
OTUS
ОНЛАЙН-ОБРАЗОВАНИЕ

Алгоритмы MinHash и SimHash

Михаил Горшков
разработчик

Поиск похожих данных: bag of words, shingling, алгоритмы MinHash и SimHash

Проблематика поиска похожих данных

Одна из основных задач, которые решает дата-майнинг, - это поиск похожих объектов. Например, имеется набор веб-страниц, и среди них нужно найти похожие.

- поиск плагиата;
- поиск зеркал сайтов;
- статьи из одного и того же источника;
- интернет-магазины - нахождение "похожих" покупателей по их покупкам. Зачем? Чтобы сформировать "портрет" типичного покупателя: домохозяйки, фотолюбители и проч. и рекомендовать им товары в зависимости от их профиля.

Поиск точных совпадений

Как бы вы решали задачу поиска точных совпадений?

- сортированный массив объектов;
- сортированный массив хэшей объектов + хэш-таблица. Можно посчитать хэш от каждого объекта, отсортировать хэши и найти все одинаковые. Далее сравнить объекты, соответствующие одинаковым хэшам.

В любом случае нужно хранить целевые данные - нужно много места - *O(N)* + сортировка.

Поиск похожих объектов

Если ищем не точные совпадения, а похожие объекты, просто посчитать хэши не получится. Небольшое изменение объекта -> кардинальное изменение хэша.

Вопрос к аудитории: что можно сделать?

Техники нахождения похожих объектов

- **bag of words** – сравнение множества слов в одном документе с множеством слов в другом;
- **shingling** – улучшает "bag of words" за счет сравнения коротких фраз, обеспечивая у слов контекст;
- **hashing** – улучшает процесс за счет отсутствия необходимости хранить копии контента. Фразы хэшируются, полученные хэши затем можно сравнивать для нахождения дубликатов;
- **MinHash** – улучшает процесс хранения хэшей контента;
- **SimHash** – еще улучшает процесс хранения хэшей и детекта дубликатов.

Bag of words

Bag of words - это просто множество слов документа. Например, предложение

"*a bump on the log in the hole in the bottom of the sea*"

будет представлено как множество

{a, in, of, on, the, log, sea, bump, hole, bottom}.

Для определения похожести предложений просто сравним множества их слов.

Как сравнивать два множества?

Нужна мера, которая определяет сходство между множествами.

Желательно определить функцию из множеств S S, T T, U U, ... в число.

Ваши предложения?

Похожесть по Жаккару (Jaccard)

Рассмотрим понятие "похожесть по Жаккару", которая работает с множествами.

Похожесть по Жаккару применяется для 2 или нескольких множеств.

Обозначим через

$$|S|$$

$$|S|$$

мощность множества S S (для конечных множеств это число элементов).

Для множеств S S и T T мера похожести по Жаккару это

$$\frac{|S \cap T|}{|S \cup T|}$$

$$\frac{|S \cap T|}{|S \cup T|}$$

то есть, отношение размера пересечения множеств S S и T T к их объединению. Будем обозначать похожесть по Жаккару множеств S S и T T через $SIM(S,T)$ $SIM(S, T)$.

$SIM(S,T) = SIM(T,S)$ $SIM(S, T) = SIM(T, S)$;

$SIM(S,S) = 1$ $SIM(S, S) = 1$;

$SIM(S,\varnothing) = 0$ $SIM(S, \varnothing) = 0$;

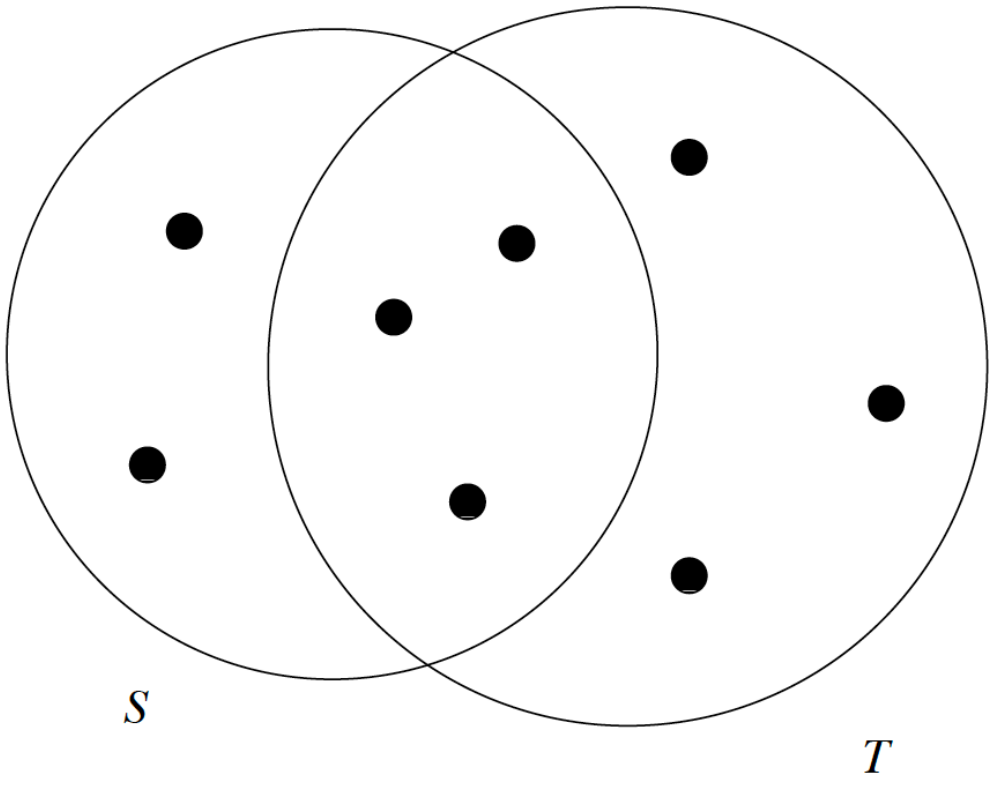
$SIM(\varnothing,\varnothing) = ???$ $SIM(\varnothing, \varnothing) = ???$.

Чем больше мера Жаккара, тем больше похожи множества.

Использование похожести по Жаккару для сравнения по методу Bag of words

Посчитаем меру Жаккара для двух множеств на примерах.

Пример 1



Два множества S и T. Пересечение содержит 3 элемента, объединение 8 элементов. Таким образом, $SIM(S,T) = \frac{3}{8}$ $SIM(S, T) = \frac{3}{8}$.

Упражнение 1

Найдите меру похожести по Жаккару у множеств *{a, a, a, b}* и *{a, a, b, b, c}* (например, это могут быть две корзины покупателей интернет-магазина). В данном случае повторяющиеся элементы возможны.

- пересечение = ?
- объединение = ?
- похожесть по Жаккару = ?

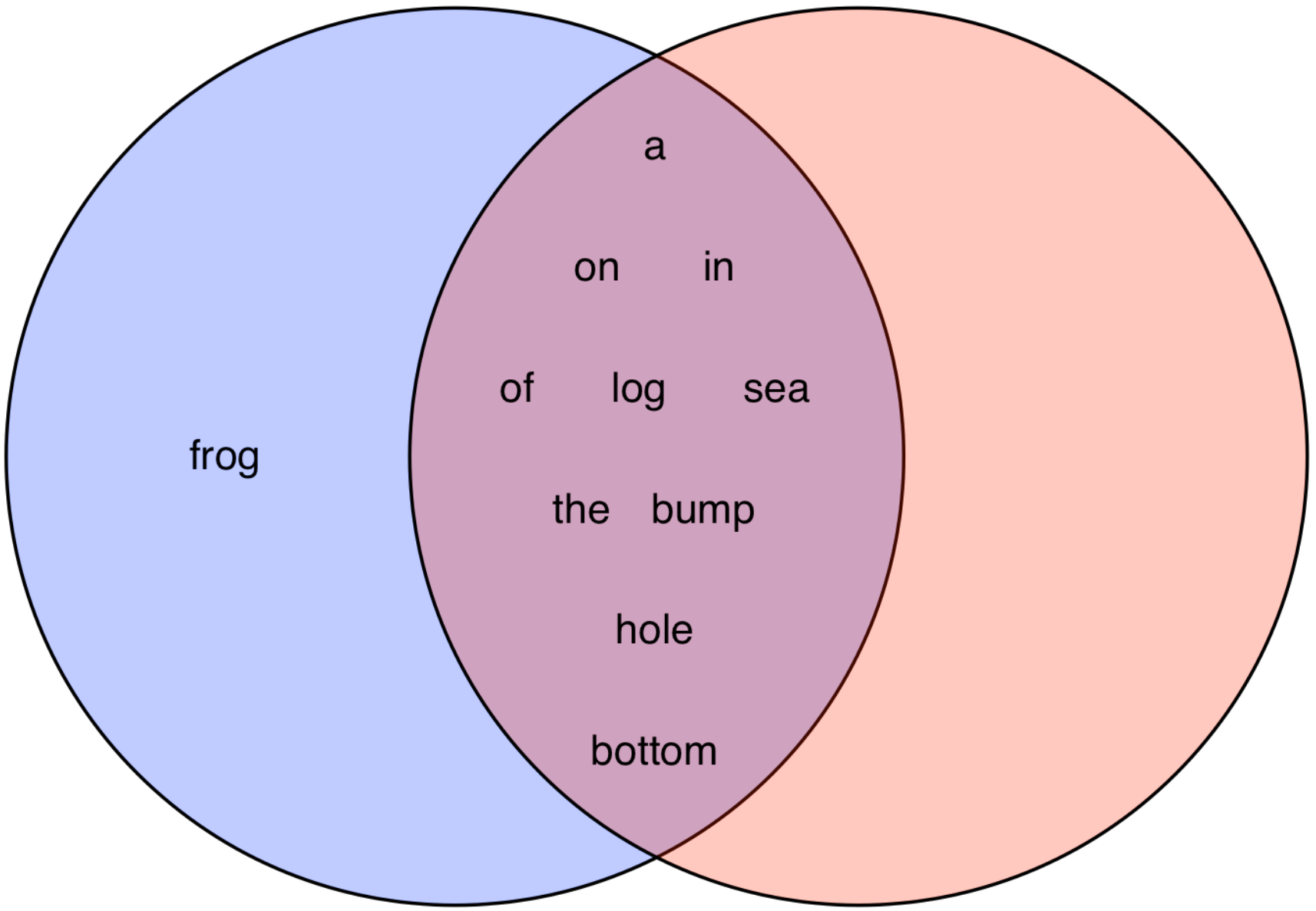
Упражнение 2

Сравним по Жаккару предложение

"*a bump on the log in the hole in the bottom of the sea*"

