Busca por Similaridade

Gerência de Dados na Web Ediçao Big Data 2013-02

Prof. Altigran Soares da Silva

Baseado nos Slides do Professor Jeffrey Ullman

Busca por Similaridade

- Muitos problemas de Mineração na Web consistem em encontrar conjuntos ***similares***:
  - Classificação de páginas com conteúdo similar
  - Determinar usuários com gostos similares para filmes
    - Problema dual: filmes que tem os mesmos fans
  - Encontrar imagens de objetos relacionados
- A melhor técnica a aplicar depende:
  - Se procuramos itens altamente similares ...
  - ou procuramos itens com alguma similaridade

Exemplo: Documentos Similares

- Documentos contendo texto similar
- Caso simples: documentos idênticos, caractere a caractere
- Caso geral: documentos similares, algumas partes iguais, não necessariamente na mesma ordem

Exemplo: Docs. Similares (2)

- Dada uma coleção de documentos, por exemplo, a Web, encontrar pares de documentos com texto em comum
  - Sites espelho ou quase espelho
    - Evitar mostrar espelhos no resultado da busca
  - Detecção de Plágio
  - Artigos similares em vários sites de notícias
    - Aplicação: agrupar artigos sobre o mesmo assunto

Três técnicas essenciais

- Geração de *Shingles* : converter documentos em conjuntos
- Minhashing: Converter grandes conjuntos para *assinaturas* curtas, preservando a similaridade
- LSH (Locality-sensitive hashing) : foco em pares de assinaturas que são provalmente similares

Shingles

- k-shingle (ou k-grama) de um documento é uma sequencia de k caracteres que aparecem no documento
- k=2; doc = abcab
  - Conjunto de 2-shingles = {ab, bc, ca}
  - Bag de 2-shingles = {ab, bc, ca, ab}
- Um documento pode ser representado por um conjunto de k-shingles

Shingles (2)

- Assumimos que docs que tem muitos shingles em comum tem um texto similar, mesmo que apareça em ordem diferente
- Cuidados: escolher k suficientemente grande, senão a maior parte dos documentos terão todos os shingles
  - k = 5 para documentos curtos; k=10 para documentos longos

Shingles (3)

- Para comprimir shingles longos, pode-se representá-los com em um hash, por exemplo, de 4 bytes
- Um documento será representado por um conjunto de valores de hash de seus k-shingles.
- Dois documentos poderiam ter valores de hash em comum mesmo que não tivessem shingles em comum. Isso seria raro.

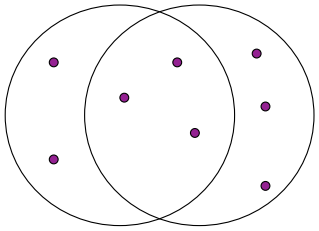
Similaridade de Conjuntos

- Muitos problemas de similaridade podem ser descritos como o problema de se encontrar subconjuntos com uma intersecção significativa em um conjunto universal
- Exemplos:
  - Documentos representados por shingles ou assinaturas de shingles
  - Produtos ou clientes similares

Similaridade de Jaccard

- Sejam dois conjuntos C_1 e C_2 a similaridade de Jaccard entre C_1 e C_2 e dada por

$$\text{Sim}(C_1, C_2) = |C_1 \cap C_2| / |C_1 \cup C_2|$$



Sim (C_1, C_2) = 3/8

Conjuntos e Matrizes Booleanas

- Linhas = elementos do conjunto universal
- Colunas = conjuntos
- Linha *e*, coluna *S* é 1 sss *e* ocorre em *S*
- Matriz tipicamente esparsa

Conjuntos e Matrizes Booleanas (2)

C_1 = {p,q,u,v}
C_2 = {r,s,t}
C_3 = {p,u,v}
C_4 = {q,r,s,t}

	C_1	C_2	C_3	C_4
p	1	0	1	0
q	1	0	0	1
r	0	1	0	1
s	0	1	0	1
t	0	1	0	1
u	1	0	1	0
v	1	0	1	0

Conjuntos e Matrizes Booleanas (3)

- Na realidade, pode não ser vantajoso representar os dados por um matriz booleana
- Matrizes esparsas são geralmente melhor representadas pela lista de suas posições que não têm valores zero
- Mas, conceitualmente, a matriz é uma abstração útil.

