MinHash, LSH y el Problema de las Semejanzas.
Organización de Datos (75.06)

Introducción:

Este apunte trata principalmente sobre como calcular la semejanza entre documentos no-estructurados, es decir documentos de texto o páginas web. Este problema puede aplicarse a innumerables situaciones como por ejemplo:

- detectar plagios
- encontrar páginas web duplicadas (mirrors)
- encontrar páginas web muy similares (versiones)
- clustering

Representando un documento:

Para saber si dos documentos son iguales o similares es necesario compararlos. Realizar una comparación byte a byte sirve para detectar documentos idénticos pero desafortunadamente no nos da una idea de semejanza. Para poder calcular la semejanza entre documentos es necesario reducir los documentos a un conjunto de items, tokens o elementos que luego podamos comparar. Para realizar esta representación hay varias opciones.

a) Representar un documento por la lista de palabras que contiene

Se extraen las palabras (tokens) del documento y el mismo queda representado por la lista de palabras que contiene. Este enfoque tiene como problema que para documentos muy largos la semejanza será muy grande ya que coincidirán en muchas palabras aun si son muy diferentes. No tiene en cuenta por ejemplo el orden de las palabras, como se escriben las mismas en el contexto de una frase, etc.

D1: "el gato es blanco y el perro come carne"
D2: "el perro es blanco y el gato come carne"
D3: "el gato es blanco y el perro tito come carne"

¿Cuáles son los dos documentos mas parecidos?

b) Representar un documento mediante k-gramas (k-shingles)

En este caso lo que se hace es extraer de cada documento todos sus k-gramas y representar al documento mediante la lista de k-gramas que contiene.

Por ejemplo si el documento es "hola que tal"
Y usamos k=3 quedaria representado como

hol,ola,la ,a q, qu,que,ue ,e t, ta,tal

Una decisión a tomar es si debe o no considerarse a los separadores y espacios en blanco al extraer los k-shingles.

El tamaño de "k" es crítico, un k muy chico va a necesitar menos espacio pero hará mas probable que documentos diferentes parezcan similares. Un k muy grande evitará estos falsos positivos pero

implicará mayor espacio para almacenar los shingles. El espacio necesario se puede aproximar ,en bytes, como k*document_size.

En general un k entre 5 y 9 es usado para la mayoría de los documentos de texto y páginas web.

Para usar menos espacio al almacenar los k-shingles se usa una función de hashing que mapee cada k-shingle a un entero de 4 bytes. De esta forma aun con k=9 el tamaño maximo de los shingles es 4*document_size.

Notar que usar 4 bytes para representar shingles con k>4 no es lo mismo que usar shingles de 4 bytes. El espacio es el mismo pero no hay 2^32 shingles de 4 bytes. Si consideramos que solo hay 25 letras en el alfabeto la cantidad de shingles de 4 bytes posibles es 25^4 que es mucho menor que 2^32.

Semejanza entre documentos:

Una vez que tenemos los documentos representados como una lista de k-shingles es posible calcular la semejanza entre dos documentos usando la distancia de Jaccard:

SIM(A,B) = A&B / A|B

Es decir el cociente entre la interseccion y la union de los dos conjuntos de shingles.

Ejemplo:

sea A = (perro, auto, avion, camion)
B = (perro, gato, auto, flores, edificios)

JAC(A,B) = 2/7

El problema que tenemos es que la lista de shingles es, potencialmente muy larga y ademas ocupa mucho espacio, es incluso mas grande que el documento mismo y por lo tanto su almacenamiento es problematico. A continuación estudiaremos como calcular la semejanza de Jaccard usando mucho menos espacio.

Min Hash:

El concepto de MinHash es muy simple. Se eligen "n" funciones de hashing y se aplica cada una de estas "n" funciones de hashing a todos los k-shingles de un documento eligiendo el mínimo valor para cada función de hashing.

Antes de explicar como funciona el algoritmo de MinHash vamos a partir de la matriz que representa que "shingles" aparecen en cada uno de los documentos. A esta matriz la llamamos la "matriz caracteristica".

