalentemente, cuando su transpuesta es triangular superior.

Decimos que una matriz es diagonal cuando sus anchos de banda inferior y superior son cero, o equivalentemente cuando es al mismo tiempo triangular superior y triangular inferior.

Las definiciones anteriores engloban matrices triangulares y diagonales no cuadradas; una matriz triangular superior (resp. inferior) con más filas que columnas (resp. columnas que filas) tiene una cantidad de filas (resp. columnas) de ceros igual o superior a la diferencia.

Reflector de Householder

Dado un vector no nulo cualquiera $\mathbf{u} \in \mathbb{C}^n \setminus \{0\}$, llamamos reflector de Householder asociado a $\mathbf{u}$ a la matriz $H(\mathbf{u}) \in \mathbb{C}^{n\times n}$ dada por

$$H(\mathbf{u}) = I - \frac{2}{\|\mathbf{u}\|_2{}^2} (\mathbf{u}\mathbf{u}^*)$$

con I la matriz identidad de tamaño n. Las entradas de $H(\mathbf{u})$ son reales si lo son las de $\mathbf{u}$.

En base a la definición anterior se tiene que $H(\mathbf{u})$ es hermítica (simétrica en el caso real) y es fácil comprobar que es su propia inversa, luego es unitaria.

Los reflectores de Householder $H(\mathbf{u})$ no modifican aquellas componentes que corresponden a ceros en el vector $\mathbf{u}$. En el caso de premultiplicar una matriz por un reflector de Householder, las filas de la matriz original correspondientes no son modificadas; si consideramos postmultiplicación, son las columnas las que no se modifican.

Teorema 3.1.1 *Si* $\mathbf{x}$ *y* $\mathbf{y}$ *son dos vectores no nulos de* $\mathbb{C}^n$ *distintos y con igual norma euclídea, entonces* $H(\mathbf{x}-\mathbf{y})\mathbf{x} = \mathbf{y}$ *si y solo si* $\mathbf{x}\cdot\mathbf{y} \in \mathbb{R}$.

Demostración: Podemos escribir $\mathbf{x} = (\mathbf{x}-\mathbf{y})/2 + (\mathbf{x}+\mathbf{y})/2$ y se tiene que

$$(\mathbf{x}-\mathbf{y})\cdot(\mathbf{x}+\mathbf{y}) = \|\mathbf{x}\|_2{}^2 - (\mathbf{y}\cdot\mathbf{x}) + (\mathbf{x}\cdot\mathbf{y}) - \|\mathbf{y}\|_2{}^2 = \mathbf{x}\cdot\mathbf{y} - \overline{\mathbf{x}\cdot\mathbf{y}} = 2\mathbb{i}\ \mathrm{Im}(\mathbf{x}\cdot\mathbf{y})$$

por la linealidad del producto escalar y el hecho de que $\mathbf{x}$ e $\mathbf{y}$ tienen la misma norma.

Supongamos primero que se tiene $\mathbf{x}\cdot\mathbf{y} \in \mathbb{R}$ de modo que $2\mathbb{i}\ \mathrm{Im}(\mathbf{x}\cdot\mathbf{y}) = 0$. A partir de lo anterior y la definición de reflector se tiene

$$\begin{aligned}
H(\mathbf{x}-\mathbf{y})\,\mathbf{x} &= \left(I - \frac{2}{\|\mathbf{x}-\mathbf{y}\|_2{}^2}(\mathbf{x}-\mathbf{y})(\mathbf{x}-\mathbf{y})^*\right)\mathbf{x} \\
&= \left(I - \frac{2}{\|\mathbf{x}-\mathbf{y}\|_2{}^2}(\mathbf{x}-\mathbf{y})(\mathbf{x}-\mathbf{y})^*\right)\frac{(\mathbf{x}-\mathbf{y})+(\mathbf{x}+\mathbf{y})}{2} \\
&= \left(I - 2\frac{\mathbf{x}-\mathbf{y}}{\|\mathbf{x}-\mathbf{y}\|_2{}^2}(\mathbf{x}-\mathbf{y})^*\right)\frac{\mathbf{x}-\mathbf{y}}{2} + \left(I - 2\frac{\mathbf{x}-\mathbf{y}}{\|\mathbf{x}-\mathbf{y}\|_2{}^2}(\mathbf{x}-\mathbf{y})^*\right)\frac{\mathbf{x}+\mathbf{y}}{2} \\
&= \frac{\mathbf{x}-\mathbf{y}}{2} - (\mathbf{x}-\mathbf{y}) + \frac{\mathbf{x}+\mathbf{y}}{2} - \left(\frac{1}{\|\mathbf{x}-\mathbf{y}\|_2{}^2}(\mathbf{x}-\mathbf{y})\underbrace{(\mathbf{x}-\mathbf{y})^*(\mathbf{x}+\mathbf{y})}_{=0}\right) \\
&= \frac{\mathbf{x}-\mathbf{y}}{2} + \frac{\mathbf{x}+\mathbf{y}}{2} \\
&= \mathbf{y}
\end{aligned}$$

Si suponemos que $\mathbf{x}\cdot\mathbf{y} \notin \mathbb{R}$, la cadena de igualdades anterior conduce a

$$\begin{aligned}
H(\mathbf{x}-\mathbf{y})\,\mathbf{x} &= \frac{\mathbf{x}-\mathbf{y}}{2} - (\mathbf{x}-\mathbf{y}) + \frac{\mathbf{x}+\mathbf{y}}{2} - \left(\frac{1}{\|\mathbf{x}-\mathbf{y}\|_2{}^2}(\mathbf{x}-\mathbf{y})(\mathbf{x}-\mathbf{y})^*(\mathbf{x}+\mathbf{y})\right) \\
&= \mathbf{y} - \left(\frac{2\mathbb{i}\ \mathrm{Im}(\mathbf{x}\cdot\mathbf{y})}{\|\mathbf{x}-\mathbf{y}\|_2{}^2}\underbrace{(\mathbf{x}-\mathbf{y})}_{\neq 0}\right) \\
&\neq \mathbf{y}
\end{aligned}$$

□

En particular, para todo $\mathbf{x} \in \mathbb{C}^n \setminus \{0\}$ se tiene que $H(\mathbf{x} - \|\mathbf{x}\|_2\omega e_1)\ \mathbf{x} \propto e_1$ donde e_1 es el primer vector de la base canónica y $\omega = \pm\mathbf{x}_1/|\mathbf{x}_1|$ con $\mathbf{x}_1$ la primera componente de $\mathbf{x}$ (si $\mathbf{x}_1 = 0$, tomamos cualquier ω con módulo uno).

3.1.2. Existencia de la factorización QR

Teorema 3.1.2 *Cualquier matriz $A \in \mathbb{C}^{n\times n}$ se puede expresar como un producto de dos matrices, $A = QR$, donde $Q \in \mathbb{C}^{n\times n}$ es unitaria y $R \in \mathbb{C}^{n\times n}$ es triangular superior.*

Demostración: Lo probaremos de manera constructiva.

Asignaremos a Q y R valores iniciales $Q \leftarrow I_n$ y $R \leftarrow A$ que sobreescribiremos más adelante, donde I_n es la matriz identidad de tamaño n. Buscamos hacer que R sea triangular superior, que Q siga siendo unitaria, y que se siga teniendo la igualdad $A = QR$.

Ya hemos visto que podemos encontrar un reflector de Householder que transforma la primera columna de R en un vector proporcional al primer vector de la base canónica (a menos que esa columna sea el vector nulo, en cuyo caso tomamos como reflector la matriz identidad).

Aplicamos dicha transformación a R, premultiplicando por el reflector de Householder H_1 ($R \leftarrow H_1 R$), y «actualizamos» Q, *post*multiplicando por H_1 ($Q \leftarrow QH_1$). Esto hace que la primera columna de R sea ceros salvo (posiblemente) en la primera posición y hace que Q siga siendo una matriz unitaria; esencialmente hemos cambiado la posición de los paréntesis en $A = I_n (H_1 H_1) A$ (recordemos que H_1 es su propia inversa), teniendo en su lugar $A = (I_n H_1)(H_1 A)$.

A partir de este punto la primera fila y la primera columna de R no se van a modificar y podemos ignorarlas en lo que sigue. Consideramos la siguiente columna de R (un vector de $n-1$ elementos, no n como antes pues no tenemos en cuenta el primer elemento) y determinamos un reflector de Householder H_2 que lo envía a un vector proporcional al primer vector de la base canónica como hemos hecho antes.

El reflector H_2 es una matriz de tamaño $(n-1)\times(n-1)$ y por tanto no podemos multiplicar R y Q por él. Lo que sí podemos hacer es «ampliarlo», añadiendo una matriz identidad en la «esquina superior izquierda», para que esas multiplicaciones sean posibles; esto es equivalente a añadir ceros antes del vector que genera el reflector de Householder.

La multiplicación de R por el reflector H_2 ampliado $\widehat{H_2}$ ($R \leftarrow \widehat{H_2} R$) no modifica ni la primera fila ni la primera columna de R y además hace ceros a partir de la tercera posición de la segunda columna. Asimismo la multiplicación de Q por el mismo reflector ($Q \leftarrow Q\widehat{H_2}$) hace que Q siga siendo unitaria.

El proceso anterior se repetiría hasta recorrer todas las columnas.

En resumen, el algoritmo busca los reflectores $H_1, H_2, \ldots, H_{n-1}$ de modo que se tenga la siguiente igualdad donde Q y R satisfacen las condiciones pedidas:

$$A = \underbrace{I_n H_1 \begin{pmatrix} I_1 & 0 \\ 0 & H_2 \end{pmatrix} \begin{pmatrix} I_2 & 0 \\ 0 & H_3 \end{pmatrix} \cdots \begin{pmatrix} I_{n-2} & 0 \\ 0 & H_{n-1} \end{pmatrix} \ddots}_{Q}$$
$$\underbrace{\begin{pmatrix} I_{n-2} & 0 \\ 0 & H_{n-1} \end{pmatrix} \begin{pmatrix} I_{n-3} & 0 \\ 0 & H_{n-2} \end{pmatrix} \cdots \begin{pmatrix} I_2 & 0 \\ 0 & H_3 \end{pmatrix} \begin{pmatrix} I_1 & 0 \\ 0 & H_2 \end{pmatrix} H_1 A}_{R}$$

□

Es claro que si $A = QR$ con A, Q y R como en el teorema anterior, y A tiene entradas reales, entonces Q y R también (en particular, Q es ortogonal).

Para matrices rectangulares con más filas que columnas se tiene un teorema similar:

Teorema 3.1.3 *Toda matriz $A \in \mathbb{C}^{F\times C}$ tal que $F > C = \operatorname{rg}(A)$ se puede expresar como un producto de dos matrices, $A = QR$, donde $Q \in \mathbb{C}^{F\times F}$ es unitaria y $R \in \mathbb{C}^{F\times C}$ se puede dividir por bloques como*

$$R = \begin{pmatrix} R_\triangle \\ 0 \end{pmatrix}$$

con $R_\triangle \in \mathbb{C}^{C\times C}$ triangular superior.

Demostración: Añadimos $F-C$ columnas «a la derecha» de A, obteniendo la matriz $\widehat{A} \in \mathbb{C}^{F\times F}$; los elementos añadidos de esta manera no son relevantes y pueden ser elegidos de manera arbitraria.

Aplicamos el teorema anterior a $\widehat{A}$, obteniendo las matrices Q y $\widehat{R}$ con $\widehat{A} = Q\widehat{R}$, Q unitaria y $\widehat{R}$ triangular superior de tamaño $F \times F$.

Definiendo R como la matriz formada por las C primeras columnas de $\widehat{R}$ en orden tenemos que R es como en el enunciado y se cumple $A = QR$ por construcción. □

En la práctica no hace falta expandir A sino que se aplica el algoritmo usado para matrices cuadradas, iterando hasta que «nos quedemos sin columnas».