Similaridade entre Colunas

C_1	C_2	
0	1	*
1	0	*
1	1	* *
0	0	
1	1	* *
0	1	*

Jaccard dos conjuntos de linhas que têm 1.

Sim (C_1, C_2) = 2/5 = 0.4

Similaridade entre Colunas (2)

- Para 2 colunas C_1 e C_2, 4 tipos de linhas

	C_1	C_2
a	1	1
b	1	0
c	0	1
d	0	0

- Note que: Sim (C_1, C_2) = a /(a +b +c).

Similaridade entre Colunas (3)

- 1: Computar assinaturas das colunas, ou seja, sumários das colunas
- 2: Examinar pares de assinaturas para encontrar assinaturas similares
  - Existe uma correlação entre similaridade de assinaturas e colunas
- 3: Verificar se as colunas com assinaturas similares são realmente similares

Similaridade entre Colunas (4)

- Comparar todos os pares de assinatura pode tomar muito tempo e/ou muito espaço
  - Aplicação do Locality-Sensitive Hashing.
- Estes métodos podem gerar falsos negativos e até mesmo falsos positivos
  - Verificação adicional pode ser necessária

Assinaturas

- Ideia principal: usar hashing para gerar uma representação compacta ou assinatura Sig (C) de cada coluna C, tal que:
  - Sig(C) é compacta o suficiente para que se possa representar na MP uma assinatura de cada coluna
  - Sim(C_1, C_2) pode ser determinada pela "similaridade" entre Sig (C_1) e Sig (C_2).

Minhashing

- Seja h(C) um função de hashing que retorna a primeira linha onde a coluna C tem valor 1, em uma permutação qualquer das linhas da matriz.
- A assinatura é construída usando várias funções diferentes deste tipo, ou seja, com várias permutações aleatórias

Exemplo Minhashing

1	4	3
3	2	4
7	1	7
6	3	6
2	6	1
5	7	2
4	5	5

Matriz de Entrada

1	0	1	0
1	0	0	1
0	1	0	1
0	1	0	1
0	1	0	1
1	0	1	0
1	0	1	0

Matriz de assinaturas *M*

2	1	2	1
2	1	4	1
1	2	1	2

Propriedade importante

- A probabilidade de $h(C_1)=h(C_2)$, considerando todas as permutações de linhas é uma aproximação de $Sim(C_1, C_2)$
- Ambas são a /(a +b +c)!
- Porque?
  - Verifica as colunas permutadas C_1 e C_2 até encontrar 1.
  - Se é uma linha tipo a então $h(C_1)=h(C_2)$. Se é uma linha tipo b ou c, então não.

Similaridade de Assinaturas

A similaridade de assinaturas é fração das funções de hash em que elas concordam.

Exemplo Minhashing

1	4	3
3	2	4
7	1	7
6	3	6
2	6	1
5	7	2
4	5	5

Matriz de Entrada

1	0	1	0
1	0	0	1
0	1	0	1
0	1	0	1
0	1	0	1
1	0	1	0
1	0	1	0

Matriz de Assinaturas *M*

2	1	2	1
2	1	4	1
1	2	1	2

Similaridades:

	1-3	2-4	1-2	3-4
Coluna	0.75	0.75	0	0
Assin.	0.67	1.00	0	0

Assinaturas Minhashing

- Seja um número qualquer (ex., 100) de permutações aleatórias de linhas
- A assinatura de conjunto/coluna C, Sig(C), pode ser vista como um vector.
- Sig(C)[i] = é a primeira linha da coluna C que tem 1 na i-ésima permutação

Assinaturas Minhashing (2)