Ejemplo:

Shingle	D1	D2	D3	D4
S0	1	1	1	0
S1	0	1	0	1
S2	0	0	0	1

Shingle	D1	D2	D3	D4
S3	1	0	1	0
S4	1	0	0	0
S5	0	0	1	0

Esta matriz puede usarse para calcular la semejanza de Jaccard entre dos documentos cualesquiera. Simplemente hay que considerar a cada documento como una columna y analizar las filas en las cuales ambos documentos tienen un 1 y las filas en las cuales alguno de los documentos tiene un 1 y el otro un cero. Las filas en las cuales ambos documentos tienen ceros no interesan pues no son parte ni de la union ni de la interseccion entre los documentos.

Sea:

x = cantidad de filas en donde ambos documentos tienen 1
y = cantidad de filas en donde un documento tiene 1 y el otro 0 o vice-versa

Jaccard(D1,D1) = X / (X+Y)

Ya que X representa la interseccion entre los documentos (ambos con 1)
Y la suma de X+Y representa la union entre los documentos.

Para entender el método MinHash vamos a comenzar con el concepto teórico: Para construir el MinHash lo que se hace es tomar una permutacion al azar de las filas de la matriz caracteristica.

En nuestro caso podemos, por ejemplo tomar la permutacion S2,S1,S5,S4,S3,S0

Permutación S2,S1,S5,S4,S3,S0

Shingle	D1	D2	D3	D4
S2	0	0	0	1
S1	0	1	0	1
S5	0	0	1	1
S4	1	0	0	0
S3	1	0	1	0
S0	1	1	1	0

La función Min Hash se construye ahora para cada documento contando la primera fila en la cual aparece un "1"

Es decir

MinHash(D1) = 3
MinHash(D2) = 1
MinHash(D3) = 2
MinHash(D4) = 0

Calculemos ahora la probabilidad de que para dos documentos diferentes la funcion MinHash de el mismo resultado.

Los casos en los cuales ambos documentos tienen ceros en la misma fila no interesan ya que se irian salteando. Para que el MinHash de igual ambos documentos deben tener su primer 1 en la misma fila.

La cantidad de casos favorables es X: Cantidad de filas en las que ambos documentos tienen 1
Y la cantidad de casos desfavorables es Y: Cantidad de filas en las que un documento tiene 1 y el otro 0.

Es decir que la probabilidad es: X/(X+Y)

Y esto es igual a la distancia de Jaccard entre los documentos.

De aqui se deduce algo muy importante: La probabilidad de que el MinHash de dos documentos sea igual es igual a la distancia de Jaccard entre dos documentos. Es decir que se puede usar el MinHash para comparar a los documentos en lugar de comparar todos los Shingles.

Para que la comparacion sea mas efectiva en lugar de usar solo un MinHash se usan varios:

Si tomamos "n" permutaciones al azar podemos construir varios MinHashes para cada documento y de esta forma representar a cada documento con un vector de MinHashes en lugar de un vector de Shingles. Como los Shingles pueden, potencialmente ser muchisimos tomando una cantidad acotada de MinHashes se logra reducir muchisimo la dimensionalidad de los datos.

Ejemplo:

MinHashes	D1	D2	D3	D4
MinHash1	3	1	3	1
MinHash2	1	0	1	2
MinHash3	2	3	0	2

Y ahora la distancia entre documentos la calculamos comparando los MinHashes de los mismos es decir:

D(D1,D3) = 2/3
D(D1,D2) = 0/3
D(D2,D4) = 1/3

Cuantas mas funciones de MinHash se usen mas precisa sera la comparación.

Implementación de MinHash:

Realizar "n" permutaciones al azar de la matriz caracteristica es algo costoso, en su lugar se pueden usar "n" funciones de hashing que generen resultados entre 0 y K-1 siendo K el número de Shingles.

Por ejemplo podemos usar:

H1 = 3x + 1 mod 6
H2 = 5x + 1 mod 6
H3 = 7x + 1 mod 6

En donde "x" es el numero de fila para el shingle correspondiente. Y la funcion solo se aplica si la matriz caracteristica tiene un 1 para ese documento en esa fila.