Finalmente, en el caso más general tenemos el siguiente teorema:

Teorema 3.1.4 *Toda matriz $A \in \mathbb{C}^{F\times C}$ se puede expresar como un producto de tres matrices, $A = QRP^\intercal$, donde P es una matriz de permutación de tamaño C, $Q \in \mathbb{C}^{F\times F}$ es unitaria y $R \in \mathbb{C}^{F\times C}$ es triangular superior y se puede dividir por bloques*

$$R = \begin{pmatrix} R_\triangle & R_\square \\ 0 & 0 \end{pmatrix}$$

de manera que $R_\triangle$ es cuadrada de tamaño $\operatorname{rg}(A)$, triangular superior e invertible.

Dicho producto se conoce como factorización QR de A y en ocasiones se escribe como $AP = QR$.

De nuevo, la demostración se basa en el algoritmo usado para matrices cuadradas, con el siguiente cambio:

En cada iteración del proceso, si nos encontramos con que hemos de calcular el reflector de Householder correspondiente al vector nulo, en vez de hacerlo intercambiamos («pivotamos») la columna «actual» de R con alguna columna que tenga alguna componente no nula (si no la hay, lo que resta de R es la matriz nula y hemos terminado), e intercambiamos las columnas correspondientes de P.

En general la factorización QR no es única: el resultado depende de qué columna se escoge a la hora de pivotar y de la elección del signo de ω a la hora de calcular los reflectores. En la práctica, se pivota en todas las iteraciones con la columna con

mayor norma (ignorando aquellas filas ya modificadas) y, en coma flotante, las elecciones de los reflectores se hacen a fin de evitar cancelaciones catastróficas. Pivotar las columnas de esta manera hace que los elementos de la diagonal principal de la matriz R estén ordenados de mayor a menor módulo.

La implementación del algoritmo se resume en un bucle que va actualizando las matrices Q, R y P de acuerdo a lo anterior:

```
qrDecomposition = mA ⊢> Module[
    {
        mQ = IdentityMatrix[First[Dimensions[mA]]], mR = mA,
        mP = IdentityMatrix[Last[Dimensions[mA]]],
        vNormsSquared = Transpose[mA] // Map[(# . #) &],
        iPermIndex, mReflector
    },
    Do[
        iPermIndex = Max[First@Ordering[vNormsSquared,-1], j];
        If[PossibleZeroQ[vNormsSquared[[iPermIndex]]], Break[]];
        mR[[All, {j, iPermIndex}]] = mR[[All, {iPermIndex, j}]];
        mP[[All, {j, iPermIndex}]] = mP[[All, {iPermIndex, j}]];
        vNormsSquared[[{j, iPermIndex}]] = vNormsSquared[[{iPermIndex, j}]];
        mReflector = mR[[All, j]]
            // MapIndexed[Switch[Sign[First[#2] - j],
                -1, 0,
                0, #1 + (Sign[#1]+Boole[PossibleZeroQ[#1]]) * Sqrt[vNormsSquared[[j]]],
                1, #1
            ] &]
            // ReflectionMatrix;
        mR = mReflector . mR;
        mQ = mQ . mReflector;
        vNormsSquared -= mR[[j, All]]^2
    , {j, Min[Dimensions[mA]]}];
    {mQ, mR, mP} // Map[FullSimplify]
]
```

Con la notación del teorema 3.1.4, podemos descomponer Q por bloques de manera similar a R de manera que

$$AP = QR = \begin{pmatrix} Q_{\|} & Q_{\perp} \end{pmatrix} \begin{pmatrix} R_{\triangle} & R_{\square} \\ 0 & 0 \end{pmatrix} = Q_{\|} \begin{pmatrix} R_{\triangle} & R_{\square} \end{pmatrix},$$

con $Q_{\|}$ de tamaño $\operatorname{rg}(A) \times F$, y $Q_{\perp}$ de tamaño $(C - \operatorname{rg}(A)) \times F$. La igualdad entre los extremos de esta cadena de igualdades se conoce como factorización QR compacta de A (y es de hecho, la descomposición que realiza la función `QRDecomposition` en Mathematica).

Las columnas de $Q_{\|}$ forman una base del espacio columna de A. Si $\operatorname{rg}(A) = F$ (lo que implica $C \geq F$), la matriz $Q_{\perp}$ y los dos bloques de ceros en el factor R no aparecen, y si $\operatorname{rg}(A) = C$ (luego $F \geq C$) entonces es la matriz $R_{\square}$ y el bloque de ceros bajo ella los que no aparecen.

3.1.3. Aplicación de la factorización QR a la búsqueda de documentos

Volviendo sobre la búsqueda de documentos, consideraremos la descomposición QR de la matriz de documentos y términos A, que como hemos visto anteriormente resulta en la igualdad $AP = QR$ donde Q es ortogonal pues A es una matriz real. El efecto de la matriz de permutación P sobre A en nuestro contexto se traduce en reordenar los documentos de la base de datos, por lo que para simplificar la notación «absorberemos» dicha reordenación y pondremos simplemente A incluso cuando P no sea la matriz identidad.

Teniendo en cuenta que por ser Q ortogonal se tiene que

$$I_T = QQ^\intercal = \begin{pmatrix} Q_\parallel & Q_\perp \end{pmatrix} \begin{pmatrix} Q_\parallel^\intercal \\ Q_\perp^\intercal \end{pmatrix} = Q_\parallel Q_\parallel^\intercal + Q_\perp Q_\perp^\intercal,$$

podemos escindir un vector petición cualquiera $\mathbf{q}$ en su proyección sobre el espacio columna de Q, $q_\parallel$, y su parte ortogonal a él, $q_\perp$:

$$\mathbf{q} = I_T \mathbf{q} = (QQ^\intercal)\,\mathbf{q} = \left(Q_\parallel Q_\parallel^\intercal + Q_\perp Q_\perp^\intercal\right)\mathbf{q} = \underbrace{Q_\parallel Q_\parallel^\intercal \mathbf{q}}_{q_\parallel} + \underbrace{Q_\perp Q_\perp^\intercal \mathbf{q}}_{q_\perp}.$$

La descomposición anterior conduce a expresión alternativa de la expresión (2.3):

$$\begin{aligned} s_d^{\cos}(\mathbf{q}) &= \frac{a_d \cdot \mathbf{q}}{\|a_d\|_2 \|\mathbf{q}\|_2} \\ &= \frac{a_d \cdot q_\parallel + a_d \cdot q_\perp}{\|a_d\|_2 \|\mathbf{q}\|_2} \\ &= \frac{a_d \cdot q_\parallel + a_d^\intercal Q_\perp Q_\perp^\intercal q_\perp}{\|a_d\|_2 \|\mathbf{q}\|_2} \\ &= \frac{a_d \cdot q_\parallel + 0 \times Q_\perp^\intercal q_\perp}{\|a_d\|_2 \|\mathbf{q}\|_2} \\ &= \frac{a_d \cdot q_\parallel}{\|a_d\|_2 \|\mathbf{q}\|_2}, \end{aligned}$$

donde en la penúltima línea hemos usado que a_d y $Q_\perp$ son ortogonales por construcción.

Esta expresión alternativa no aporta beneficios notables en cuanto a coste computacional (pues requiere factorizar la matriz A para calcular $q_\parallel$), pero sí tiene relevancia desde el punto de vista de la búsqueda de información: la parte de $\mathbf{q}$ que queda fuera del espacio columna de A no contribuye a la hora de calcular el producto escalar usado en la medida (2.3).

El argumento anterior hace pensar que podríamos «ignorar» completamente $q_\perp$, reemplazando la petición «imperfecta» $\mathbf{q}$ por su mejor aproximación en el espacio columna de A, $q_\parallel$. Para ver que $q_\parallel$ es la mejor aproximación, tomamos $\mathbf{x}$ un vector del espacio columna cualquiera, de modo que

$$\begin{aligned} \|\mathbf{q} - \mathbf{x}\|_2^{\,2} &= \|\mathbf{q} - q_\parallel + q_\parallel - \mathbf{x}\|_2^{\,2} \\ &= \|\mathbf{q} - q_\parallel\|_2^{\,2} + \|q_\parallel - \mathbf{x}\|_2^{\,2} \end{aligned}$$

por ser $\mathbf{q} - q_\parallel = q_\perp$ ortogonal a $q_\parallel - \mathbf{x}$. El mínimo se obtiene cuando $\mathbf{x} = q_\parallel$.

Definimos entonces

$$s_d^{\cos,\,\mathrm{proj}}(\mathbf{q}) := \frac{a_d \cdot q_\parallel}{\|a_d\|_2 \|q_\parallel\|_2}. \tag{3.1}$$

Observamos que para $\mathbf{q}$ dado, (2.3) y (3.1) verifican la relación

$$\frac{s_d^{\cos}(\mathbf{q})}{s_d^{\cos,\,\mathrm{proj}}(\mathbf{q})} = \frac{\|q_\parallel\|_2}{\|\mathbf{q}\|_2},$$

siendo el lado derecho menor o igual que uno como consecuencia del teorema de Pitágoras aplicado a $\mathbf{q} = q_\parallel + q_\perp$. En el caso concreto en que $\mathrm{rg}\,(A) = T$, como en el ejemplo 2.1.1, se tiene que $\mathbf{q} = q_\parallel$ y entonces $\forall d.\ s_d^{\cos}(\mathbf{q}) = s_d^{\cos,\,\mathrm{proj}}(\mathbf{q})$.

Por tanto, la medida (3.1) tiende a considerar relevantes más documentos que la medida (2.3), aunque no necesariamente sean todos verdaderamente relevantes. En la jerga de la búsqueda de información, usar (3.1) en vez de (2.3) aumenta la exhaustividad (*recall*) a costa de arriesgarse a reducir la precisión.

3.1.4. Implementación

Podemos implementar la medida (3.1) como otro caso en la función de cálculo de similaridad que ya teníamos:

```
similarity[aDatabase_Association, OptionsPattern[{Method → "Cosine"}]]
    := Switch[OptionValue[Method],
        (* ... *)
        "QR Projection", Module[{projectionFactor =
                qrDecomposition[Transpose[aDatabase] // Values // Map[Values] // N]
                    // Chop // (AssociationThread[#[[1]] // Transpose, #[[2]]])&
                    // Select[AnyTrue[Not @* PossibleZeroQ]] // Keys // Transpose
                    // # . Transpose[#] & // Chop
            },
            vQuery ⊢> aDatabase // Map[
                1 - CosineDistance[Values[#], projectionFactor . Values[vQuery]] &]
        ]
    ]
```

La implementación de la medida de similitud (3.1) resulta notablemente distinta al resto, pero en esencia es similar a la implementación que hicimos en su momento de la medida (2.3).

Volviendo al ejemplo 2.1.1 del capítulo anterior, como la matriz de la base de datos tiene rango máximo, las medidas (2.3) y (3.1) toman los mismos valores:

```
similarity[database, Method → "QR Projection"][query] // Values // N
{0.408248, 0.316228, 0, 0, 0.707107, <<10>>, 0, 0.5, 0.816497, 0, 0.408248}
similarity[database, Method → "Cosine"][query] // Values // N
{0.408248, 0.316228, 0, 0, 0.707107, <<10>>, 0, 0.5, 0.816497, 0, 0.408248}
```

En el ejemplo 2.1.2, este no es el caso, y en efecto la medida (3.1) considera la mayoría de vectores un poco más similares:

```
similarity[database2, Method → "QR Projection"][query2] // Values // Chop
{0, 0.266272, 0.266272, 0.376565, 0.266272, 0.470706, 0.652230, 0, <<3>>, 0.543525}
similarity[database2, Method → "Cosine"][query2] // Values // N // Chop
{0, 0.258199, 0.258199, 0.365148, 0.258199, 0.456435, 0.632456, 0, <<3>>, 0.527046}
```

Hasta ahora hemos implementado las medidas de similitud usando *curried functions** sin justificar por qué. La razón es que el paso intermedio, en el que solo aplicamos la base de datos, permite preprocesar los datos para realizar las búsquedas más eficientemente, evitando recalcular datos que no dependen de los vectores de búsqueda. Las medidas vistas anteriormente no se beneficiaban de este paso intermedio, pero la medida (3.1) sí lo hace: el factor de proyección $Q_{\|}Q_{\|}^{\mathsf{T}}$ (`projectionFactor` en la implementación anterior) solo depende de los vectores documento y por tanto no hace falta calcularlo repetidamente para cada vector petición.