Matriz de Entrada

1	0	1	0
1	0	0	1
0	1	0	1
0	1	0	1
0	1	0	1
1	0	1	0
1	0	1	0

Elementos possíveis

Conjuntos

Matriz de Assinaturas *M*

2	1	2	1
2	1	4	1
1	2	1	2

Permutações

Conjuntos

Implementação

- Suponha um bilhão de linhas
- É difícil gerar um permutação aleatória de 1 até 1 bilhão
- A representação de cada permutação aleatória demanda 1 bilhão de entradas
- Acesso às linhas nas ordens permutadas leva a thrashing
  - Thrashing = uso inadequado da mem. virtual

Implementação – (2)

- Aproximação razoável da permutação de linhas: toma-se 100 (?) funções de hashing independentes (h_1, h_2, ..., h_n)
- Para cada coluna C e cada função h_i manter uma entrada M(i,C).
- Intenção é fazer com que M(i, c) seja o menor valor de h_i(r) para o qual a coluna C tem 1 na linha r.
- Ou seja, h_i(r) dá a ordem das linhas na i-ésima permutação.

Implementação – (3)

para cada linha *r*

  para cada coluna *C*

    se C tem 1 na linha *r*

      para cada função de hash h_i **faça**

        se $h_i(r)$ tem o valor menor que $M(i, C)$

        então $M(i, C) := h_i(r)$;

Exemplo

Linha	C1	C2
1	1	0
2	0	1
3	1	1
4	1	0
5	0	1

$h(x) = x \bmod 5$
$g(x) = 2x+1 \bmod 5$

	Sig1	Sig2
$h(1) = 1$	1	-
$g(1) = 3$	3	-
$h(2) = 2$	1	2
$g(2) = 0$	3	0
$h(3) = 3$	1	2
$g(3) = 2$	2	0
$h(4) = 4$	1	2
$g(4) = 4$	2	0
$h(5) = 0$	1	0
$g(5) = 1$	2	0

Row	S_1	S_2	S_3	S_4	$x+1 \mod 5$	$3x+1 \mod 5$
0	1	0	0	1	1	1
1	0	0	1	0	2	4
2	0	1	0	1	3	2
3	1	0	1	1	4	0
4	0	0	1	0	0	3

	S_1	S_2	S_3	S_4
h_1	∞	∞	∞	∞
h_2	∞	∞	∞	∞

	S_1	S_2	S_3	S_4
h_1	1	∞	∞	1
h_2	1	∞	∞	1

Row	S_1	S_2	S_3	S_4	$x+1 \mod 5$	$3x+1 \mod 5$
0	1	0	0	1	1	1
1	0	0	1	0	2	4
2	0	1	0	1	3	2
3	1	0	1	1	4	0
4	0	0	1	0	0	3

	S_1	S_2	S_3	S_4
h_1	1	∞	∞	1
h_2	1	∞	∞	1

	S_1	S_2	S_3	S_4
h_1	1	∞	2	1
h_2	1	∞	4	1

Row	S_1	S_2	S_3	S_4	$x+1 \mod 5$	$3x+1 \mod 5$
0	1	0	0	1	1	1
1	0	0	1	0	2	4
2	0	1	0	1	3	2
3	1	0	1	1	4	0
4	0	0	1	0	0	3

	S_1	S_2	S_3	S_4
h_1	1	∞	2	1
h_2	1	∞	4	1

	S_1	S_2	S_3	S_4
h_1	1	3	2	1
h_2	1	2	4	1

Row	S_1	S_2	S_3	S_4	$x+1 \mod 5$	$3x+1 \mod 5$
0	1	0	0	1	1	1
1	0	0	1	0	2	4
2	0	1	0	1	3	2
3	1	0	1	1	4	0
4	0	0	1	0	0	3

	S_1	S_2	S_3	S_4
h_1	1	3	2	1
h_2	1	2	4	1

	S_1	S_2	S_3	S_4
h_1	1	3	2	1
h_2	0	2	0	0

Row	S_1	S_2	S_3	S_4	$x+1 \mod 5$	$3x+1 \mod 5$
0	1	0	0	1	1	1
1	0	0	1	0	2	4
2	0	1	0	1	3	2
3	1	0	1	1	4	0
4	0	0	1	0	0	3