Comenzamos con

MinHashes	D1	D2	D3	D4
H1	INF	INF	INF	INF
H2	INF	INF	INF	INF
H3	INF	INF	INF	INF

Y tomamos la primera fila de la matriz caracteristica:

Shingle	D1	D2	D3	D4
S0 (x=1)	1	1	1	0

Vemos que hay que aplicar H1, H2 y H3 para D1, D2 y D3 (no para D4 que tienen un 0)

H1 = 3*1 + 1 mod 6 = 4
H2 = 5*1 + 1 mod 6 = 0
H3 = 7*1 + 1 mod 6 = 2

Actualizamos:

MinHashes	D1	D2	D3	D4
H1	4	4	4	INF
H2	0	0	0	INF
H3	2	2	2	INF

Ahora tomamos el segundo shingle:

S1 (x=2)	0	1	0	1

Y calculamos H1, H2 y H3 para D2 y D4

H1 = 3*2 + 1 mod 6 = 1
H2 = 5*2 + 1 mod 6 = 5
H3 = 7*2 + 1 mod 6 = 3

Ahora si H1,H2 o H3 es MENOR que el valor que tenemos actualmente para el MinHash actualizamos y sino no.

MinHashes	D1	D2	D3	D4
H1	4	1	4	1
H2	0	0	0	5
H3	2	2	2	3

Y asi seguimos por cada fila hast que nos quedan construidos todos los MinHashes.

Notar que es conveniente tomar una cantidad de funciones de hashing que sea un número primo para mejor dispersión. (6 es un muy mal numero)

El problema de los k-parecidos:

Este problema consiste en encontrar los k-pares de documentos mas parecidos sobre una colección de documentos. La solución por fuerza bruta consiste en calcular la semejanza de cada documento contra todos los demás y luego elegir las k semejanzas mas grandes. Desafortunadamente esta solucion para millones de documentos es impracticable ya que podría tardar incluso días (la maldición de la dimensionalidad).

Es necesario entonces realizar una aproximación que permita encontrar "casi" todos los k-pares mas parecidos de forma eficiente. Para esto se hace uso del hashing localmente sensible (LSH)

LSH (Local Sensitive Hashing):

La idea de LSH es muy simple: construir una función de hashing en la cual las claves similares tiendan a almacenarse en el mismo bucket. Es posible realizar esto a partir de HashMin.

Partimos de una matriz en donde cada columna es un documento y cada fila es una de las funciones de hashing y en cada posicion se almacena el hashmin para esa funcion y ese documento.

Se particiona esta matriz en "b" secciones de "r" filas cada una. Por ejemplo si se usaron 20 funciones de hashing para hashmin podemos usar b=5 y r=4.

Luego dentro de cada seccion "b" aplicamos una función de hashing que mapee los r elementos de cada documento a un número grande de buckets. El proceso se repite para cada seccion.

Una vez realizado esto se calcula la semejanza unicamente entre documentos que han coincidido en un mismo bucket para algunas de las "b" secciones.

Variando "b" y "r" puede parametrizarse la cantidad de falsos positivos y falsos negativos que puede darnos este método.

Ejemplo con 6 documentos y 9 funciones de hashing:

	D1	D2	D3	D4	D5	D6
H1	2	**1**	3	4	**1**	2
H2	5	**3**	1	2	**3**	7
H3	1	**1**	2	2	**1**	1

	D1	D2	D3	D4	D5	D6
H4	4	**2**	**2**	2	1	1
H5	1	**3**	**3**	2	2	1
H6	3	**3**	**3**	2	1	3
H7	**3**	**2**	**3**	3	**2**	1
H8	**1**	**4**	**1**	3	**4**	4
H9	**1**	**2**	**1**	3	**2**	2

Si particionamos las filas de la tabla en 3 grupos de 3 filas cada uno y usamos alguna funcion de hashing que mapee los valores F1,F2,F3 a por ejemplo un espacio de direcciones 1..1000 vamos a ver que para el primer bloque los documentos 2 y 5 van al mismo bucket pues coinciden sus 3 valores (1,3,1). En el segundo bloque tenemos D2-D3 y en el tercer bloque D1-D3 y D2-D5.

Tambien puede, dependiendo de la función de hashing, algún falso positivo producir dos o mas documentos en un mismo bucket pero esto no es un problema, solo son algunos pares mas a analizar.

Es decir que los tres pares de documentos a analizar son D2-D5, D2-D3 y D1-D3. Entre esos 3 pares de documentos tendremos, probablemente, a las mayores semejanzas.

Para calcular b y r sabemos que:

Sea t la semejanza entre 2 documentos.