Este paso intermedio permitiría también la modificación de la matriz intermedia para la adición, modificación y borrado de documentos y términos (sección 3.3), aunque la implementación anterior no permita hacerlo directamente.

Observamos que tanto (2.3) como (3.1) son casos particulares de otra medida de similitud, dada por

$$s_d^{\cos,\mathrm{gen}}(M)(\mathbf{q}) := \frac{v_d \cdot (M\mathbf{q})}{\|v_d\|_2 \|M\mathbf{q}\|_2} \tag{3.2}$$

donde M es una matriz $T \times T$ fija y v_d es un vector T elementos que varía con cada documento. Si en la expresión anterior tomamos $M = I_T$ (la matriz identidad de tamaño $T \times T$) y $v_d = a_d$ tenemos la medida (2.3) y si tomamos $M = Q_{\|}Q_{\|}^{\mathsf{T}}$ y $v_d = a_d$, la (3.1).

De cara a la implementación, podemos almacenar los vectores v_d en forma matricial y simplemente acceder a ellos para calcular (3.2). Por ahora nos limitaremos a implementar la medida (3.1):

```
prepareQrState[aDatabase_Association] :=
    qrDecomposition[Transpose[dDatabase] // Values // Map[Values] // N] // Chop
        // AssociationThread[#[[1]] // Transpose, #[[2]]] & \
        // Select[AnyTrue[Not @* PossibleZeroQ]] // Keys \
        // Transpose // {aDatabase // Values // Map[Values], Transpose[#] . # // Chop}&

generalizedCosineSimilarity = {lmDocumentStates, mProjectionFactor, vQuery} ⊢> \
    If[MatrixQ[lmDocumentStates], lmDocumentStates, Fold[Dot, lmDocumentStates]] \
    // Map[1 - CosineDistance[#, mProjectionFactor . Values[vQuery]] &]
```

Podemos comprobar que esta implementación produce las mismas similitudes que la anterior:

```
similarity[database, Method → "QR Projection"][query]
        - generalizedCosineSimilarity[Sequence @@ prepareQrState[database], query]
    // Map[Normal] // Values // Chop // AllTrue[PossibleZeroQ]
True
```

*Una *curried function* [26] es una función de varios argumentos que los toma de uno en uno, en vez de todos a la vez, permitiendo la aplicación parcial de parte de ellos. En otras palabras, la versión currificada de una función $f : X \times Y \to Z$ es de la forma $f : X \to (Y \to Z)$ y se evaluaría mediante $f(x)(y)$ en vez de $f(x, y)$. En ocasiones una *curried function* con tres o más argumentos puede tomar dos o más de ellos a la vez en vez de de uno en uno. Ver también la función `CurryApplied` en Mathematica.

3.1.5. Reducción de Rango

Iniciamos el capítulo diciendo que ciertas pequeñas perturbaciones a la matriz A corrigen hasta cierto punto la incertidumbre de los datos contenidos en ella y pueden reducir la dimensión del espacio columna de A. ¿Qué significa para nosotros perturbación «pequeña»?

En este contexto, la cualidad a la que nos referimos es la norma matricial de Frobenius, que esencialmente es la norma euclídea de las componentes de la matriz: si denotamos por e_{td} las componentes de la matriz perturbación E, su norma de Frobenius viene dada por

$$\|E\|_F = \sqrt{\sum_{t=1}^{T}\sum_{d=1}^{D} {e_{td}}^2} = \sqrt{\mathrm{Tr}(EE^\intercal)} = \sqrt{\mathrm{Tr}(E^\intercal E)}$$

donde Tr es la función traza de una matriz (la suma de las componentes de la diagonal).

A partir de la última expresión se deduce que tanto pre- como postmultiplicar por una matriz ortogonal conserva la norma de Frobenius, esto es, si U y V son unitarias de dimensiones apropiadas, se tiene que $\|UE\|_F = \sqrt{\mathrm{Tr}(E^\intercal U^\intercal UE)} = \sqrt{\mathrm{Tr}(E^\intercal E)} = \|E\|_F$ y que $\|EV\|_F = \sqrt{\mathrm{Tr}(EVV^\intercal E^\intercal)} = \sqrt{\mathrm{Tr}(EE^\intercal)} = \|E\|_F$. Aplicando esto a la factorización QR descrita antes, si $A = QRP^\intercal$ tenemos que $\|A\|_F = \|R\|_F$, luego cualquier cambio que altere la norma de R cambia la norma de A de la misma manera.

Un cambio comparativamente pequeño que podemos hacer es anular parte de R. Puesto que estamos usando un algoritmo que ordena los elementos diagonales de R de mayor a menor módulo, anularemos las últimas filas dado que así se producen menores cambios en cuanto a norma y se preserva el hecho de que las últimas filas (y solo las últimas filas) de R son nulas.

Volviendo al ejemplo de la sección 2.1.1, la norma de Frobenius de la matriz R (véase la figura 3.1) es aproximadamente 8 y la de la parte contenida sobre la línea marcada (las ocho primeras filas) es aproximadamente 7,81. Haciendo ceros en las dos últimas filas, conseguimos reducir el rango en dos sin apenas alterar la norma de A.

En nuestra implementación, dado que solo nos importa la matriz Q, hacer ceros en R es equivalente a reducir el número de columnas de $Q_\parallel$ en uno:

```
prepareQrState[aDatabase_Association, OptionsPattern[{MatrixRank → Automatic}]] := \
    qrDecomposition[Transpose[aDatabase] // Values // Map[Values] // N] // Chop \
        // AssociationThread[#[[1]] // Transpose, #[[2]]] & \
        // Select[AnyTrue[Not @* PossibleZeroQ]] // Keys \
        // Replace[OptionValue[MatrixRank], { \
            Automatic | 0 → Identity,
            n: x_Integer | UpTo[x_Integer] /; Positive[x] → (Take[#, n] &)
            n_Integer /; Negative[n] → (Drop[#, n] &)
        }] // Transpose // {Values[aDatabase] // Map[Values], Transpose[#].# // Chop} &
```

Las diferencias de similitud al reducir el rango en uno son relativamente pequeñas:

```
generalizedCosineSimilarity[Sequence@@prepareQrState[database,MatrixRank → -1],query]\
    - generalizedCosineSimilarity[Sequence @@ prepareQrState[database], query]
{-0.0523224, 0.00585055, 0.0154598, <<15>>, 0, -0.0523224}
```

$$\left(\begin{array}{cccccccccc|ccc}
-2{,}24 & -0{,}89 & -0{,}89 & -0{,}45 & 0 & -0{,}45 & -0{,}89 & -1{,}34 & 0 & -0{,}89 & -0{,}89 & \ll 8 \gg & -0{,}89 \\
0 & -2{,}05 & -0{,}1 & -0{,}29 & -0{,}98 & -1{,}27 & -0{,}1 & 0{,}1 & -0{,}98 & -0{,}59 & -0{,}59 & \ll 8 \gg & -0{,}1 \\
0 & 0 & -1{,}79 & -0{,}88 & -0{,}51 & 0{,}29 & 0{,}45 & -0{,}45 & 0{,}05 & -1{,}2 & -1{,}2 & \ll 8 \gg & -1{,}23 \\
0 & 0 & 0 & -1{,}39 & -0{,}19 & 0{,}23 & 0{,}02 & -0{,}02 & -0{,}55 & -0{,}27 & 0{,}45 & \ll 8 \gg & 0{,}36 \\
0 & 0 & 0 & 0 & -1{,}32 & 0{,}03 & -0{,}1 & 0{,}1 & 0{,}02 & -0{,}58 & -0{,}69 & \ll 8 \gg & -0{,}27 \\
0 & 0 & 0 & 0 & 0 & 1{,}03 & 0{,}33 & -0{,}33 & -0{,}13 & -0{,}69 & 0{,}13 & \ll 8 \gg & -0{,}23 \\
0 & 0 & 0 & 0 & 0 & 0 & 0{,}93 & 0{,}15 & -0{,}07 & -0{,}15 & -0{,}47 & \ll 8 \gg & -0{,}23 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0{,}92 & 0{,}08 & 0{,}17 & 0{,}55 & \ll 8 \gg & 0{,}27 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0{,}85 & -0{,}29 & 0{,}36 & \ll 8 \gg & -0{,}13 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0{,}63 & 0{,}27 & \ll 8 \gg & 0{,}54
\end{array}\right)$$

Figura 3.1: Matriz R de la factorización QR de la matriz del ejemplo 2.1.1 con entradas redondeadas a dos decimales. Ocho columnas han sido omitidas por falta de espacio. La parte de la matriz que queda a la izquierda de la línea vertical es el bloque que hemos denotado $R_{\triangle}$, y la que queda a la derecha, $R_{\square}$.

Rango	Variación	
	Máxima	Percentil 90
1	1.000000	1.000000
2	0.732467	0.707107
3	0.369598	0.338235
4	0.448908	0.337108
5	0.249269	0.178434
6	0.238350	0.145024
7	0.202940	0.143500
8	0.181113	0.096965
9	0.0523224	0.0523224
10	0.000000	0.000000

Tabla 3.1: Percentiles cien y noventa de las variaciones de similitud mediante (3.1) conforme el rango de la matriz base de datos del ejemplo 2.1.1 disminuye. Menor variación es mejor.

Podemos comparar los efectos de la «altura del corte» (véanse las tablas 3.1 y 3.2). Observamos que las diferencias de similitud de cada documento se mantienen por debajo del 10 % en el 90 por ciento de los casos hasta que se reduce el rango en tres*, por lo que podríamos decir que la aproximación de rango ocho correspondiente es la forma más «simplificada» de representar los datos de la matriz A.

*En general, el umbral de modificación que se considera aceptable varía según el corpus considerado; una variación del 10 % suele ser inaceptable, pero a modo de ilustración del método basta.

Rango	Variación	
	Máxima	Percentil 90
1	0.623000	0.353553
2	0.405976	0.361842
3	0.375758	0.375758
4	0.315563	0.219183
5	0.236602	0.236602
6	0.284243	0.216178
7	0.304149	0.249738
8	0.299138	0.254058
9	0.147870	0.131476
10	0.132545	0.0200847

Tabla 3.2: Percentiles cien y noventa de las variaciones de similitud mediante (3.1) conforme el rango de la matriz base de datos del ejemplo 2.1.2 disminuye. Menor variación es mejor.

3.2. El LSA y la Descomposición en Valores Singulares

3.2.1. Existencia de la Descomposición en Valores Singulares

Teorema 3.2.1 *Toda matriz A con F filas y C columnas se puede expresar como un producto de tres matrices, $A = U\Sigma V^*$, donde U y V son matrices unitarias $F \times F$ y $C \times C$ respectivamente y Σ es una matriz real $F \times C$ diagonal cuyos elementos diagonales (los «valores singulares» de A) son no negativos y están ordenados de mayor a menor. Este producto se conoce como descomposición en valores singulares de A, usualmente abreviado «SVD» por sus siglas en inglés.*

Demostración: Para ver la existencia de dicha descomposición seguiremos los pasos descritos en [30, 27], para lo cual empezaremos considerando la matriz A^*A. Por ser simétrica y semi-definida positiva es también diagonalizable ortogonalmente y sus autovalores son reales y no negativos.