	S_1	S_2	S_3	S_4
h_1	1	3	2	1
h_2	0	2	0	0

	S_1	S_2	S_3	S_4
h_1	1	3	0	1
h_2	0	2	0	0

Row	S_1	S_2	S_3	S_4
0	1	0	0	1
1	0	0	1	0
2	0	1	0	1
3	1	0	1	1
4	0	0	1	0

SIM(S_1,S_4)=2/3

	S_1	S_2	S_3	S_4
h_1	1	3	0	1
h_2	0	2	0	0

SIM(S_1,S_4)=1

Implementação (4)

- Em geral, os dados são fornecidos em colunas e não em linhas
  - Ex., colunas = documentos, linhas = shingles.
- Se este for o caso, a matriz pode ser ordenada por linha
- Compute $h_i(r)$ será computada somente uma vez para cada linha

Location-Sensitive Hashing (LSH)

- Mesmo que todas as assinaturas ou objetos possam ser colocados na memória, a comparação todos os pares de assinaturas ou objetos é quadrático no número de colunas.
- Ex: 10^6 colunas ➔ $5*10^{11}$ comparações
- Considerando que cada comparação toma 1 micro-segundo, a operação levaria 6 dias.

LSH

- Idéia Geral: Usar uma função f(x,y) que diz se *x* e *y* formam um par candidato, ou seja, um par de elementos que deve ter sua similaridade verificada
- Para matrizes minhash: Armazenar as colunas em um hashing. As colunas que caem em um mesmo elemento formam pares candidatos.

Geração de Candidatos

- Determinar um limiar de similaridade
  - $0 < s < 1$
- Um par de colunas C,D é um *par candidato* se as suas assinaturas convergem em pelo menos um fração *s* das linhas
- Ou seja, $M(i, C) = M(i, D)$ para pelo menos uma fração s dos valores de *i*.