с предложением

"*a frog on the bump on the log in the hole in the bottom of the sea*"



Чему равно $SIMSIM$?

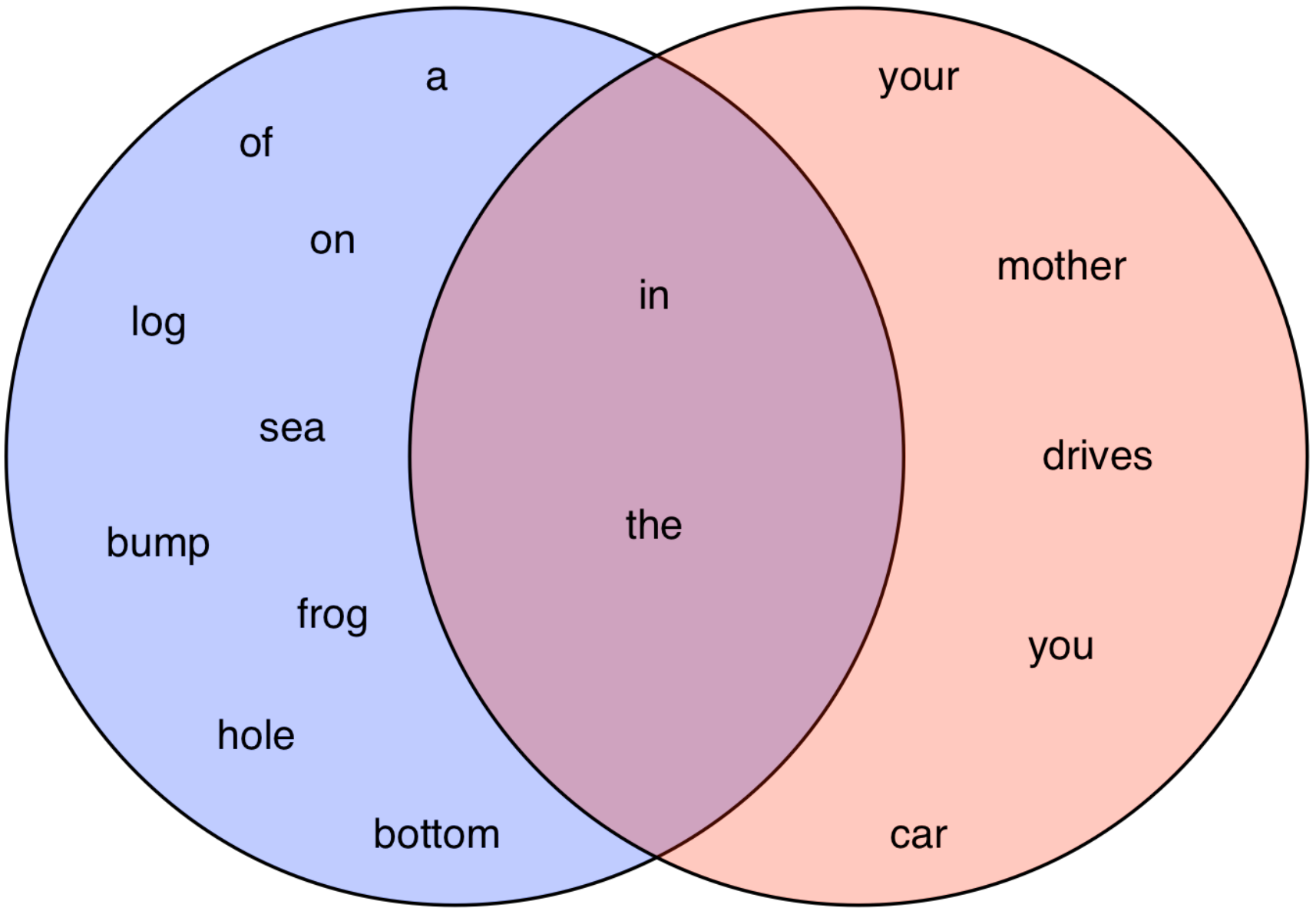
Упражнение 3

Сравним по Жаккару предложение

"*a bump on the log in the hole in the bottom of the sea*"

с предложением

"*your mother drives you in the car*"



SIM_{SIM} = ?

Проблема

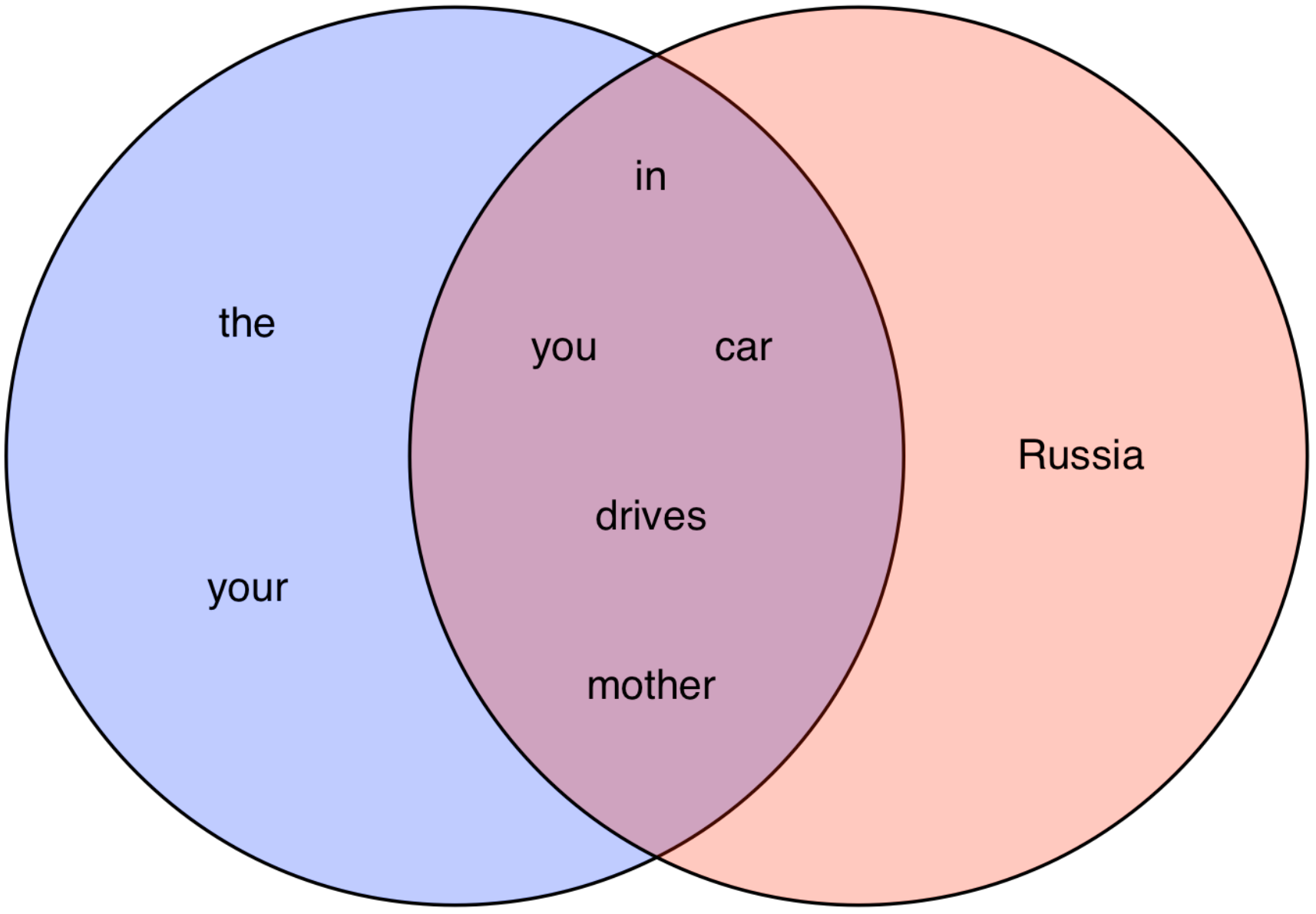
С bag of words есть проблема: какая? (вопрос к аудитории)

Сравним по Жаккару предложение

"*your mother drives you in the car*"

с предложением

"*In mother Russia, car drives you!*"



Пересечение содержит 5 элементов, объединение 8 элементов. Таким образом, $SIM = \frac{5}{8} SIM = \frac{5}{8}$, хотя предложения с точки зрения человека полностью различны.