Supondremos que sus autovalores no nulos están ordenados de mayor a menor: $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_r > 0$ donde $r = \operatorname{rg}(A^*A) = \operatorname{rg}(A)$ y λ_i es el i-ésimo mayor autovalor de A^*A para $i = 1, 2, \ldots, \operatorname{mín}(F, C)$. Para $i = 1, 2, \ldots, r$ definimos $\mathbf{v}_i$ como un autovector unitario asociado a λ_i ortogonal a los demás y ponemos $\sigma_i = +\sqrt{\lambda_i}$.

Se tiene que $\mathbf{u}_i = A\mathbf{v}_i/\sigma_i$ es un autovector unitario de AA^*:

$$AA^*\mathbf{u}_i = AA^*A\mathbf{v}_i/\sigma_i = A{\sigma_i}^2\mathbf{v}_i/\sigma_i = {\sigma_i}^2\mathbf{u}_i$$
$$\mathbf{u}_i \cdot \mathbf{u}_i = (A\mathbf{v}_i/\sigma_i) \cdot (A\mathbf{v}_i/\sigma_i) = {\mathbf{v}_i}^*A^*A\mathbf{v}_i/{\sigma_i}^2 = {\mathbf{v}_i}^*{\sigma_i}^2\mathbf{v}_i/{\sigma_i}^2 = 1$$

Sea V_r la matriz cuyas columnas son los $\mathbf{v}_i$ en orden, Σ_r la matriz diagonal cuyos elementos diagonales son los σ_i en orden y U_r, cuyas columnas son los $\mathbf{u}_i$ en orden.

Tenemos la igualdad $U_r = AV_r{\Sigma_r}^{-1}$, de donde $U_r\Sigma_r{V_r}^* = A$, expresión que en ocasiones se denomina SVD «compacta» de A [1].

Podemos añadir columnas a U_r y V_r hasta completar bases ortonormales de los espacios $\mathbb{C}^F$ y $\mathbb{C}^C$ respectivamente, obteniendo así las matrices U y V. Si también añadimos filas y columnas de ceros a Σ_r hasta que haya F filas y C columnas, obteniendo en este caso la matriz Σ, tenemos finalmente la igualdad $A = U\Sigma V^*$ como buscamos. □

Implementaremos únicamente la SVD compacta siguiendo el esquema anterior. Aprovecharemos también para incluir la opción de indicar el número de valores singulares buscados dado que la usaremos más adelante a la hora de aplicar reducción de rango.

```
compactSvd[mA_?MatrixQ, OptionsPattern[{SingularValues → Automatic}]] := Module[
    {vSigmas, vvRightSingularVectors},
    {vSigmas, vvRightSingularVectors}
        = Eigensystem[
                ConjugateTranspose[mA] . mA,
                OptionValue[SingularValues] /. Automatic → UpTo[Infinity]
            ] // Transpose // Select[Positive @* First] // Transpose
            // Comap[{First /* Sqrt, Last /* Transpose}];
    {
        mA . vvRightSingularVectors / Threaded[vSigmas],
        DiagonalMatrix[vSigmas],
        vvRightSingularVectors
    }
]
```

Es importante notar que el algoritmo que hemos descrito en esta sección tiene dos problemas graves:

- Requiere calcular A^*A, que puede ser una matriz de gran tamaño y almacenarla puede requerir más memoria de la disponible.
- Requiere hallar los autovalores de dicha matriz, que en coma flotante puede ser difícil (si no imposible) de hacer sin perder precisión y de manera poco costosa.

Se han desarrollado algoritmos [5] que solventan estos problemas, aunque su implementación resulta mucho más compleja que la mostrada aquí.

En cuanto a resultados que podemos extraer de la SVD tenemos el siguiente teorema [19, 7]:

Teorema 3.2.2 *Sea $A \in \mathbb{C}^{F\times C}$ y sea $A = U\Sigma V^*$ su SVD. Entonces:*

- *Las primeras* $\operatorname{rg}(A)$ *columnas de U forman una base del espacio columna de A.*
- *Las primeras* $\operatorname{rg}(A)$ *columnas de V forman una base del espacio fila de A.*
- *Las últimas* $F - \operatorname{rg}(A)$ *columnas de U forman una base del núcleo de A^*.*
- *Las últimas* $C - \operatorname{rg}(A)$ *columnas de V forman una base del núcleo de A.*
- *Se cumple que* $\|A\|_F = \sqrt{\sum_{i=1}^{\operatorname{rg}(A)} {\sigma_i}^2}$.

Demostración: La última propiedad se probó al inicio de la sección 3.1.5, y la segunda y tercera son la primera y cuarta aplicadas a A^*.

Para demostrar cuarta propiedad, observamos que se tiene $A\mathbf{x} = 0$ si y solo si $(U^*AV)(V^*\mathbf{x}) = 0$ por ser U y V unitarias (en particular, U^* es invertible). Como el primer factor de la última igualdad es igual a Σ, se tiene que $A\mathbf{x} = 0$ si y solo si $V^*\mathbf{x}$ está en el núcleo de Σ, lo cual implica que el núcleo de A tiene por base las últimas $C - \operatorname{rg}(A)$ columnas de V por la forma que tiene Σ.

La primera propiedad se demuestra de manera similar considerando la premultiplicación de $\Sigma = U^*AV$ por U. □

3.2.2. Teorema de Eckart-Young-Mirsky

La SVD nos proporciona una forma de hallar la matriz de un rango dado más cercana a otra matriz cualquiera:

Teorema 3.2.3 (de Eckart-Young-Mirsky [8, 13]) *Consideremos la SVD de la matriz $A \in \mathbb{C}^{F\times C}$ es $A = U\Sigma V^\intercal = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^\intercal$ donde $r = \operatorname{rg}(A)$, y para $k \leq r$ definimos $A_{(k)} = \sum_{i=1}^{k} \sigma_i \mathbf{u}_i \mathbf{v}_i^\intercal$.*

Entonces se tiene

$$\min_{X \text{ tal que } \operatorname{rg}(X)\leq k} \|A - X\|_F = \|A - A_{(k)}\|_F.$$

Demostración: seguiremos el esquema planteado en [14], para lo cual es necesaria la desigualdad de Weyl [21]:

$$\sigma_{i+j-1}(X + Y) \leq \sigma_i(X) + \sigma_j(Y)$$

con X, Y matrices cualquiera y $\sigma_i(X)$ el i-ésimo mayor valor singular de X.

Sea $B \in \mathbb{C}^{F\times C}$ de rango k. Aplicando la desigualdad de Weyl a $A - B$ y B, tenemos que $\sigma_{i+k}(A) \leq \sigma_i(A - B) + \sigma_{k+1}(B)$, siendo el último sumando nulo por ser B de rango k.

Obtenemos la siguiente cadena de desigualdades:

$$\|A - A_{(k)}\|_F = \sum_{i=k+1}^{r} \sigma_i(A)^2 = \sum_{i=1}^{r-k} \sigma_{i+k}(A)^2 \leq \sum_{i=1}^{r-k} \sigma_i(A - B)^2 \leq \sum_{i=1}^{r} \sigma_i(A - B)^2 = \|A - B\|_F$$

□

3.2.3. Aplicación de la SVD a la búsqueda de documentos

Usar factorización QR es una forma relativamente poco costosa tanto de determinar el rango de A como de reducirlo para reducir la incertidumbre que poseen sus datos. Sin embargo, esta factorización es insuficiente para analizar las relaciones entre términos (pues solo considera el espacio columna y no el espacio fila) y además para un rango dado el proceso de reducción de rango puede inducir una perturbación más grande de lo estrictamente necesario.

Consideraremos ahora la descomposición en valores singulares de la matriz base de datos A, $A = U\Sigma V^\top$. En base a las propiedades descritas anteriormente, para reducir el rango de A nos basta con anular los valores singulares no nulos más pequeños: Una vez fijado el rango deseado k, consideramos la división por bloques indicada en la figura 3.2 de las matrices resultantes de descomponer A y definimos la aproximación buscada $A_{(k)}$.

$$
\begin{aligned}
A &= U\Sigma V^\top \\
&= \left(\begin{array}{cccc|ccc|ccc} \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_k & \mathbf{u}_{k+1} & \cdots & \mathbf{u}_{\mathrm{rg}(A)} & \mathbf{u}_{\mathrm{rg}(A)+1} & \cdots & \mathbf{u}_T \end{array}\right) \ddots \\
&\quad \times \left(\begin{array}{cccc|ccc|ccc} \sigma_1 & & & & & & & & & \\ & \sigma_2 & & & & & & & & \\ & & \ddots & & & & & & & \\ & & & \sigma_k & & & & & & \\ \hline & & & & \sigma_{k+1} & & & & & \\ & & & & & \ddots & & & & \\ & & & & & & \sigma_{\mathrm{rg}(A)} & & & \\ \hline & & & & & & & 0 & & \\ & & & & & & & & \ddots & \\ & & & & & & & & & 0 \end{array}\right) \times \left(\begin{array}{c} \mathbf{v}_1^\top \\ \mathbf{v}_2^\top \\ \vdots \\ \mathbf{v}_k^\top \\ \hline \mathbf{v}_{k+1}^\top \\ \vdots \\ \mathbf{v}_{\mathrm{rg}(A)}^\top \\ \hline \mathbf{v}_{\mathrm{rg}(A)+1}^\top \\ \vdots \\ \mathbf{v}_D^\top \end{array}\right) \\
A_{(k)} &= U_{(k)}\Sigma_{(k)}V_{(k)}^\top \\
&= \left(\begin{array}{cccc|ccc|ccc} \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_k & \mathbf{u}_{k+1} & \cdots & \mathbf{u}_{\mathrm{rg}(A)} & \mathbf{u}_{\mathrm{rg}(A)+1} & \cdots & \mathbf{u}_T \end{array}\right) \ddots \\
&\quad \times \left(\begin{array}{cccc|ccc|ccc} \sigma_1 & & & & & & & & & \\ & \sigma_2 & & & & & & & & \\ & & \ddots & & & & & & & \\ & & & \sigma_k & & & & & & \\ \hline 0 & & & & 0 & & & & & \\ & \ddots & & & & 0 & & & & \\ & & 0 & & & & 0 & & & \\ \hline & & & 0 & & & & 0 & & \\ & & & & \ddots & & & & \ddots & \\ & & & & & 0 & & & & 0 \end{array}\right) \times \left(\begin{array}{c} \mathbf{v}_1^\top \\ \mathbf{v}_2^\top \\ \vdots \\ \mathbf{v}_k^\top \\ \hline \mathbf{v}_{k+1}^\top \\ \vdots \\ \mathbf{v}_{\mathrm{rg}(A)}^\top \\ \hline \mathbf{v}_{\mathrm{rg}(A)+1}^\top \\ \vdots \\ \mathbf{v}_D^\top \end{array}\right)
\end{aligned}
$$

Figura 3.2: Descomposición por bloques de la SVD y SVD con rango reducido. Los elementos no indicados de Σ y $\Sigma_{(k)}$ son ceros. Ambas matrices pueden tener filas abajo o columnas a la derecha cuyas componentes sean cero de modo que los productos tengan sentido.

En ambos casos los vectores entre $\mathbf{u}_{\mathrm{rg}(A)+1}$ y $\mathbf{u}_T$ y entre $\mathbf{v}_{\mathrm{rg}(A)+1}$ y $\mathbf{v}_D$ (los cuatro inclusive) son irrelevantes para determinar A y $A_{(k)}$. La reducción de rango hace que los vectores entre $\mathbf{u}_{k+1}$ y $\mathbf{u}_{\mathrm{rg}(A)}$ y entre $\mathbf{v}_{k+1}$ y $\mathbf{v}_{\mathrm{rg}(A)}$ también lo sean.