La probabilidad de que coincidan los r valores de un bloque es

t^r

La probabilidad de que no coincida alguno de los r valores es:

1 – (t^r)

La probabilidad de que no coincidan para ningun bloque b es:

[1-(t^r)] ^ b

Por lo tanto se sabe que la probabilidad de que dos documentos hasheen al mismo bucket es:

1 - [(1 – t^r) ^b]

Donde t es la semejanza de Jaccard entre dos documentos.
Con esta fórmula podemos tabular la probabilidad de que los documentos sean seleccionados en funcion de "t" para distintos valores de b y r y de esta forma elegir valores para b y r que sirvan. Por ejemplo podemos querer que si t>0,5 la probabilidad de ser seleccionados sea de al menos un 80%.

LSH para otras Distancias

La semejanza entre documentos o elementos de un conjunto puede verse como la distancia n-dimensional entre los mismos. A menor distancia mayor semejanza.

La distancia entre dos puntos n-dimensionales se define como d(x,y) y debe cumplir las siguientes propiedades:

d(x,y) >= 0 para todo x,y
d(x,y) = 0 si y solo si x == y
d(x,y) = d(y,x)
d(x,y) <= d(x,z) + d(z,y) -desigualdad triangular-

La distancia puede definirse tanto para espacios euclideanos como para espacios no-euclideanos. Basicamente un espacio es euclideano si es posible calcular el promedio o centroide entre un conjunto de puntos.

La distancia euclideana mas conocida se calcula como:

SQRT (SUM [(xi-yi) ^ 2])

Es facil demostrar que esta fórmula cumple con las 4 propiedades pedidas para una distancia.

La fórmula es en realidad un caso particular de la mas general:

SQRN (SUM [(xi-yi) ^ N])

Cuando n=2 tenemos la fórmula anterior. Cuando n=1 tenemos la llamada "distancia Manhattan)

La función de semejanza de Jaccard tambien puede usarse como distancia pero hay que tener en cuenta que Jaccard(X,Y) crece cuando X mas se acerca a Y por lo tanto la distancia de Jaccard se define como:

d(x,y) = 1 – Jaccard(x,y)

Notar que la distancia de Jaccard no es euclideana.

Otra distancia no-euclideana que suele usarse para strings es la distancia de edicion, es decir cuantas inserciones o borrados hay que hacer para convertir un string en otro.

Ejemplo d("abcd","bdg") = 3 (borrar a, borrar c, agregar g)

Otra distancia muy popular es la de Hamming, que basicamente indica cuantos elementos del vector son diferentes. En general se usa sobre vectores binarios (tiras de bits) y mide cuantos bits difieren.

Ej d(1101100, 1000101) = 3

La distancia de Hamming cuando la cantidad de bits es muy grande puede estimarse usando una familia de funciones LSH muy simple. Tomamos "m" bits al azar de los vectores y solo comparamos esos bits.

Finalmente una distancia que se usa muchisimo es la del coseno entre dos vectores

cos(x,y) = x . y / |x| |y|

Cuando los vectores x e y tiene muchisimas dimensiones es posible reducirlos y estimar los cosenos mediante una familia de funciones LSH.

Ejemplo sean:

v1 = (2,3,-4,1,2,5)
v2 = (1,1,3,-2,3,4)

y queremos llevarlos de 6 dimensiones a 3.

Elegimos 3 vectores al azar formados por -1 o +1 del tamaño de V

h1 = (-1,+1,-1,-1,-1,+1)
h2 = (+1,+1,+1,-1,+1,-1)
h3= (-1,+1,-1,+1,+1,-1)

Y ahora podemos representar a v1 como

(sign(h1*v1), sign(h2*v2), sign(h3*v3))

h1 * v1 = -2 + 3 +4 -1 -2 +5 = 7 sign(7) = +1
h2 * v1 = 2 + 3 + 4 -1 +2 -5 = +1
h3*v1 = -2+3+4+1+2-5 = +1

(si alguna cuenta da 0 puede elegirse +1 o -1 al azar)

es decir que v1r quedaria (+1,+1,+1)

Se hace lo mismo con v2 y luego el coseno entre v1r y v2r es una estimacion del coseno entre V1 y V2. Cuantas mas funciones de hashing se usen mas eficiente la aproximacion.