Нужен лучший подход!

Вопрос к аудитории: что можно сделать?

Шинглинг (shingling) документа

Нужен контекст, в котором находится фраза. Для контекста берем соседние элементы-слова.

A shingle - "галька", "место, покрытое галькой", "короткая женская стрижка", "кусочек материала для покрытия крыши".

To shingle - "выкладывать так, чтобы кусочки перекрывались", "покрывать крышу шинглами".

Как работает шинглинг?

- из документа конструируется набор коротких строк, которые в нем содержатся;
- если документы имеют общие фразы или предложения, то в их наборах коротких строк будет много общих элементов;
- определим kk-шингл для документа как любую подстроку длиной kk, находящуюся в документе;
- сопоставим каждому документу его набор kk-шинглов.

Пример 1

Документ DD - строка $abcdabdabcdabd$, берем $k = 2k = 2$. Тогда набор 2-шинглов для D будет *{ab, bc, cd, da, bd}*.

Упражнение

"*your mother drives you in the car*", $k = 3k = 3$.

Представьте в виде набора шинглов!

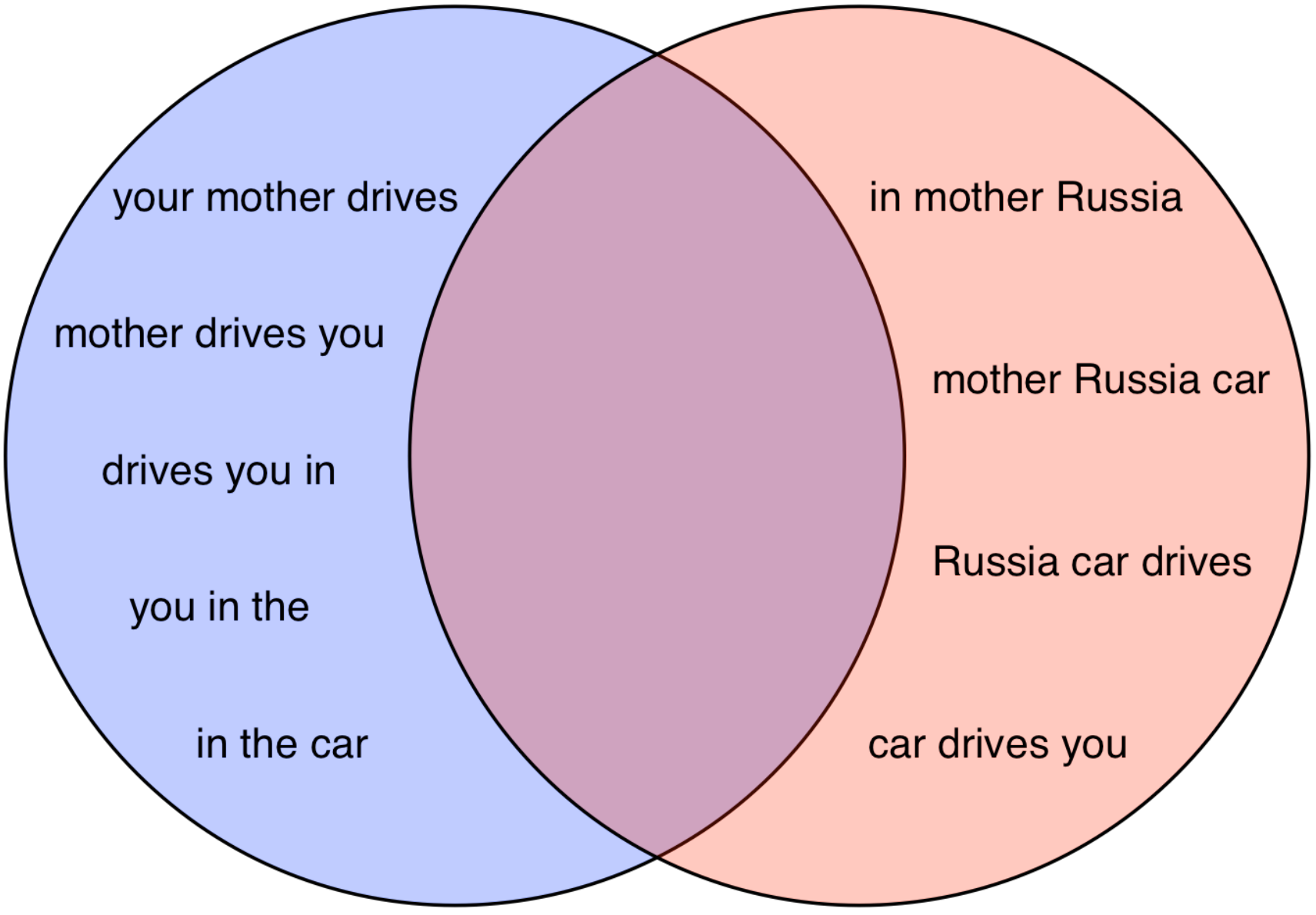
Похожесть по Жаккару наборов шинглов

Еще раз сравним предложение

"*your mother drives you in the car*"

с предложением

"*in mother Russia, car drives you!*"



$$SIM = 0 SIM = 0$$

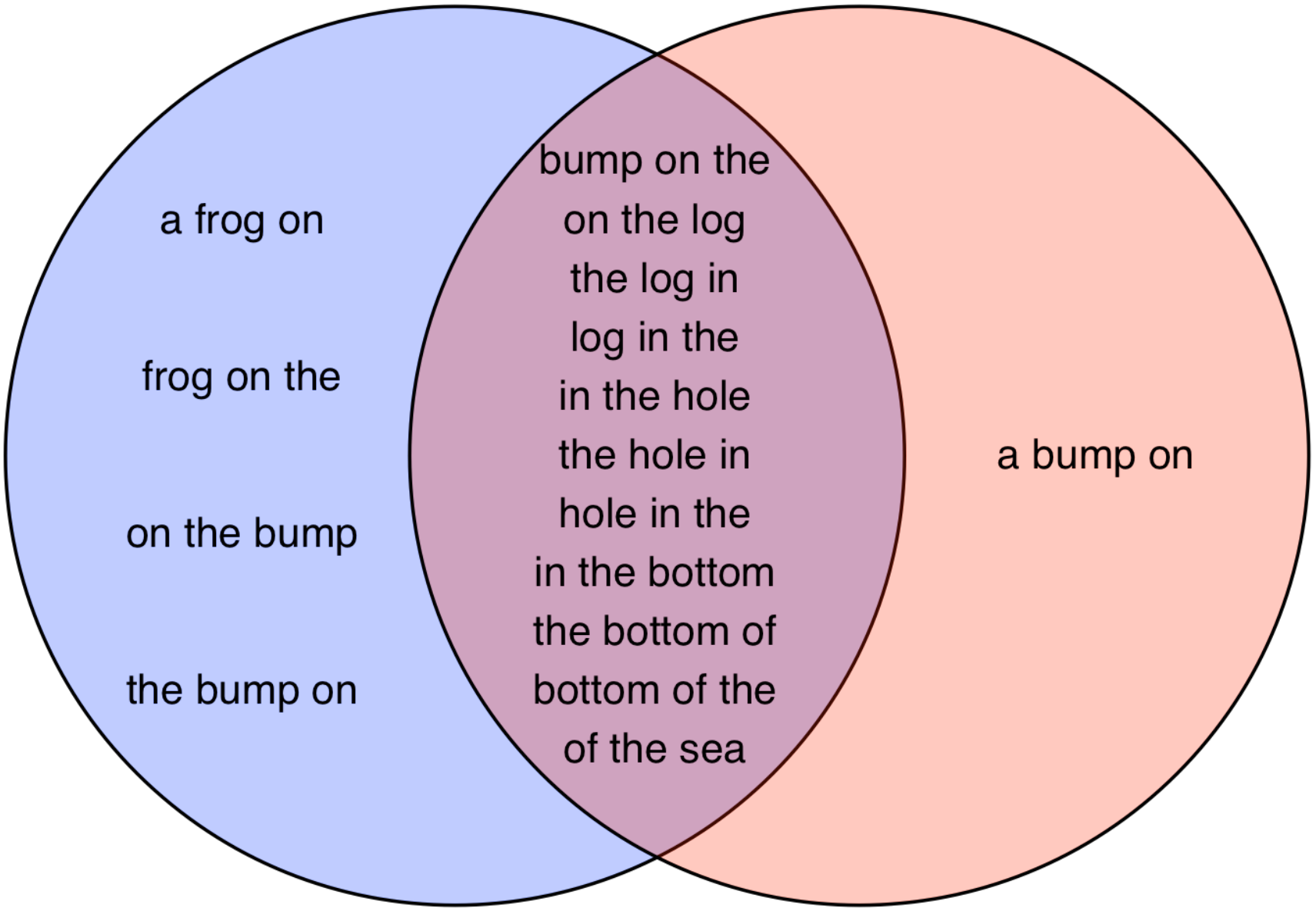
Вернемся к примеру с лягушкой.