Con la notación anterior, se tiene que $\|A - A_{(k)}\|_F = \sqrt{\sum_{i=k+1}^{\mathrm{rg}(A)} \sigma_i}$ y, en base al teorema de Eckart-Young-Mirsky, que $A_{(k)}$ es la matriz de rango k más cercana a A según la distancia asociada a la norma de Frobenius.

Si repetimos el análisis que hicimos para obtener la medida (3.1), esta vez a la matriz $A_{(k)}$, obtenemos que

$$\begin{aligned} s_d^{\cos}(\mathbf{q}) &= \frac{\left(A_{(k)} e_d\right)^\intercal \mathbf{q}}{\left\|A_{(k)} e_d\right\|_2 \|\mathbf{q}\|_2} \\ &= \frac{\left(U_{(k)} \Sigma_{(k)} {V_{(k)}}^\intercal e_d\right)^\intercal \mathbf{q}}{\left\|\Sigma_{(k)} {V_{(k)}}^\intercal e_d\right\|_2 \|\mathbf{q}\|_2} \\ &= \frac{\left(\Sigma_{(k)} {V_{(k)}}^\intercal e_d\right)^\intercal \left({U_{(k)}}^\intercal \mathbf{q}\right)}{\left\|\Sigma_{(k)} {V_{(k)}}^\intercal e_d\right\|_2 \|\mathbf{q}\|_2} \\ &= \frac{{v_{(k),d}}^\intercal \left({U_{(k)}}^\intercal \mathbf{q}\right)}{\left\|v_{(k),d}\right\|_2 \|\mathbf{q}\|_2} \end{aligned}$$

donde e_d denota el d-ésimo vector canónico de $\mathbb{R}^D$. Observamos que podemos calcular la medida (2.3) sin calcular $A_{(k)}$ y que, de nuevo, podemos definir una medida en base a la proyección de $\mathbf{q}$ al espacio columna:

$$s_d^{\cos,\ \mathrm{SVD},\ (k)}(\mathbf{q}) := \frac{{v_{(k),d}}^\intercal \left({U_{(k)}}^\intercal \mathbf{q}\right)}{\left\|v_{(k),d}\right\|_2 \left\|U_{(k)} \mathbf{q}\right\|_2} \tag{3.3}$$

Al igual que la medida (3.1) para la factorización QR, (3.3) aumenta la exhaustividad y puede reducir la precisión, considerando generalmente relevantes más documentos que (2.3). Desde el punto de vista computacional esta medida requiere menos memoria, pues los vectores $v_{(k),d}$ son de tamaño k y dado que $\mathbf{q}$ suele tener pocos elementos no nulos, calcular $U_{(k)}\mathbf{q}$ es relativamente poco costoso. Asimismo, observamos que para calcular (3.3) no hace falta factorizar completamente la matriz A, pues fijado k nos basta con determinar los k valores singulares más grandes (para lo cual usualmente se usa un proceso iterativo precedido de una bidiagonalización [5, 20]) y las matrices $U_{(k)}$ y $V_{(k)}$ asociadas.

3.2.4. Implementación

De nuevo la medida (3.3) es un caso particular de (3.2), así que podemos implementarla usando el mismo esquema que en el caso de la factorización QR. Para la implementación de la reducción de rango en este caso basta con indicar cuántos valores singulares queremos en el momento de descomponer la matriz, como ya habíamos previsto.

```
prepareSvdState[aDatabase_Association, OptionsPattern[{MatrixRank → Automatic}]] := \
    compactSvd[
        aDatabase // Values // Map[Values] // N,
        SingularValues → OptionValue[MatrixRank]
    ] // {{#[[3]], Transpose[#[[2]]]}, Transpose[#[[1]]]} &
```

Usando la base de datos de ejemplo con la que estamos trabajando (que tiene rango 10), reducir el rango en uno produce cambios cercanos al uno por mil; la mayor variación es de aproximadamente 7×10^{-3}.

```
generalizedCosineSimilarity[Sequence@@prepareSvdState[database, MatrixRank → 9],query]
    - generalizedCosineSimilarity[Sequence @@ prepareSvdState[database], query]
{0.002028, 0.000103, -0.000354, <<15>>, 0.000957, 0.002028}
```

Podemos comparar los efectos de reducir el rango, del mismo modo que para la factorización QR. Basándonos en los resultados recogidos en la tabla 3.3 (cf. la tabla 3.1), para la base de datos del ejemplo 2.1.1, la aproximación de rango siete parece ser la más «simplificada», aunque quizá la de rango seis pudiera resultar aceptable también. En el caso del ejemplo 2.1.2 también podemos reducir el rango en dos sin causar variaciones grandes, como se observa en la tabla 3.4 (cf. la tabla 3.2).

$\operatorname{rg}(A)$	Max	90 % Quantile
1	1.0	1.0
2	0.985579	0.878562
3	0.453133	0.285552
4	0.447203	0.244098
5	0.373474	0.237177
6	0.267024	0.119806
7	0.122873	0.0585781
8	0.0263336	0.00970509
9	0.00669728	0.0045406
10	0.0	0.0

Tabla 3.3: Percentiles cien y noventa de las variaciones de similitud mediante (3.3) conforme el rango de la matriz base de datos del ejemplo 2.1.1 disminuye. Menor variación es mejor.

$\operatorname{rg}(A)$	Max	90 % Quantile
1	1.0	1.0
2	0.999456	0.639940
3	0.619118	0.520687
4	0.692487	0.287495
5	0.391385	0.297690
6	0.340313	0.185684
7	0.332209	0.133763
8	0.226543	0.124723
9	0.0697887	0.0567215
10	0.0663256	0. 0612231
11	0.0	0.0

Tabla 3.4: Percentiles cien y noventa de las variaciones de similitud mediante (3.3) conforme el rango de la matriz base de datos del ejemplo 2.1.2 disminuye. Menor variación es mejor.

3.3. Modificación de la base de datos

En esta sección describiremos técnicas la modificación de una base de datos existente A cuando se dispone de la aproximación de rango reducido $A_{(k)}$ usada en la medida (3.3). En particular, explicaremos cómo añadir términos y documentos y cómo editar las ponderaciones entre ellos.

Desde el punto de vista de las bases de datos en informática, estas operaciones se corresponden con la creación y actualización de entradas en el modelo CRUD de almacenamiento persistente [23]; la tercera operación de dicho modelo, lectura, en nuestro caso se reduce a mostrar las ponderaciones asociadas a los términos y documentos, y la cuarta, borrado, es un proceso relativamente complejo detallado en [20].

3.3.1. Adición de Términos

Supongamos que queremos introducir S nuevos términos a una base de datos ya existente para la que ya hemos determinado la aproximación de rango k mediante SVD, $A_{(k)} = U_{(k)}\Sigma_{(k)}{V_{(k)}}^{\mathsf{T}}$. Al igual que los términos originales, expresaremos los nuevos términos en forma matricial obteniendo la matriz N, con D columnas y S filas.

Definimos la matriz por bloques $B = \begin{pmatrix} A_{(k)} & N \end{pmatrix}^{\mathsf{T}}$; es evidente que B tiene rango al menos k. Se tiene la siguiente cadena de igualdades:

$$\begin{aligned} B = \begin{pmatrix} A_{(k)} \\ N \end{pmatrix} &= \begin{pmatrix} U_{(k)}\Sigma_{(k)}{V_{(k)}}^{\mathsf{T}} \\ N \end{pmatrix} \\ &= \begin{pmatrix} U_{(k)} & 0 \\ 0 & I \end{pmatrix} \left(\begin{pmatrix} \Sigma_{(k)} \\ NV_{(k)} \end{pmatrix} {V_{(k)}}^{\mathsf{T}} + \begin{pmatrix} 0 \\ N\left(I - V_{(k)}{V_{(k)}}^{\mathsf{T}}\right) \end{pmatrix} \right) \\ &= \begin{pmatrix} U_{(k)} & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} \Sigma_{(k)} & 0 \\ NV_{(k)} & 1 \end{pmatrix} \begin{pmatrix} {V_{(k)}}^{\mathsf{T}} \\ N\left(I - V_{(k)}{V_{(k)}}^{\mathsf{T}}\right) \end{pmatrix} \\ &= \begin{pmatrix} U_{(k)} & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} \Sigma_{(k)} & 0 \\ NV_{(k)} & I \end{pmatrix} \begin{pmatrix} V_{(k)} & \left(I - V_{(k)}{V_{(k)}}^{\mathsf{T}}\right)N^{\mathsf{T}} \end{pmatrix}^{\mathsf{T}} \end{aligned}$$

donde cada I representa la matriz identidad del tamaño adecuado según dónde aparece.

El primer factor es una matriz unitaria y el segundo, triangular inferior. Si hacemos que el tercer factor sea unitario, solo hará falta aplicar SVD al segundo factor para calcular la SVD de B.

Calculando la factorización QR compacta de $\left(I - V_{(k)}{V_{(k)}}^{\mathsf{T}}\right)N^{\mathsf{T}}$ obtenemos el producto QRP^{T} y podemos continuar la cadena de igualdades anterior:

$$\begin{aligned} B &= \begin{pmatrix} U_{(k)} & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} \Sigma_{(k)} & 0 \\ NV_{(k)} & I \end{pmatrix} \begin{pmatrix} V_{(k)} & QRP^{\mathsf{T}} \end{pmatrix}^{\mathsf{T}} \\ &= \begin{pmatrix} U_{(k)} & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} \Sigma_{(k)} & 0 \\ NV_{(k)} & PR^{\mathsf{T}} \end{pmatrix} \begin{pmatrix} V_{(k)} & Q \end{pmatrix}^{\mathsf{T}} \end{aligned}$$

y ahora el tercer factor tiene filas ortonormales.

Aplicando SVD al segundo factor podemos escindir por bloques las matrices de la descomposición de la siguiente manera:

$$\begin{pmatrix} \Sigma_{(k)} & 0 \\ NV_{(k)} & PR^\intercal \end{pmatrix} = \begin{pmatrix} P_{(k)} & {P_{(k)}}^\perp \end{pmatrix} \begin{pmatrix} \widehat{\Sigma}_{(k)} & 0 \\ 0 & \widehat{\Sigma}_{(S)} \end{pmatrix} \begin{pmatrix} Q_{(k)} \\ {Q_{(k)}}^\perp \end{pmatrix}$$

de donde la mejor aproximación de rango k a B viene dada por

$$B_{(k)} = \underbrace{\begin{pmatrix} U_{(k)} & 0 \\ 0 & I \end{pmatrix} P_{(k)}}_{U_B} \underbrace{\widehat{\Sigma}_{(k)}}_{\Sigma_B} \underbrace{\begin{pmatrix} {V_{(k)}}^\intercal & Q \end{pmatrix} Q_{(k)} \Big)^\intercal}_{{V_B}^\intercal},$$

cuya descomposición consideraríamos actual para la base de datos modificada.

3.3.2. Adición de Documentos

El proceso de adición de documentos es similar al de adición de términos.

Supongamos que queremos introducir E nuevos documentos a una base de datos ya existente para la que, de nuevo, ya hemos determinado la aproximación de rango k mediante SVD, $A_{(k)} = U_{(k)}\Sigma_{(k)}{V_{(k)}}^\intercal$. Expresaremos los nuevos documentos en forma matricial obteniendo la matriz N, con E columnas y T filas, y ahora ponemos $B = \begin{pmatrix} A_{(k)} & N \end{pmatrix}$.