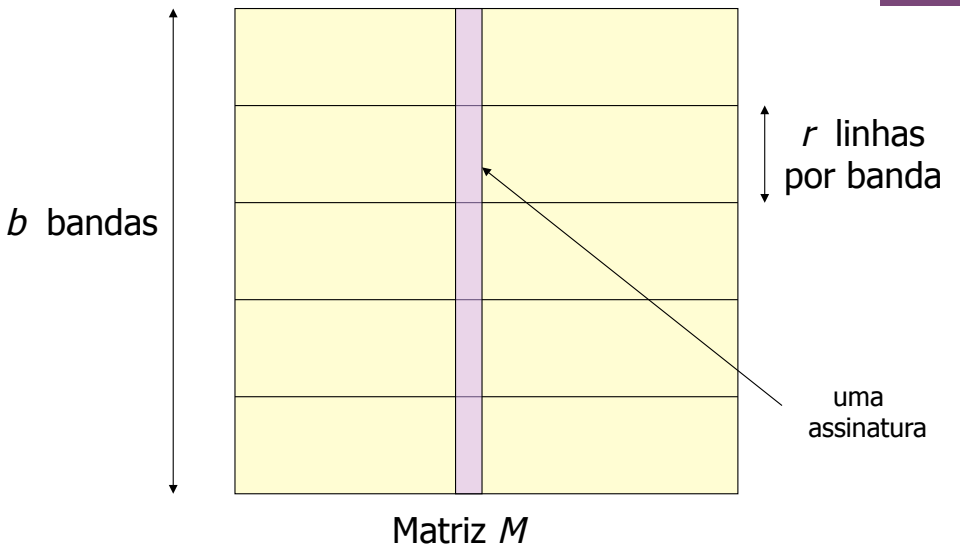
LSH para Assinaturas Minhashing

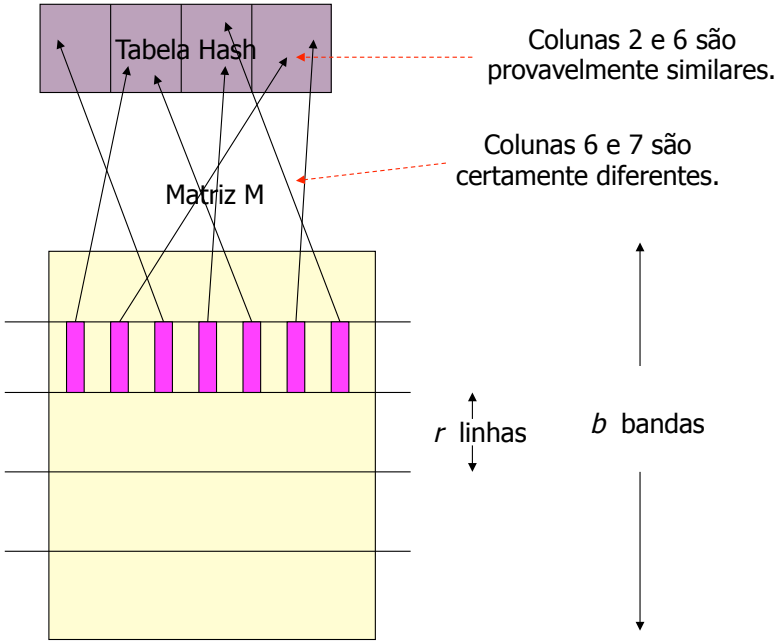
- Idea geral: gerar vários valores de hash para as colunas da matriz de assinaturas M
- Deve-se garantir que apenas colunas similares sejam armazenadas no mesmo elemento
- Pares candidatos são aqueles que ficam pelo menos uma vez no mesmo elemento.

Partição em Bandas

- Dividir a matriz *M* em *b* bandas de *r* linhas.
- Para cada banda, mapear sua parte em M de cada coluna para uma tabela hash de *k* elementos.
  - k é tão grande quanto possível
- *Pares Candidatos* de colunas são aqueles mapeados para o mesmo elemento em uma ou mais bandas.
- Ajustar *b* e *r* para obter a maioria dos pares similares e poucos pares não similares.

Partição em Bandas (2)





Partição em Bandas (3)

- Como a elementos suficientes, as colunas provavelmente não serão mapeadas para o mesmo elemento a não ser que elas sejam idênticas em uma banda em particular.
- Assim, assume-se se duas colunas estão mapeadas para o mesmo elemento, então elas são idênticas na banda correspondente.

Exemplo

- Suponha 100.000 colunas
- As assinaturas usam 100 inteiros
  - 4x100.00x100 =40x1milhão = 40Mb.
- Queremos encontrar todos os pares com 80% de similaridade
- Existirão 5.000.000.000 de pares de assinaturas para comparar.
- Vamos usar 20 de 5 inteiros por banda

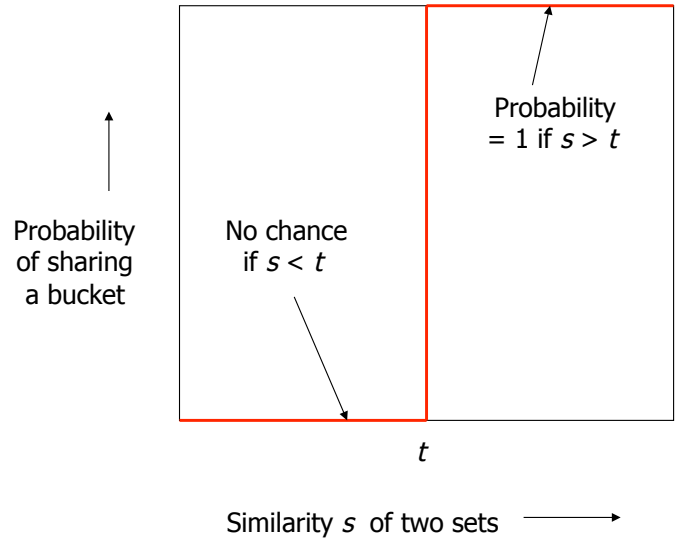
Exemplo

- Suponha C1 e C2 com similaridade de 80%
- Probabilidade de C1 e C2 serem idênticas em uma banda em particular é $(0.8)^5 = 0.328$.
- Probabilidade de C1, C2 não serem similares em nenhuma das 20 bandas é: $(1-0.328)^{20} = .00035$
  - ou sejam, cerca 1/3000 das colunas com similaridade de 80% são falso negativos