Сравним предложение

"*a bump on the log in the hole in the bottom of the sea*"

с предложением

"*a frog on the bump on the log in the hole in the bottom of the sea*"



$SIM = \frac{11}{16} SIM = \frac{11}{16}$

Выбор размера шингла

- если выбрать k слишком маленьким, во многих документах будут одни и те же шинглы - точность определения похожести снизится;
- если выбрать k слишком большим, не найдутся похожие документы, так как набор шинглов будет отличаться даже для похожих документов;
- **k следует выбрать настолько большим, чтобы вероятность появления любого заданного шингла в любом документе была небольшой**.

Грубо: если у нас корпус е-майлов, выбор k = 5 достаточен.

Почему так? Пусть имеем 20 наиболее употребимых символов в е-майл. Имеем $20^5 \approx$ 3 млн возможных шинглов. Так как обычно е-майл гораздо короче 3 млн симв, k = 5 подходит.

Для обычных текстов можно брать k = 9.

Проблема: шинглы занимают слишком много места

Можно хранить наборы шинглов для документов и сравнивать их между собой, но это занимает много места.

Что можно предложить - вопрос к аудитории?

Хэширование шинглов

- не будем использовать сами шинглы, а выберем хэш-функцию, которая мапит шинглы длины k в число - номер корзины;
- результат будем интерпретировать как шингл;
- набор шинглов, представляющий документ, - это набор чисел - номеров корзин;

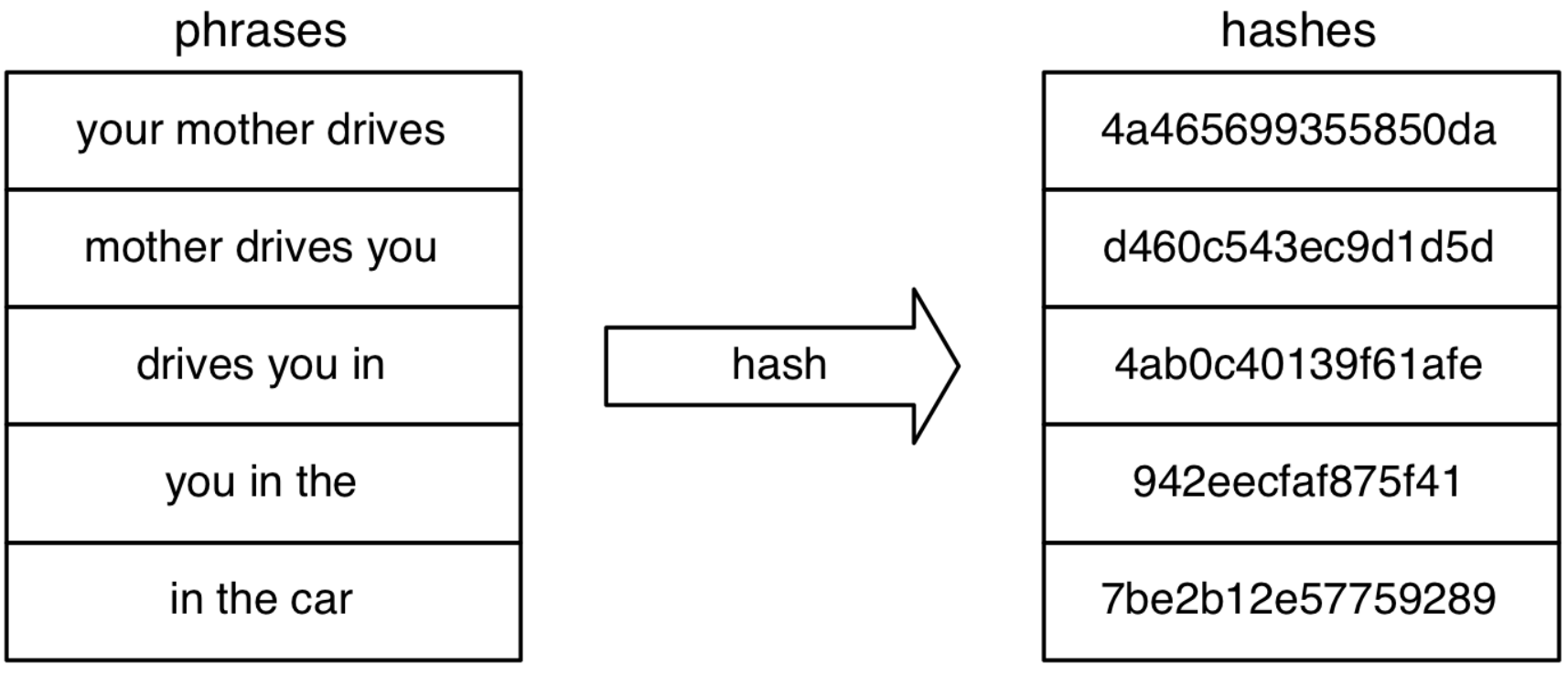
Например, делаем 9-шинглы по документу (посимвольно) и хэшируем их в диапазон 0 - 2^{32} - каждый шингл умещается в 4 байта вместо 9.

Бонусы:

- данные сжимаются;
- после хэширования значения распределяются более равномерно;
- можем манипулировать хэшированными шинглами как числами, что удобно.

Пример

"*your mother drives you in the car*"



Проблема: даже после хэширования большой размер

Хэши могут занять значительно больше места, чем занимал исходный документ.

Если документов миллионы -> хэши не поместятся в оперативную память -> трудно работать с ними. Даже если поместятся, очень ресурсоемко будет сравнивать все пары документов, вычисляя для каждой пары меру Жаккара.

Идеально:

- для всего документа иметь одну "подпись", а не множество хэшей;
- хорошо бы, чтобы мера похожести по Жаккару между двумя документами легко вычислялась по двум "подписям" документов.

Цель: найти способ задать короткую "подпись" для документа, при помощи которой можно было бы вычислить расстояние Жаккара

Есть ли предложения как это сделать?

Матричное представление множества шинглов

Визуализируем множество шинглов с помощью характеристической матрицы.

- матрица представляет собой визуализацию множества шинглов для одного документа;
- столбцы матрицы соответствуют множествам шинглов, строки соответствуют элементам "универсального множества", откуда взяты элементы наших множеств;
- универсальное множество - общее для всех документов;
- если элемент, соответствующий строке r является членом множества для колонки c, в соответствующей ячейке матрицы стоит 1. Иначе 0;
- матрицы часто разреженные **(почему? вопрос к аудитории)**;
- это только для визуализации, в реальности не будем хранить так данные.

Пример

Универсальное множество: *{a, b, c, d, e}*.

S_1 = *{a, d}*;

S_2 = *{c}*;

S_3 = *{b, d, e}*;

S_4 = *{a, c, d}*.

Элемент/Множество	S_1	S_2	S_3	S_4
a	1	0	0	1
b	0	0	1	0
c	0	1	0	1
d	1	0	1	1
e	0	0	1	0

Подписи для множества шинглов

Разработаем идею "подписей" множества шинглов.

- множества шинглов большие: шингл может быть 16 байт, а хэш - 4 байта;
- заменим большие множества шинглов на небольшие подписи.

Бонусы:

- можем посчитать подписи у документов и просто сравнить их для оценки похожести по Жаккару.

Введем понятие ***minhash*** характеристической матрицы - это и будет подписью множества шинглов документа.

MinHash

Чтобы посчитать **minhash** множества, представленного любой колонкой характеристической матрицы (то есть любого из множеств S_i S_i), выберем какую-либо пермутацию (случайную перестановку) строк этой матрицы.

MinHash - *это функция отображения множества в число*.

MinHash любой колонки - это первый по счету элемент, в перемешанном порядке, в которой колонка содержит единицу.

Пример

Вернемся к предыдущей матрице

Элемент/Множество	S_1 s_1	S_2 s_2	S_3 s_3	S_4 s_4
a	1	0	0	1
b	0	0	1	0
c	0	1	0	1
d	1	0	1	1
e	0	0	1	0

Пусть мы выбрали порядок строк $beadc$ $beadc$ для неё:

Элемент/Множество	S_1 s_1	S_2 s_2	S_3 s_3	S_4 s_4
b	0	0	1	0
e	0	0	1	0
a	1	0	0	1
d	1	0	1	1
c	0	1	0	1

Эта перестановка определяет **minhash**-функцию h, которая мапит наборы в строки.

Вычислим значение **minhash** множества S_1 S_1 в соответствии с h.

- первая колонка для множества S_1 S_1, имеет 0 в строке b, поэтому переходим к строке e;
- там также 0 в колонке для S_1 S_1;
- переходим к строке a, там 1.

Таким образом, $h(S_1)$ $h(S_1) = a$.

Аналогично, опускаемся вниз по остальным колонкам, пока не дойдем до 1. Получаем:

$h(S_2)$ $h(S_2) = c$;

$h(S_3)$ $h(S_3) = b$;

$h(S_4)$ $h(S_4) = a$.

Понятно ли это?

Minhash и похожесть по Жаккару

Вероятность того, что функция minhash для случайной перестановки строк выдает одно и то же значение для двух множеств, равна мере похожести по Жаккару между этими множествами.

$$p(h(S_i) = h(S_j)) = SIM(S_i, S_j)$$

$$p(h(S_i) = h(S_j)) = SIM(S_i, S_j)$$

Интуитивно это понятно (напр. на диаграмме $S_1 S_1$ и $S_4 S_4$). Докажем, почему это так в общем случае.