Repitiendo el razonamiento hecho para la adición de términos considerando la transposición de las matrices involucradas, llegamos a que Consideramos la factorización QR compacta de la matriz $\left(I - U_{(k)}{U_{(k)}}^\intercal\right)N$ obteniendo así la igualdad

$$\left(I_T - U_{(k)}{U_{(k)}}^\intercal\right)NP = QR$$

de donde [2]

$$B = \begin{pmatrix} U_{(k)} & Q \end{pmatrix} \begin{pmatrix} \Sigma_{(k)} & {U_{(k)}}^\intercal N \\ 0 & RP^\intercal \end{pmatrix} \begin{pmatrix} {V_{(k)}}^\intercal & 0 \\ 0 & I \end{pmatrix}$$

Aplicando SVD a la matriz central tenemos que

$$\begin{pmatrix} \Sigma_{(k)} & {U_{(k)}}^\intercal N \\ 0 & RP^\intercal \end{pmatrix} = \begin{pmatrix} P_{(k)} & {P_{(k)}}^\perp \end{pmatrix} \begin{pmatrix} \widehat{\Sigma}_{(k)} & 0 \\ 0 & \widehat{\Sigma}_{(S)} \end{pmatrix} \begin{pmatrix} Q_{(k)} \\ {Q_{(k)}}^\perp \end{pmatrix}$$

y entonces la mejor aproximación de rango k a B es

$$B_{(k)} = \underbrace{\begin{pmatrix} U_{(k)} & Q \end{pmatrix} P_{(k)}}_{U_B} \underbrace{\widehat{\Sigma}_{(k)}}_{\Sigma_B} \underbrace{\left(\begin{pmatrix} {V_{(k)}}^\intercal & 0 \\ 0 & I_E \end{pmatrix} Q_{(k)} \right)^\intercal}_{{V_B}^\intercal}$$

obteniendo así las matrices actualizadas U_B, Σ_B y V_B.

3.3.3. Modificación de documentos

En este caso la operación deseada es modificar la ponderación de E términos. Expresaremos esta modificación en dos partes: una matriz Y de tamaño $T \times E$ con entradas ceros y unos que especifica qué términos modificar, y una matriz Z de tamaño $D \times E$ que especifica esas modificaciones. Esto se resume en que la matriz «nueva» es $B = A_{(k)} + YZ^\intercal$.

Al igual que para la incorporación de nuevos documentos, el procedimiento de edición también usa la factorización QR compacta, obteniendo las descomposiciones siguientes:

$$\begin{aligned}\left(I_T - U_{(k)}{U_{(k)}}^\intercal\right)YP_Y &= Q_Y R_Y \\ \left(I_D - V_{(k)}{V_{(k)}}^\intercal\right)ZP_Z &= Q_Z R_Z\end{aligned}$$

De aquí,

$$B = \begin{pmatrix}U_{(k)} & Q_Y\end{pmatrix}\underbrace{\left(\begin{pmatrix}\Sigma_{(k)} & 0\\ 0 & 0\end{pmatrix} + \begin{pmatrix}{U_{(k)}}^\intercal Y\\ R_Y {P_Y}^\intercal\end{pmatrix}\begin{pmatrix}Z^\intercal V_{(k)} & P_Z {R_Z}^\intercal\end{pmatrix}\right)}_{\widehat{B}}\begin{pmatrix}{V_{(k)}}^\intercal\\ {Q_Z}^\intercal\end{pmatrix}.$$

Al igual que para los procesos de adición, hemos de aplicar SVD al segundo factor, pero esta vez no hay una forma «simple» de abaratar dicha descomposición.

$$\widehat{B} = \begin{pmatrix}P_{(k)} & {P_{(k)}}^\perp\end{pmatrix}\begin{pmatrix}\widehat{\Sigma}_{(k)} & 0\\ 0 & \widehat{\Sigma}_{(E)}\end{pmatrix}\begin{pmatrix}Q_{(k)}\\ Q_{(k)}^\perp\end{pmatrix}$$

de donde la mejor aproximación de rango k a B es

$$B_{(k)} = \underbrace{\begin{pmatrix}U_{(k)} & Q_Y\end{pmatrix}P_{(k)}}_{U_B}\underbrace{\widehat{\Sigma}_{(k)}}_{\Sigma_B}\underbrace{\left(\begin{pmatrix}V_{(k)} & Q_Z\end{pmatrix}Q_{(k)}\right)^\intercal}_{{V_B}^\intercal}$$

de nuevo obteniendo así las matrices actualizadas U_B, Σ_B y V_B.

3.3.4. Implementación

La implementación de las técnicas de modificación anteriores se resume en la aplicación de las fórmulas descritas. De cara a la integración con la función de búsqueda, implementaremos los métodos de manera que preprocesen los datos ya calculados, esto es, que tomen como datos de entrada la base de datos descompuesta mediante SVD y devuelvan los datos en formato apropiado para ser argumento de la función de búsqueda.

En otras palabras, las modificaciones podrán ser aplicadas de la siguiente manera:

```
generalizedCosineSimilarity[
    Sequence @@ (
        prepareSvdState[database // Transpose, MatrixRank → rango]
        // modificacion1
        // modificacion2
        // ...
    ),
    query
]
```

donde solo hay que especificar `rango` y cada `modificación`. El paréntesis, una vez evaluado, representa el estado de la base de datos; no hace falta recalcularlo para cada vector petición, sino que conviene mantenerlo en memoria.

La implementación de las tres operaciones descritas en esta sección (adición de términos, adición de documentos y edición de pesos) se halla al final del listado de códigos del apéndice D. Dichas implementaciones usan `OperatorApplied`, de modo que esperan que se les pasen los datos de la modificación primero y la base de datos como su último argumento. Esto evita tener que definir nuevas funciones para usar el fragmento de código anterior.

Para ilustrar lo anterior consideramos añadir el término «Pixel Art» a la base de datos del ejemplo 2.1.1 (con las ponderaciones adecuadas), con rango 8, e incorporarlo al vector petición, podemos usar

```
generalizedCosineSimilarity[
    Sequence @@ (
        prepareSvdState[database // Transpose, MatrixRank → 8]
        // addTerms[{{1,0, 0,0, 0,0, 1,0, 1,0, 1,1, 0,1, 1,1, 1,0, 0,1}}, {10, 20}]
    ),
    query ~Append~ ("Pixel Art" → 1)
]
{0.595482, 0.296402, 0.0192178, <<15>>, 0.00213558, 0.595482}
```

para actualizar la base de datos y realizar la búsqueda.

A lo anterior se puede encadenar más operaciones intermedias:

```
generalizedCosineSimilarity[
    Sequence @@ (
        prepareSvdState[database // Transpose, MatrixRank → 8]
        // addTerms[{{1,0, 0,0, 0,0, 1,0, 1,0, 1,1, 0,1, 1,1, 1,0, 0,1}}, {10, 20}]
        // addDocuments[{{1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1}} // Transpose, {11, 20}]
    ),
    query ~Append~ ("Pixel Art" → 1)
]
{0.605605, 0.302629, 0.0061957, <<16>>, 0.605605, 0.288521}
```

(el vector de retorno de este fragmento tiene un elemento más que el del fragmento anterior)

Conviene notar que en operaciones sucesivas se debe tener en cuenta que T y D pueden haber cambiado a causa de modificaciones previas, como es el caso en este último ejemplo.

Apéndice A

Run-Length Encoding

Run-Length Encoding [31, 32] («Codificación por longitud de racha» en inglés, a menudo abreviado RLE) es una forma muy simple de compresión de datos cuyo propósito principal es minimizar el espacio usado para almacenar rachas de un mismo valor. Para ello, se divide la cadena de valores a comprimir en las rachas de valores idénticos que la conforman y, después, por cada racha, se emiten dos valores: la longitud de la racha y el valor original en ella (en ocasiones aparece primero el valor y luego la longitud; ambas representaciones son funcionalmente equivalentes).

Supongamos queremos comprimir la siguiente ristra de 40 números:

$$\underbrace{11111}_{51}\underbrace{000000}_{60}\underbrace{1111}_{41}\underbrace{33333}_{53}\underbrace{22222}_{52}\underbrace{000000}_{60}\underbrace{444}_{34}\underbrace{000}_{30}\underbrace{1}_{11}\underbrace{99}_{29}$$

En la parte inferior hemos indicado cómo se emitiría cada racha, resultando en una salida de 20 caracteres, un ratio de compresión del 50 %.

Esta forma de compresión produce un mejor ratio de compresión conforme las rachas en la fuente se vuelven más largas, pero en el peor de los casos (cuando no hay rachas como tal porque cada valor consecutivo es distinto) *Run-Length Encoding* puede duplicar el tamaño de los datos originales:

$$\underbrace{1}_{11}\underbrace{0}_{10}\underbrace{1}_{11}\underbrace{3}_{13}\underbrace{2}_{12}\underbrace{0}_{10}\underbrace{4}_{14}\underbrace{0}_{10}\underbrace{1}_{11}\underbrace{9}_{19}$$

En este caso la salida resulta ser de 20 caracteres para una entrada de solo diez, siendo el ratio de compresión del 200 %.

Para mitigar este problema, en ocasiones la fuente es preprocesada para tratar de crear esas rachas, por ejemplo, usando codificación por deltas [32].

Apéndice B

Neuronas y la función ReLU

En el contexto del aprendizaje automático (*machine learning* en inglés) una neurona es cada una de las «células» de cada capa de una red neuronal [12, 22, 32]. Cada neurona tiene una serie de entradas y emite una sola salida que depende de una ponderación interna (léase combinación lineal) de las entradas a la neurona a la que se aplica una función «de activación» común a todas las neuronas (esto es, cada neurona pondera sus entradas de una manera generalmente distinta y dos neuronas con la misma ponderación resultante emiten la misma salida). Tanto las entradas como salidas se modelan como números reales.

Las neuronas se dividen en capas; la primera capa se denomina «de entrada» y la última, «de salida»; las capas intermedias se denominan «ocultas». Cada neurona recibe como entradas las salidas de las neuronas de la capa anterior y envía su salida como entrada de las de la capa siguiente, con la excepción de que la entrada de las neuronas de la capa de entrada proviene «del exterior» y la salida de la capa de salida va también «hacia el exterior» de la red.

La determinación de la ponderación de cada entrada a una neurona se va ajustando durante el proceso de entrenamiento de la red, que inherentemente depende del propósito de la red. La ponderación de una entrada dada puede ser positiva o negativa; normalmente una ponderación negativa tiende a hacer que la neurona no emita salida cuando recibe la entrada correspondiente. Aparte de la ponderación de sus entradas, cada neurona tiene un sesgo propio ajustable usado para moderar la salida, como si tuviera una entrada conectada a otra neurona que siempre emite como salida un uno.

Existen múltiples posibles funciones de activación para determinar qué salida emite una neurona [15], Como cabe esperar, la más simple de estas funciones es la función identidad, que simplemente emite el resultado de ponderar las entradas y sumar el sesgo. Sin embargo, la función de activación usada más comúnmente resulta ser la función «ReLU», acrónimo de *Rectified Linear Unit* («Unidad Lineal Rectificada» en inglés):

$$\mathrm{ReLU}(x) = \mathrm{máx}(0, x) = \begin{cases} x & x \geq 0 \\ 0 & x \leq 0 \end{cases}$$

Apéndice C

El sistema SMART

El control sobre los términos considerados ayuda a la hora de determinar qué documentos resultan ser más relevantes para una búsqueda dada. La implementación «tradicional» de este método de control se conoce como sistema SMART, siglas de *System for the Mechanical Analysis and Retrieval of Text*, «sistema para el análisis y la obtención de texto mecanizados» [17, 18].

Originalmente diseñado para la búsqueda de textos en inglés, el sistema SMART preprocesa los documentos de la base de datos, determinando automáticamente los términos contenidos en ella: Primero se eliminan las palabras comunes y que no aportan al contenido semántico de los documentos y peticiones, tales como «y», «de», «pero» o «es», y después se reducen las palabras restantes a sus lexemas: «información» → «inform». A cada lexema se le asigna una ponderación dentro de cada vector documento y petición como si fuera un término más.