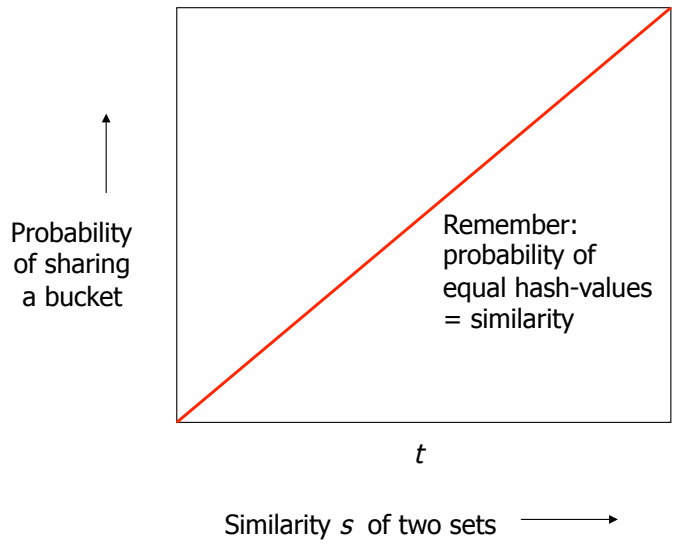
LSH - Tradeoff

- Número de minhashes, número de bandas, número de linhas por banda determinação as taxas de falso positivos/negativos
- Ex: se tivéssemos apenas 15 bandas de 5 linhas, o número de falso positivos iria cair, mas o número de falso negativos iria subir

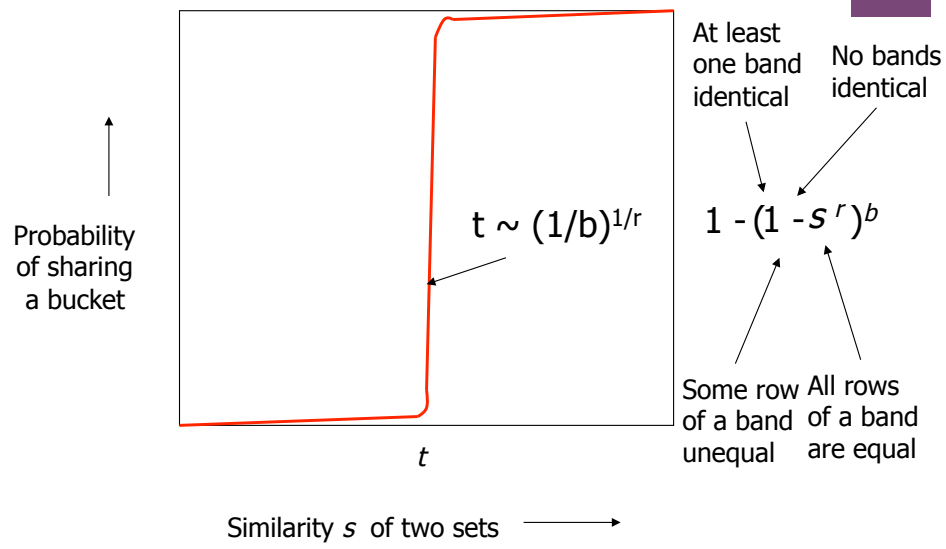
Analysis of LSH – What We Want



What One Band of One Row Gives You



What b Bands of r Rows Gives You



Example: $b = 20; r = 5$

s	$1-(1-s^r)^b$
.2	.006
.3	.047
.4	.186
.5	.470
.6	.802
.7	.975
.8	.9996

LSH Summary

- Tune to get almost all pairs with similar signatures, but eliminate most pairs that do not have similar signatures.
- Check in main memory that candidate pairs really do have similar signatures.
- Optional: In another pass through data, check that the remaining candidate pairs really represent similar *sets* .