Нужно рассмотреть столбцы для двух множеств.

Рассмотрим столбцы для множеств $S_i S_i$ и $S_j S_j$, тогда строки можно разделить на 3 категории:

- строки типа XX имеют 1 в обоих столбцах;
- строки типа YY имеют 1 в одном из столбцов и 0 в другом;
- строки типа ZZ имеют 0 в обоих столбцах.

Так как матрица разреженная, большинство строк имеет тип ZZ. Однако, и $SIM(S_i, S_j) SIM(S_i, S_j)$, и вероятность того, что $h(S_i) = h(S_j) h(S_i) = h(S_j)$ определяется соотношением числа строк типов X X и Y Y.

Пусть имеется xx строк типа XX и yy строк типа YY.

Тогда

$$SIM(S_i, S_j) = \frac{x}{x + y}$$

$$SIM(S_i, S_j) = \frac{x}{x + y}$$

потому что xx это размер $S_i \cap S_j S_i \cap S_j$, а $x + yx + y$ это размер $S_i \cup S_j S_i \cup S_j$.

Рассмотрим вероятность того, что $h(S_i) = h(S_j) h(S_i) = h(S_j)$.

Если строки были перемешаны случайно, и мы начинаем с начала, вероятность, что мы встретим строку типа XX до строки типа YY равна

$$\frac{x}{x + y}$$

$$\frac{x}{x + y}$$

Но если первая строка сверху, отличная от ZZ, - это строка типа XX, то $h(S_i) = h(S_j) h(S_i) = h(S_j)$.

Почему? Вопрос к аудитории!

С другой стороны, если первая строка не типа ZZ имеет тип YY, то множество с 11 получает эту строку как его значение **minhash**.

Однако множество с 00 в этой строке точно получит некоторую строку далее в перемешанном списке.

Таким образом, мы знаем, что $h(S_i) \neq h(S_j) h(S_i) \neq h(S_j)$, если мы первой встретим строку типа YY.

Делаем вывод, что вероятность того, что

$$p(h(S_i) = h(S_j)) = \frac{x}{x+y}$$

$$p(h(S_i) = h(S_j)) = \frac{x}{x+y}$$

что одновременно является **мерой похожести по Жаккару между S_i S_i и S_j S_j**.

Получили результат для одной случайно перестановки. Но этого недостаточно, нужно сделать много перестановок, чтобы погрешность была меньше.

Подписи MinHash

Вернемся к набору множеств, представленных характеристической матрицей M M. Чтобы представить множества, мы выбираем случайно некоторое число n n случайных перестановок строк M M, то есть сделаем рандомное число пермутаций.

Назовем **minhash**-функции, определяемые этими пермутациями через h_1 h_1, h_2 h_2, . . . , h_n h_n. Из колонки, представляющей множество S S, сконструируем **minhash**-подпись для S S, а именно вектор $[h_1(S), h_2(S), \ldots, h_n(S)]$ $[h_1(S), h_2(S), \ldots, h_n(S)]$.

Обычно мы представляем этот список hash-значений как колонку. Таким образом, мы можем сформировать матрицу подписей M M, в которой i i-я колонка M M заменяется **minhash**-подписью для множества i i-й колонки.

Заметим, что матрица подписей имеет такое же число колонок, как M M, но только n n строк.

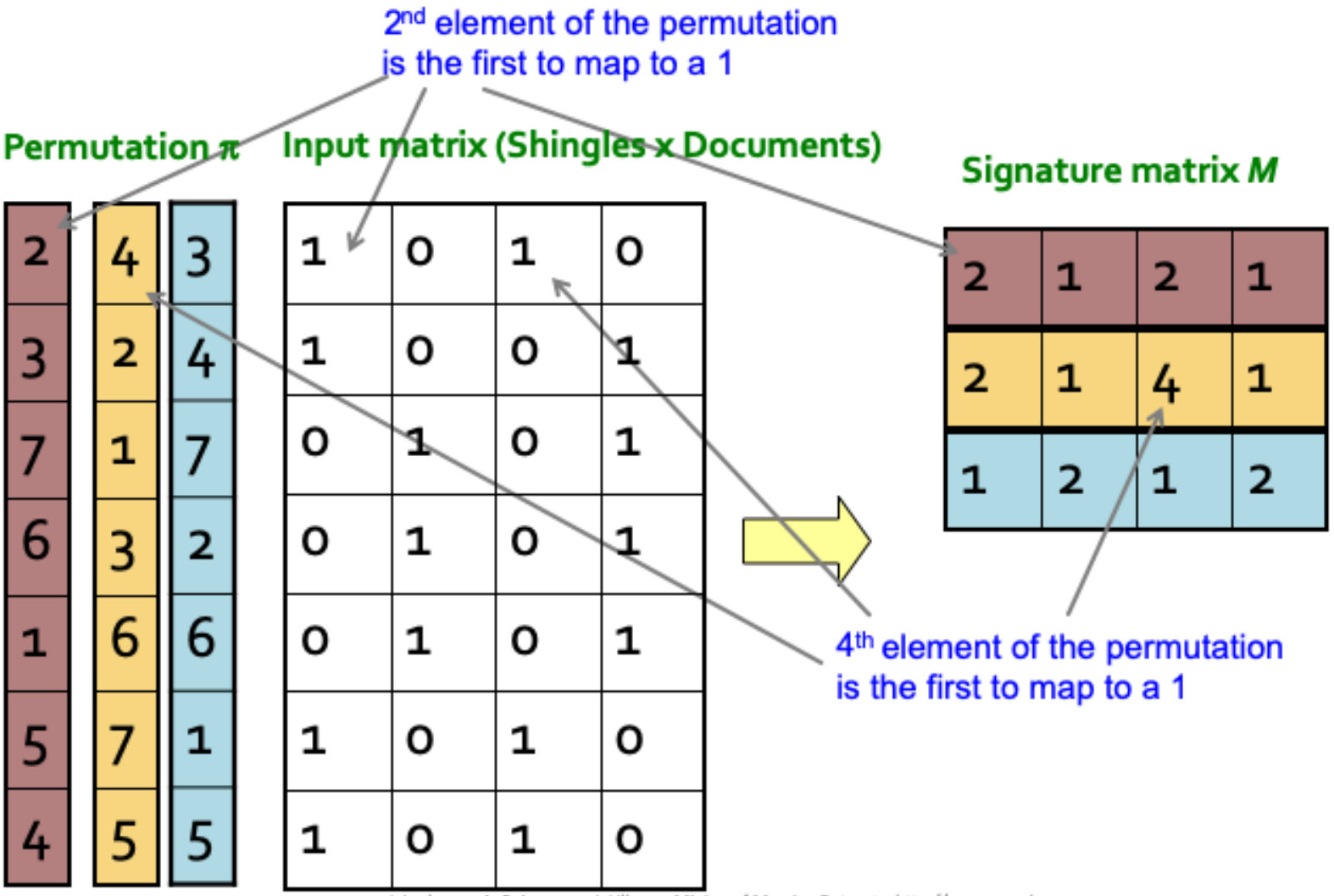
Несмотря на то, что M M можно представить не явно, а в сжатом виде (только с единицами как минимум в одной колонке), матрица подписей может быть гораздо меньше M M.

Пример

- входная матрица - не пермутированная;
- альтернатива для hh - хранить не сами значения, а индексы.

Note: Another (equivalent) way is to store row indexes:

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



Похожесть по Жаккару через подписи

Мы знаем, что

$$p(h_k(S_i) = h_k(S_j)) = SIM(S_i, S_j)$$

$$p(h_k(S_i) = h_k(S_j)) = SIM(S_i, S_j)$$

Обобщим на несколько хэш-функций.

Похожесть двух подписей это доля хэш-функций, в которых они совпадают.

Похожесть колонок такая же, как ожидаемая похожесть их подписей.

Min-Hashing Example

Permutation π

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

Input matrix (Shingles x Documents)

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

Signature matrix *M*

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

Similarities:

	1-3	2-4	1-2	3-4
Col/Col	0.75	0.75	0	0
Sig/Sig	0.67	1.00	0	0

Цель достигнута?

- выберем n=100 случайных пермутаций строк;
- пусть sig(C) - колонка-вектор;
- sig(C)[i] = по i-той пермутации индекс (значение) первой строки, в которой колонка C содержит 1;
- подпись (sketch) документа всего ~**100** байт!

Мы сжали длинные битовые вектора в короткие подписи.

Эффективное вычисление подписей MinHash

Проблема

- делать явные пермутации характеристической матрицы слишком затратно;
- даже выбор случайной пермутации из миллионов или миллиардов элементов ресурсоемкая операция, а требуемая сортировка строк (исходной матрицы в соответствии с пермутацией) займет еще больше ресурсов.

Таким образом, пермутированные матрицы аналогичные тем, что мы рассмотрели ранее, сделать концептуально привлекательно, но малореалистично.

Решение

К счастью, возможно симулировать эффект случайной перестановки с помощью случайной хэш-функции, которая мапит номера строк на такое же к-во корзин. Хэш-функция, которая мапит целые числа $0$0, $1$1, . . . , $k-1$$k-1$ в номера корзин от $0$0 до $k-1$$k-1$, может замапить некоторые пары чисел в одну и ту же корзину, а другие корзины оставит незаполненными. Однако, это не имеет значения, если $k$$k$ большое и таких коллизий не слишком много.