Además de lexemas, el sistema SMART considera algunas expresiones y grupos sintácticos como términos. La determinación de qué expresiones y grupos se basa en emparejar los lexemas obtenidos mediante el preprocesado anterior que aparezcan comparativamente juntos (separados por pocas palabras) con mayor frecuencia. El peso correspondiente a estas expresiones viene dado en función de los pesos de los términos que las forman; en el caso común en que los pesos están en el intervalo $[0, 1]$, el peso de las expresiones suele tomarse como el producto de los pesos de los términos que las componen.

Por último, a fin de evitar problemas por sinonimia, en ocasiones también se usan tesauros (diccionarios de sinónimos) a la hora de determinar los términos de una petición. La elaboración de estos tesauros suele ser difícil y usualmente solo se aplica a los términos menos comunes.

Una última técnica usada en el sistema SMART se denomina «retroalimentación de relevancia» (*relevance feedback*): Parte de la determinación de pesos de las componentes de un vector petición depende de las búsquedas de usuarios anteriores, intentando alejarlas de aquellos documentos que los usuarios han considerado como no relevantes y acercándolas a aquellos que sí lo son. Esta alteración se realiza restando al vector petición a priori un vector (pequeño en magnitud, para evitar diferencias drásticas) proporcional a los vectores documento correspondientes a los documentos no relevantes y sumando otro vector (algo mayor en magnitud que el anterior) que sea proporcional a los vectores documentos considerados relevantes.

Apéndice D

Listado de códigos

```
database = IntegerDigits[
    36^^q9vwuqqx9w4b32pojqzyj1uozz3zuxmv7x630g, 2, 10*20] \
    // ArrayReshape[#, {10, 20}]& \
    // Map[AssociationThread[{
        "Cartas",       "Difícil",  "Estrategia",   "Multijugador", "Mundo Abierto",
        "Por Turnos",   "Puzles",   "PvP",          "RPG",          "Roguelike"
    }, #] &] \
    // AssociationThread[{
        "Ara Fell",                  "Baba is You",              "Brogue",
        "Chroma: Bloom and Blight",  "Divinity: Original Sin 2", "DotA 2",
        "Fell Seal: Arbiter's Mark", "Fire Emblem: Heroes",      "Iconoclasts",
        "Islands of Insight",        "Octopath Traveller",       "One Step From Eden",
        "Slay the Spire",            "Slice & Dice",             "Symphony of War",
        "Taiji",                     "Terraria",        "The Elder Scrolls V: Skyrim",
        "The Witness",               "Wargroove"
    }, #] &

database2 = IntegerDigits[
    36^^4lzfzxpizuw6yr2w3dde738byf9o1nfzi5eo4, 2, 16*12] \
    // ArrayReshape[#, {16, 12}]& \
    // Map[AssociationThread[{
        "Acquisitions Expert", "Ardent Electromancer", "Cascade Seer",
        "Changeling Outcast" , "Farisght Adept"      , "Joraga Bard",
        "Mul Daya Channelers", "Nimble Trapfinder"   , "Skyclave Cleric",
        "Tajuru Paragon"     , "Vine Gecko"          , "Werefox Bodyguard"
    }, #] &] \
    // AssociationThread[{
        "Ally"  , "Bard", "Cleric", "Druid"  , "Elf"  , "Elemental","Fox"    ,"Human",
        "Knight", "Kor" , "Lizard", "Merfolk", "Rogue", "Shaman"   ,"Warrior","Wizard"
    }, #] & \
    // Transpose

query = database \
    // First // Keys // AssociationMap[Boole@MemberQ[{"Multijugador", "RPG"}, #] &]

query2 = database2 // First \
    // Keys //AssociationMap[Switch[#, "Druid", 1, "Elf", 5, "Wizard", 2, _, 0] &]
```

```
similarity[aDatabase_Association, OptionsPattern[{Method → "Cosine"}]] \
    := Switch[OptionValue[Method],
        "Dot Product", vQuery ⊢> aDatabase // Map[Values /@ (# . vQuery) &],
        "Cosine", vQuery ⊢> aDatabase // Map[1 - Values/@CosineDistance[#, vQuery] &],
        "Pseudo-Cosine", vQuery ⊢> aDatabase // Map[
            Values /@ (# . vQuery) / Total[#] / Total[vQuery] &
        ],
        "Dice", vQuery ⊢> aDatabase // Map[
            2 * Values /@ (# . vQuery) / (Total[#] + Total[vQuery]) &
        ],
        "Product-Moment Correlation", vQuery ⊢> aDatabase // Map[
            Standardize[Values[#], Mean, 1&] . Standardize[Values[vQuery], Mean, 1&] &
        ],
        "Product-Moment Covariance", vQuery ⊢> aDatabase // Map[
            1 - CosineDistance[
                Standardize[Values[#], Mean, 1&], Standardize[Values[vQuery], Mean, 1&]
            ] &
        ],
        "Overlap", vQuery ⊢> aDatabase // Map[
            Total[Min /@ Transpose[Values /@ {#, vQuery}]] / Min[Total /@ {#, vQuery}]&
        ],
        "Spreading Activation", vQuery ⊢> aDatabase // Map[
            Total @* KeyValueMap[
                vQuery[[#1]] * #2 / Total@Transpose[aDatabase][[#1]] / Total[vQuery] &
            ]
        ]
        "QR Projection", Module[{projectionFactor =
                qrDecomposition[Transpose[aDatabase] // Values // Map[Values] // N]
                    // Chop // (AssociationThread[#[[1]] // Transpose, #[[2]]])&
                    // Select[AnyTrue[Not @* PossibleZeroQ]] // Keys // Transpose
                    // # . Transpose[#] & // Chop
            },
            vQuery ⊢> aDatabase // Map[
                1 - CosineDistance[Values[#], projectionFactor . Values[vQuery]] &]
        ]
    ]

searchQuery[aDatabase_Association,OptionsPattern[{Method→"Cosine",Threshold → 0.8}]]\
    := vQuery ⊢>
        similarity[aDatabase, Method → OptionValue[Method]][vQuery] \
            // Select[# > OptionValue[Threshold] &]

Manipulate[
    searchQuery[database, Method → method, Threshold → threshold][query] // Keys,
    {threshold, 0, 1},
    {similarity, {
        "Dot Product", "Cosine", "Pseudo-Cosine", "Dice", "Product-Moment Correlation",
        "Product-Moment Covariance", "Overlap", "Spreading Activation"
    }}
]
```

```
qrDecomposition = mA ⊢> Module[
    {
        mQ = IdentityMatrix[First[Dimensions[mA]]], mR = mA,
        mP = IdentityMatrix[Last[Dimensions[mA]]],
        vNormsSquared = Transpose[mA] // Map[(# . #) &],
        iPermIndex, mReflector
    },
    Do[
        iPermIndex = Max[First@Ordering[vNormsSquared,-1], j];
        If[PossibleZeroQ[vNormsSquared[[iPermIndex]]], Break[]];
        mR[[All, {j, iPermIndex}]] = mR[[All, {iPermIndex, j}]];
        mP[[All, {j, iPermIndex}]] = mP[[All, {iPermIndex, j}]];
        vNormsSquared[[{j, iPermIndex}]] = vNormsSquared[[{iPermIndex, j}]];
        mReflector = mR[[All, j]]
            // MapIndexed[Switch[Sign[First[#2] - j],
                -1, 0,
                0, #1 + (Sign[#1]+Boole[PossibleZeroQ[#1]]) * Sqrt[vNormsSquared[[j]]],
                1, #1
            ] &]
            // ReflectionMatrix;
        mR = mReflector . mR;
        mQ = mQ . mReflector;
        vNormsSquared -= mR[[j, All]]^2
    , {j, Min[Dimensions[mA]]}];
    {mQ, mR, mP} // Map[FullSimplify]
]

compactSvd[mA_?MatrixQ, OptionsPattern[{SingularValues → Automatic}]] := Module[
    {vSigmas, vvRightSingularVectors},
    {vSigmas, vvRightSingularVectors}
        = Eigensystem[
                ConjugateTranspose[mA] . mA,
                OptionValue[SingularValues] /. Automatic → UpTo[Infinity]
            ] // Transpose // Select[Positive @* First] // Transpose
            // Comap[{First /* Sqrt, Last /* Transpose}];
    {
        mA . vvRightSingularVectors / Threaded[vSigmas],
        DiagonalMatrix[vSigmas],
        vvRightSingularVectors
    }
]

generalizedCosineSimilarity = {lmDocumentStates, mProjectionFactor, vQuery} ⊢> \
    If[MatrixQ[lmDocumentStates], lmDocumentStates, Fold[Dot, lmDocumentStates]] \
    // Transpose // Map[1 - CosineDistance[#, mProjectionFactor . Values[vQuery]] &]

prepareQrState[aDatabase_Association, OptionsPattern[{MatrixRank → Automatic}]] := \
    qrDecomposition[Transpose[aDatabase] // Values // Map[Values] // N] // Chop \
        // AssociationThread[#[[1]] // Transpose, #[[2]]] & \
        // Select[AnyTrue[Not @* PossibleZeroQ]] // Keys \
        // Replace[OptionValue[MatrixRank], { \
            Automatic | 0 → Identity,
            n: x_Integer | UpTo[x_Integer] /; Positive[x] → (Take[#, n] &),
            n_Integer /; Negative[n] → (Drop[#, n] &)
        }] // Transpose // {Values[aDatabase] // Map[Values], Transpose[#].# // Chop} &

prepareSvdState[aDatabase_Association, OptionsPattern[{MatrixRank → Automatic}]] := \
    compactSvd[
        aDatabase // Values // Map[Values] // N,
        SingularValues → OptionValue[MatrixRank]
    ] // {{#[[2]], Transpose[#[[3]]]}, Transpose[#[[1]]]} &
```

```
addTerms =OperatorApplied[{lmOldState, mNewTerms, liOriginalSizes (*{T, D}*)}⊢>Module[
    {
        mOldK = Min@Dimensions[lmOldState[[1, 1]]],
        mOldU = Transpose[lmOldState[[2]]]
            // PadRight[#, {First[liOriginalSizes], First[liOriginalSizes]}] &,
        mOldSigma = lmOldState[[1, 1]] // PadRight[#, liOriginalSizes] &,
        mOldV = lmOldState[[1, 2]]
            //Transpose //PadRight[#, {Last[liOriginalSizes], Last[liOriginalSizes]}]&,
        mQ, mR, mP, mPk, mSigmaK, mQk
    },
    {mQ, mR, mP} = qrDecomposition[
        (IdentityMatrix[Length[mOldV]] - mOldV . Transpose[mOldV])
        . Transpose[mNewTerms]];
    {mQ, mR} = AssociationThread[Transpose[mQ], mR // Chop]
        // Select[AnyTrue[Not@*PossibleZeroQ]] // {Keys[#] // Transpose, Values[#]} &;
    {mPk, mSigmaK, mQk} = compactSvd[
        ArrayFlatten[{{mOldSigma, 0}, {mNewTerms . mOldV, mP . Transpose[mR]}}],
        SingularValues → UpTo[mOldK]
    ];
    {
        {
            mSigmaK,
            Transpose[ArrayFlatten[{{mOldV, mQ}}] . mQk]
        },
        Transpose[
            BlockDiagonalMatrix[{mOldU, IdentityMatrix[Length[mPk]-Length[mOldU]]}].mPk
        ]
    } // Chop
], {3, 1, 2}]

addDocuments = OperatorApplied[{lmOldState, mNewDocuments, liOriginalSizes} ⊢> Module[
    {
        mOldK = Min@Dimensions[lmOldState[[1, 1]]],
        mOldU = Transpose[lmOldState[[2]]]
            // PadRight[#, {First[liOriginalSizes], First[liOriginalSizes]}] &,
        mOldSigma = lmOldState[[1, 1]] // PadRight[#, liOriginalSizes] &,
        mOldV = lmOldState[[1, 2]]
            //Transpose //PadRight[#, {Last[liOriginalSizes], Last[liOriginalSizes]}]&,
        mQ, mR, mP, mPk, mSigmaK, mQk
    },
    {mQ, mR, mP} = qrDecomposition[
        (IdentityMatrix[Length[mOldU]] - mOldU . Transpose[mOldU]) . mNewDocuments];
    {mQ, mR} = AssociationThread[Transpose[mQ], mR // Chop]
        // Select[AnyTrue[Not@*PossibleZeroQ]] // {Keys[#] // Transpose, Values[#]} &;
    {mPk, mSigmaK, mQk} = compactSvd[
        ArrayFlatten[{
            {mOldSigma, Transpose[mOldU] . mNewDocuments},
            {0, mP . Transpose[mR]}
        }],
        SingularValues → UpTo[mOldK]
    ];
    {
        {
            mSigmaK,
            Transpose[
               BlockDiagonalMatrix[{mOldV,IdentityMatrix[Length@mQk-Length@mOldV]}].mQk
            ]
        },
        Transpose[ArrayFlatten[{{mOldU, mQ}}] . mPk]
    } // Chop
], {3, 1, 2}]
```

```
editWeights = OperatorApplied[{lmOldState, mWhichTerms, mWhatChanges, liOriginalSizes}\
    ⊢> Module[{
        mOldK = Min@Dimensions[lmOldState[[1, 1]]],
        mOldU = Transpose[lmOldState[[2]]]
            // PadRight[#, {First[liOriginalSizes], First[liOriginalSizes]}] &,
        mOldSigma = lmOldState[[1, 1]] // PadRight[#, liOriginalSizes] &,
        mOldV = lmOldState[[1, 2]]
            //Transpose //PadRight[#, {Last[liOriginalSizes], Last[liOriginalSizes]}]&,
        mQy, mRy, mPy, mQz, mRz, mPz, mPk, mSigmaK, mQk, mBpart
    },
    {mQy, mRy, mPy} = qrDecomposition[
        (IdentityMatrix[Length[mOldU]] - mOldU . Transpose[mOldU]) . mWhichTerms];
    {mQy, mRy} = AssociationThread[Transpose[mQy], mRy // Chop]
        // Select[AnyTrue[Not@*PossibleZeroQ]] // {Keys[#] // Transpose, Values[#]} &;
    {mQz, mRz, mPz} = qrDecomposition[
        (IdentityMatrix[Length[mOldV]] - mOldV . Transpose[mOldV]) . mWhatChanges];
    {mQz, mRz} = AssociationThread[Transpose[mQz], mRz // Chop]
        // Select[AnyTrue[Not@*PossibleZeroQ]] // {Keys[#] // Transpose, Values[#]} &;
    mBpart = ArrayFlatten[{{mOldU . Transpose[mWhichTerms]}, {mRy . Transpose[mPy]}}]
        . ArrayFlatten[{{mWhatChanges . Transpose[mOldV], mRz . Transpose[mPz]}}];
    {mPk, mSigmaK, mQk} = compactSvd[
        mBpart + PadRight[mOldSigma, Dimensions[mBpart]],
        SingularValues → UpTo[mOldK]
    ];
    {
        {
            mSigmaK,
            Transpose[ArrayFlatten[{{mOldV, mQz}}] . mQk]
        },
        Transpose[ArrayFlatten[{{mOldU, mQy}}] . mPk]
    } // Chop
], {4, 1, 2, 3}]
```