Таким образом, вместо выбора $n$$n$ случайных пермутаций строк, мы случайно выбираем $n$$n$ хэш-функций $h_1$$h_1$, $h_2$$h_2$, . . . , $h_n$$h_n$ над строками.

Вычислим подпись для матрицы, обсчитывая каждую строку следующим образом.

Пусть $SIG(i,c)$$SIG(i,c)$ это элемент матрицы подписей для $i$$i$-той хэш-функции и столбца $c$$c$.

Проинитим $SIG(i,c)$$SIG(i,c)$ в ∞ для всех $i$$i$ и $c$$c$. Обсчитываем строку $r$$r$ следующим образом:

- вычислить $h_1(r)$$h_1(r)$, $h_2(r)$$h_2(r)$, . . . , $h_n(r)$$h_n(r)$;
- для каждой колонки $c$$c$ сделать следующее:
  - если колонка $c$$c$ имеет $0$0 в строке $r$$r$, ничего не делаем.
  - если колонка $c$$c$ имеет $1$1 в строке $r$$r$, то для каждого $i$$i$ = $1$1, $2$2, . . . , $n$$n$ пусть $SIG(i,c)$$SIG(i,c)$ будет наименьшим значением из текущего $SIG(i,c)$$SIG(i,c)$ и $h_i(r)$$h_i(r)$.

Как выбрать случайную хэш-функцию $h(x)$$h(x)$? Универсальное хэширование:

$$h_{a,b}(x) = ((a \cdot x + b)\%P)\%N$$

$$h_{a,b}(x) = ((a \cdot x + b)\%P)\%N$$

где:

- $a$$a$, $b$$b$ - случайные целые числа;
- $P$$P$ - простое число ($P$$P$ > $N$$N$).

Пример вычисления матрицы подписей

Строка	S_1 s_1	S_2 s_2	S_3 s_3	S_4 s_4
a	1	0	0	1
b	0	0	1	0
c	0	1	0	1
d	1	0	1	1
e	0	0	1	0

- заменим значения a, b, c, d, e числами 0 - 4;

Строка	S_1 s_1	S_2 s_2	S_3 s_3	S_4 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

- выберем хэш-функции $h_1(x) = (x + 1)\%5 h_1(x) = (x + 1)\%5$ и $h_2(x) = (3x + 1)\%5$ $h_2(x) = (3x + 1)\%5$
- впишем их результат в новые колонки

Строка	S_1 s_1	S_2 s_2	S_3 s_3	S_4 s_4	h_1 h_1	h_2 h_2
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

- заметим, что эти простые хэш-функции дают настоящие пермутации строк, но это возможно, потому что число строк 5 - простое. В общем случае будут коллизии.

Вычислим матрицу подписей

Изначально матрица состоит из одних ∞

-	S_1 s_1	S_2 s_2	S_3 s_3	S_4 s_4
h_1 h_1	∞	∞	∞	∞
h_2 h_2	∞	∞	∞	∞

Смотрим строку 0 предыдущей матрицы. $h_1(0)h_1(0) = 11$ и $h_2(0)h_2(0) = 11$.

Строка 0 имеет единицы в столбцах для множеств S_1S_1 и S_4S_4, поэтому только эти колонки в матрице подписей могут меняться. Так как 11 меньше ∞, меняем оба значения в колонках для S_1S_1 и S_4S_4.

Текущее состояние матрицы:

-	S_1S_1	S_2S_2	S_3S_3	S_4S_4
h_1h_1	1	∞	∞	1
h_2h_2	1	∞	∞	1

Теперь переходим к строке 1 предыдущей матрицы. Эта строка имеет 1 только в S_3S_3, и ее значения хэша $h_1(1)h_1(1) = 22$ и $h_2(1)h_2(1) = 44$. Таким образом, $SIG(1, 3)SIG(1, 3) = 22$ и $SIG(2, 3)SIG(2, 3) = 44$. Все другие ячейки остаются как есть, так как для них в строке 1 содержится 0.

Новое состояние матрицы:

-	S_1S_1	S_2S_2	S_3S_3	S_4S_4
h_1h_1	1	∞	2	1
h_2h_2	1	∞	4	1

Строка 2 содержит единицы в колонках для S_2S_2 и S_4S_4, и ее значения хэша $h_1(2)h_1(2) = 33$ и $h_2(2)h_2(2) = 22$. Мы можем поменять значения в подписи для S_4S_4, но значения в этой колонке матрицы подписей каждое меньше, чем соответствующее значение хэша. Однако, так как колонка для S_2S_2 содержит бесконечности, мы заменяем ее на (3, 2).

Результат:

-	S_1S_1	S_2S_2	S_3S_3	S_4S_4
h_1h_1	1	3	2	1
h_2h_2	1	2	4	1

Затем рассмотрим строку 3. Все колонки, кроме S_2S_2, содержат 1, и значения хэша $h_1(3)h_1(3) = 44$ и $h_2(3)h_2(3) = 00$. Значение 4 для h_1h_1 уже превышает то, что содержится в матрице подписей для всех колонок, поэтому мы не будем менять значения в первой строке матрицы подписей. Однако, значение 0 для h_2h_2 меньше, чем то, что уже присутствует, так что мы уменьшаем $SIG(2, 1)SIG(2, 1)$, $SIG(2, 3)SIG(2, 3)$ и $SIG(2, 4)SIG(2, 4)$ до 0. Заметим, что мы не можем уменьшить $SIG(2, 2)SIG(2, 2)$, потому что колонка для S_2S_2 содержит 0 в строке, которую мы рассматриваем.

Получается матрица:

-	S_1S_1	S_2S_2	S_3S_3	S_4S_4
h_1h_1	1	3	2	1
h_2h_2	0	2	0	0

Наконец, рассмотрим строку 4. $h_1(4)h_1(4) = 00$ и $h_2(4)h_2(4) = 33$. Так как строка 4 имеет 1 только в колонке для $S_3 S_3$, мы сравниваем текущую колонку матрицы подписей (2, 0) только со значениями хэша (0, 3). Так как 0 < 2, мы меняем $SIG(1, 3)SIG(1, 3)$ на 00, но так как $33 > 00$, мы не меняем $SIG(2, 3)SIG(2, 3)$.

Окончательная матрица подписей:

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

Оценим похожесть по Жаккару

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

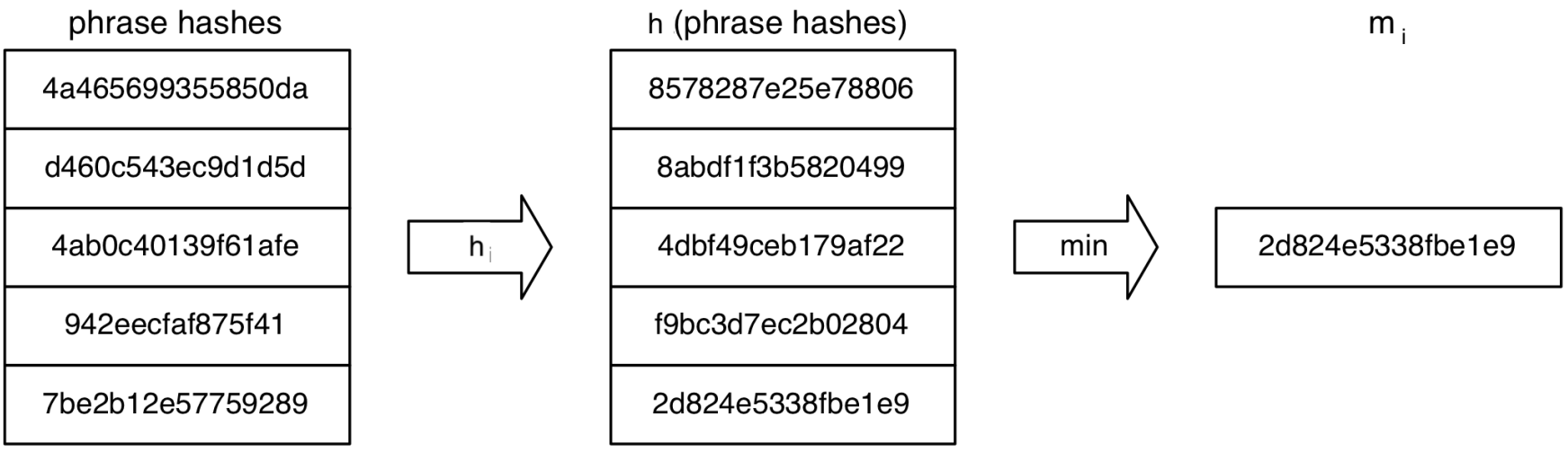
Оценим похожесть по Жаккару между множествами по матрице подписей.

Затемим, что колонки 1 и 4 идентичны, так что мы догадываемся, что $SIM(S_1, S_4)SIM(S_1, S_4) = 1.0$ 1.0. По предыдущей матрице мы видим, что истинная похожесть по Жаккару между $S_1 S_1$ и $S_4 S_4$ равна 2/3. На самом деле по ЗБЧ приближение будет более точным с увеличением размерности.

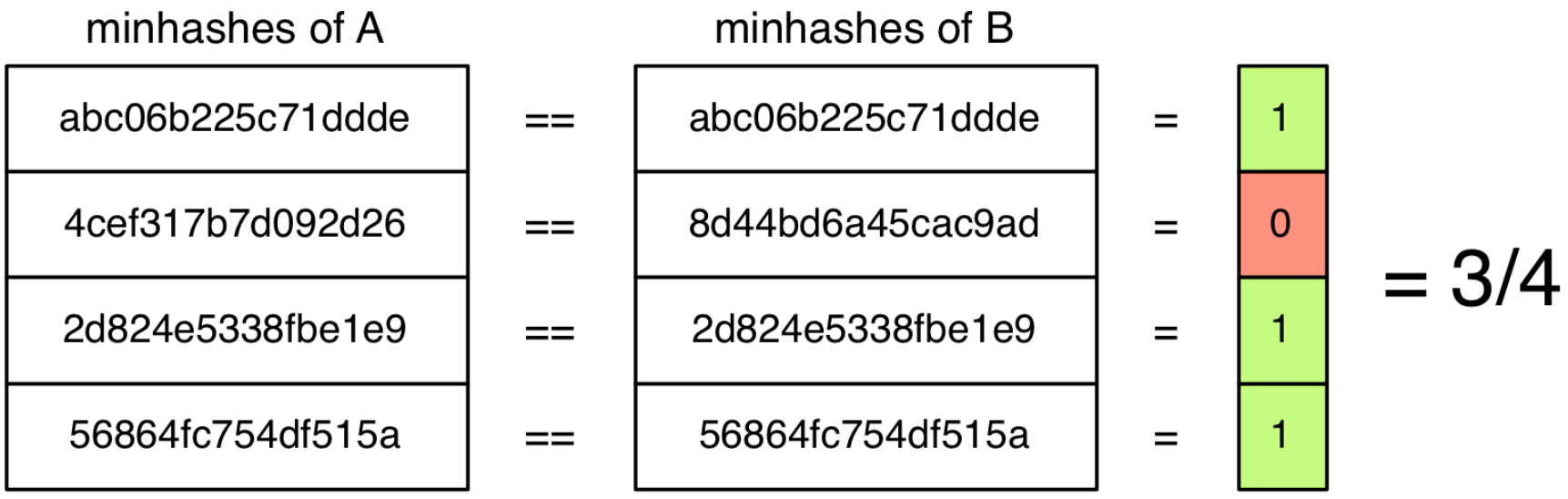
Колонки в подписях $S_1 S_1$ и $S_3 S_3$ совпадают в половине строк (точное расстояние 1/4), колонки $S_1 S_1$ и $S_2 S_2$ отличаются на 100%, похожесть по Жаккару - 0 (это верная оценка).

Для нашего примера выше

Документу соответствует набор хэшей фиксированного размера, вычисленных с помощью nn рандомизирующих хеш-функций $h(i)h(i)$.



Подпись документа - это сортированный список минимальных хэшей $m_0 m_0$... $m_{n-1} m_{n-1}$. Этот метод достигает приближения к похожести по Жаккару попарным сравнением минимальных хешей каждого документа.



Теперь требуемое место для каждого документа $O(1)O(1)$, сложность сравнения двух документов $O(1)O(1)$.

Реализация алгоритма

Рассмотрите дома.

https://github.com/chrisjmccormick/MinHash/blob/master/runMinHashExample.py (https://github.com/chrisjmccormick/MinHash/blob/master/runMinHashExample.py)

Проблема алгоритма MinHash

Проблема сложности сравнения документов устранена, но осталась проблема нахождения всех дубликатов из набора документов. С помощью MinHash мы вынуждены сканировать все документы за время $O(N)O(N)$. Что делать, если у нас сотни миллионов документов?

Алгоритм SimHash

Алгоритм **SimHash** помогает решить эту проблему. По набору входных хэшей (хэшированные шинглы) **SimHash** строит единственный хэш со свойством — два похожих множества входных хэшей дают такие же результирующие хэши. Большинство хэширующих функций обладают свойством, что немного отличающиеся input-ы производят очень разные output-ы, что делает **SimHash** уникальным.

Для каждой битовой позиции мы считаем число входных хешей, в которых этот бит установлен и вычитаем количество входных хешей, в которых этот бит не установлен. После этого каждая позиция с отрицательным счетчиком сбрасывается в 0, а все остальные в 1.

inputs	1	0	1	1	1	0	1	1
	0	0	1	0	1	1	1	0
	0	1	1	0	0	0	1	1
	0	1	0	0	0	0	1	0
	1	1	1	1	0	0	1	1
	1	0	0	1	1	1	0	0
	0	0	0	0	1	0	1	1
counters	-1	-1	1	-1	1	-3	6	1
result	0	0	1	0	1	0	1	1

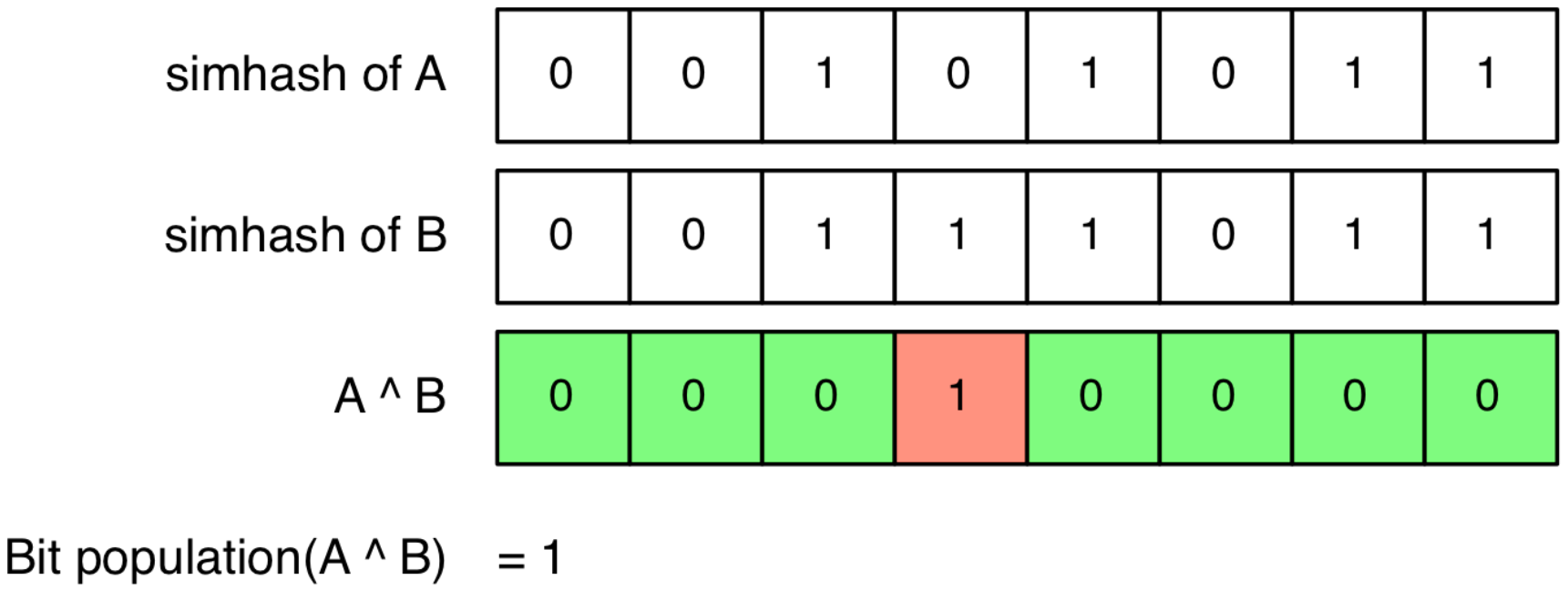
Вычисление расстояния между SimHashes

Чтобы измерить похожесть двух simhashes, мы подсчитываем количество бит, которые отличаются между двумя запросами (**расстояние Хэмминга**). Наоборот, к-во бит, которые одинаковы, это мера похожести.

Как вычислить число бит, в которых значения отличаются? - вопрос к аудитории

Расстояние Хэмминга

bits population(a ^ b).



Выводы

- вместо того, чтобы хранить много хэшей для каждого документа, можно хранить единственное целое число;
- удобнее стало находить набор почти одинаковых документов.

#Сортировка SimHashes

Зачем сортировать документы по их SimHashes?