Índice de figuras

Índice de tablas

Bibliografía

[1] Zhaojun Bai, James Demmel, Jack Dongarra, Axel Ruhe y Henk van der Vorst, *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*, Society for Industrial and Applied Mathematics, sección «Decompositions» 2000.

[Citado en la página 38]

[2] Michael W. Berry, Susan T. Dumais y Gavin W. O'Brien, *Using Linear Algebra for Intelligent Information Retrieval*, SIAM Review, Vol. 37, Nº 4, pp. 573–595, 1999.

[Citado en la página 45]

[3] Michael W. Berry, Zlatko Drmac y Elizabeth R. Jessup, *Matrices, Vector Spaces, and Information Retrieval*, Society for Industrial and Applied Mathematics, 1999.

[Citado en las páginas 1, 3, 8, 25 y 26]

[4] Chris Buckley, Gerald Salton, James Alan y Amit Singhal, *Automatic Query Expansion Using SMART: TREC 3*, Text Retrieval Conference, 1994.

[Citado en la página 25]

[5] Alan K. Cline y Inderjit S. Dhillon, *Handbook of Linear Algebra*, CRC Press, Segunda Edición, Cap. 45, 2014.

[Citado en las páginas 39 y 42]

[6] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer y Richard Harshman, *Indexing by Lantent Semantic Analysis*, Journal of the American Society for Information Science, 41, pp 391–407, 1990.

[Citado en la página 26]

[7] James Demmel, *Applied numerical linear algebra*, SIAM, Philadelphia, 1997.

[Citado en la página 39]

[8] Carl Eckart y Gale Young, *The approximation of one matrix by another of lower rank*, Psychometrika, Vol. 1, Nº 3, pp 211–218, 1936.

[Citado en la página 40]

[9] Gene H. Golub y Charles F. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, 4ª edición, 2013.

[Citado en la página 28]

[10] William P. Jones y George W. Furnas, *Pictures of Relevance: A Geometric Analysis of Similarity Measures*, Journal of the American Society for Information Science, 1987.

[Citado en la página 9]

[11] Donald E. Knuth, *Two Notes on Notation*, Amer. Math. Monthly 99, Nº 5, 403–422, 1992.

Versión archivada con identificador `arXiv:math/9205211 [math.HO]`

URL `https://arxiv.org/abs/math/9205211v1`

Archivado el 1 de mayo de 1992, accedido el 11 de abril de 2025.

[Citado en la página 27]

[12] Warren S. McCulloch y Walter Pitts, *A logical calculus of the ideas immanent in nervous activity* The Bulletin of Mathemtical Biophysics, 5, pp 1135–133, 1943.

[Citado en la página 49]

[13] Leon Mirsky, *Symmetric gauge functions and unitarily invariant norms*, Q. J. Math, 11, pp. 50–59, 1960.

[Citado en la página 40]

[14] Gillaume Rabusseau, *SVD, Matrix Norms and Low Rank Approximation Theorem*, lección 5 del curso IFT 6760A, universidad de Montreal, 2020.

[Citado en la página 40]

[15] Prajit Ramachandran, Barret Zoph, Quoc V. Le, *Searching for Activation Functions*, v2, 2017.

Versión archivada con identificador `arXiv:1710.05941v2 [cs.NE]`

URL `https://arxiv.org/abs/1710.05941v2`

Archivado el 27 de octubre de 2017, accedido el 28 de diciembre de 2024.

[Citado en la página 49]

[16] Geralt Salton y Chris Buckley, *Term Weighting Approaches in Automatic Text Retrieval*, Inf. Process. Manag. Vol. 24, pp 513–523, 1988.

[Citado en la página 4]

[17] Gerard Salton y Michael E. Lesk, *The SMART Automatic Document Retrieval System—An Illustration*, Communications of the ACM Vol. 8, Nº 6, pp 391–398, 1965.

[Citado en las páginas 1 y 50]

[18] Gerard Salton y Michael J. McGill, *Introduction to Modern Information Retrieval*, McGraw-Hill Computer Science Series, Cap. 4, 1983.

[Citado en las páginas 1, 25 y 50]

[19] David S. Wakins, *Fundamentals of Matrix Computations*, Wiley-Interscience, Segunda Edición, Cap. 3–4, 2002.

[Citado en la página 39]

[20] Dian I. Witter, *Downdating the latent semantic indexing model for information retrieval*, Universidad de Tennessee, 2002.

[Citado en las páginas 42 y 44]

[21] Da Zheng, *Lecture Notes from October 11, 2012*, curso *Matrix Theory*, Math6304, Universidad de Houston, 2012.

[Citado en la página 40]

Webgrafía

[22] 3Blue1Brown, *But what is a neural network? | Deep learning chapter 1*

URL https://youtu.be/aircAruvnKk

Publicado el 5 de octubre de 2017, accedido el 28 de diciembre de 2024.

[Citado en la página 49]

[23] Avelon Pang, *CRUD Operations Eplained*

URL https://scribe.rip/geekculture/crud-operations-explained-2a44096e9c88 (original en Medium)

Publicado el 4 de junio de 2021, accedido el 17 de mayo de 2025.

[Citado en la página 44]

[24] Cornelis J. van Rijsbergen, *Information Retrieval*, Butterworths, 2ª edición, cap. 5.

URL https://www.dcs.gla.ac.uk/Keith/Chapter.5/Ch.5.html

Accedido el 25 de marzo de 2025.

[Citado en la página 14]

[25] DammitMatt, en respuesta al post de Suspicious-Web-9246 *[Request] Theoretically, what is the size of internet right now?*, r/theydidthemath, Reddit.

URL https://www.reddit.com/r/theydidthemath/comments/18xjm75/comment/kg4kwn4

Publicado el 3 de enero de 2024, accedido el 27 de mayo de 2025.

[Citado en la página 1]

[26] Graham Hutton y Mark P. Jones, *Requently Asked Questions for compl.lang.functional*, sección 3.2 «Currying», Universidad de Nottingham, versión de noviembre de 2002.

URL https://people.cs.nott.ac.uk/pszgmh/faq.html#currying

Accedido el 7 de abril de 2025.

[Citado en la página 35]

[27] Gregrory Gundersen, *Proof of the Singular Value Decomposition*.

URL https://gregorygundersen.com/blog/2018/12/20/svd-proof/

Publicado el 20 de diciembre de 2018, accedido el 17 de abril de 2025.

[Citado en la página 38]

[28] IDC, Statista y otras fuentes, *Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2023, with forecasts from 2024 to 2028*.

URL https://www.statista.com/statistics/871513/worldwide-data-created

Publicado el 31 de mayo de 2024, accedido el 27 de mayo de 2025.

[Citado en la página 1]

[29] Lauren Morton, *Co-op Skyrim is finally real and works almost exactly how I hoped*, PC Gamer.

URL https://www.pcgamer.com/co-op-skyrim-is-finally-real-and-works-almost-exactly-how-i-hoped

Publicado el 19 de julio de 2022, accedido el 3 de diciembre de 2024.

[Citado en la página 26]

[30] Matthew Leingang, respuesta 2276493 a la pregunta 2276374 *Strang's proof of SVD and intuition behind matrices U and V*, Math Stack Exchange.

URL https://ao.ngn.tf/exchange/math/questions/2276374/#2276493 (Original en Math.SE)

Publicado el 11 de mayo de 2017, accedido el 17 de mayo de 2025.

[Citado en la página 38]

[31] Retro Game Mechanics Explained, *Pokémon Sprite Compression Explained*, sección que comienza en el minuto 5:45.

URL https://youtu.be/aF1Yw_wu2cM

Publicado el 30 de junio de 2020, accedido el 20 de noviembre de 2024.

[Citado en la página 48]

[32] Steven W. Smith, *The Scientist and Engineer's Guide to Digital Signal Processing*, cap. 26–27, secciones «Neural Network Architecture», «Delta Compression» y «Run-Length Encoding», 1997.

URLs respectivas:
https://www.dspguide.com/ch26/2.htm
https://www.dspguide.com/ch27/2.htm
https://www.dspguide.com/ch27/4.htm

Accedido el 29 de enero de 2024.

[Citado en las páginas 48 y 49]