Несортированные документы

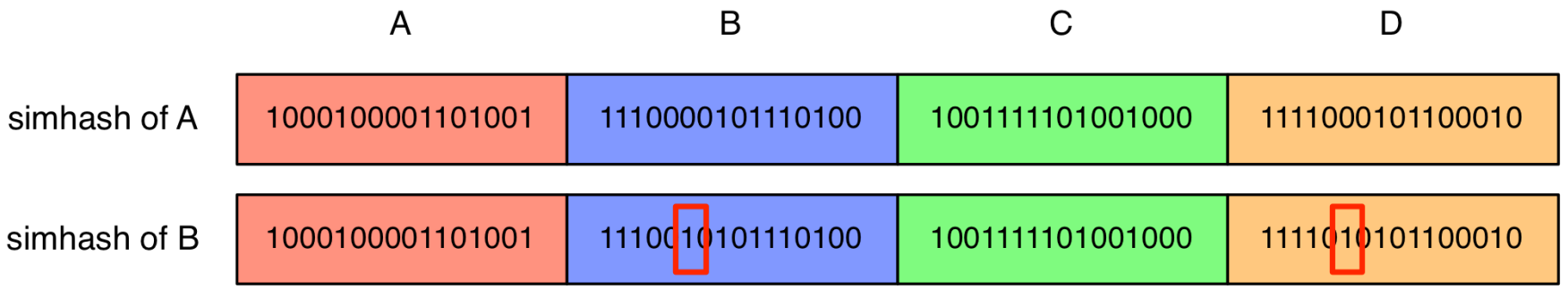
Номер	SimHash	Binary	Hamming from prev
1	37586	1001001011010010	
2	50086	1100001110100110	7
3	2648	0000101001011000	11
4	934	0000001110100110	9
5	40957	1001111111111101	9
6	2650	0000101001011010	9
7	64475	1111101111011011	7
8	40955	1001111111111011	4

Если отсортировать документы по SimHashes, то легко находятся документы с минимальным расстоянием по Хэммингу

Номер	SimHash	Binary	Hamming from prev
4	934	0000001110100110	
3	**2648**	**0000101001011000**	**9**
6	**2650**	**0000101001011010**	**1**
1	37586	1001001011010010	5
8	40955	1001111111111011	6
5	40957	1001111111111101	2
2	50086	1100001110100110	9
7	64475	1111101111011011	9

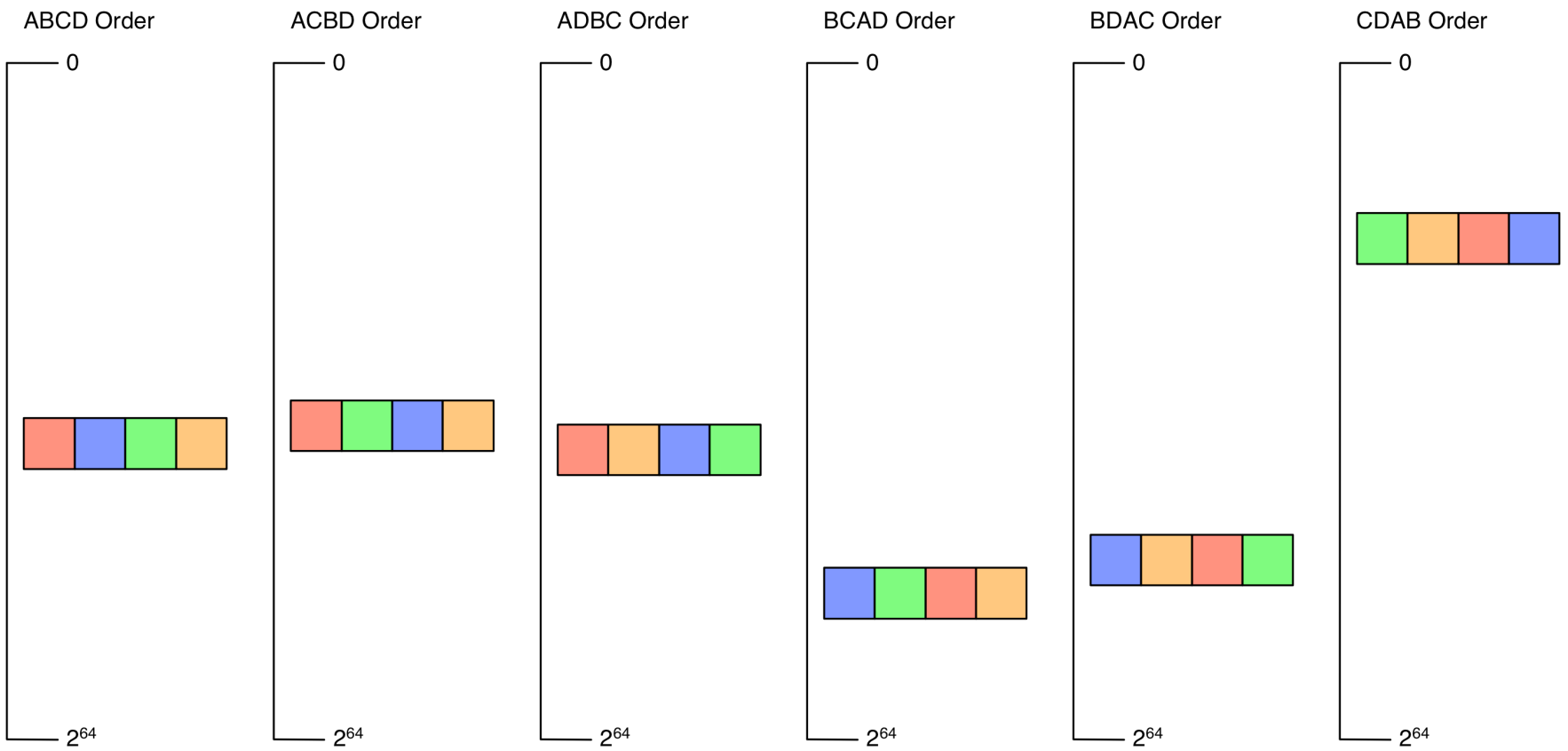
Нахождение набора почти одинаковых документов

Например, предположим, что чтобы документы считались дубликатами, они должны отличаться максимум двумя битами. Разделим наш 64-битный хэш на 4 части по 16 бит A, B, C и D. Если два документа отличаются максимум на 2 бита, то разные биты появятся максимум в двух этих частях. Соответственно, остальные части не будут отличаться.



Могут быть идентичны A и C, могут быть B и C, но представим, что мы знаем, что это всегда A и B. Будем хранить все simhash-и в отсортированном порядке, и затем можем найти множество элементов, у которых тот же префикс AB и сравнивать только с этими элементами. Можем найти это за логарифмическое время.

Так как мы заранее не знаем, какие диапазоны бит совпадут, будем хранить несколько разных комбинаций: AB, AC, AD, BC, BD, or CD. Итак, у нас будет 6 сортированных списков, каждый с simhash-ами: ABCD, ACDB, ADBC, BCAD, BDAC and CDAB.



Для любого заданного запроса мы вычисляем фиксированное число сортированных списков, делаем $O(d * ln(n))O(d * ln(n))$ поисков и маленькое число сравнений, и находим все почти одинаковые документы по нашему запросу. Каждый из этих dd запросов можно выполнять параллельно.

Домашнее задание

Будет после 3-его вебинара по вероятностным алг-мам и структурам данных.

Литература и полезные ссылки

Математика

MinHash

Broder, Andrei Z.; Charikar, Moses; Frieze, Alan M.; Mitzenmacher, Michael (1998), "Min-wise independent permutations"

SimHash

Similarity Estimation Techniques from Rounding Algorithms Moses S. Charikar
http://www.cs.princeton.edu/courses/archive/spring04/cos598B/bib/CharikarEstim.pdf (http://www.cs.princeton.edu/courses/archive/spring04/cos598B/bib/CharikarEstim.pdf)

Сравнение алгоритмов MinHash и SimHash

In Defense of MinHash Over SimHash Anshumali Shrivastava, Ping Li
http://proceedings.mlr.press/v33/shrivastava14.pdf (http://proceedings.mlr.press/v33/shrivastava14.pdf)

Описание алгоритмов MinHash, SimHash

WikiPedia https://en.wikipedia.org/wiki/Locality-sensitive_hashing (https://en.wikipedia.org/wiki/Locality-sensitive_hashing)

Mining of Massive Datasets Jure Leskovec Stanford Univ. Anand Rajaraman Milliway Labs Jeffrey D. Ullman Stanford Univ.

http://infolab.stanford.edu/~ullman/mmds/book.pdf (http://infolab.stanford.edu/~ullman/mmds/book.pdf)
http://www.mmds.org/ (http://www.mmds.org/) (есть слайды к книге и видео)

Пример реализации алгоритма MinHash:

- python: https://github.com/chrisjmccormick/MinHash/blob/master/runMinHashExample.py (https://github.com/chrisjmccormick/MinHash/blob/master/runMinHashExample.py)

Примеры реализации алгоритма SimHash:

- python: https://github.com/seomoz/simhash-py (https://github.com/seomoz/simhash-py)
- C++ https://github.com/seomoz/simhash-cpp (https://github.com/seomoz/simhash-cpp)

Спасибо
за внимание!


OTUS
ОНЛАЙН-ОБРАЗОВАНИЕ

Алгоритмы MinHash и SimHash

Михаил Горшков
разработчик

Поиск похожих данных: bag of words, shingling, алгоритмы MinHash и SimHash

Проблематика поиска похожих данных

Одна из основных задач, которые решает дата-майнинг, - это поиск похожих объектов. Например, имеется набор веб-страниц, и среди них нужно найти похожие.

- поиск плагиата;
- поиск зеркал сайтов;
- статьи из одного и того же источника;
- интернет-магазины - нахождение "похожих" покупателей по их покупкам. Зачем? Чтобы сформировать "портрет" типичного покупателя: домохозяйки, фотолюбители и проч. и рекомендовать им товары в зависимости от их профиля.

Поиск точных совпадений

Как бы вы решали задачу поиска точных совпадений?

- сортированный массив объектов;
- сортированный массив хэшей объектов + хэш-таблица. Можно посчитать хэш от каждого объекта, отсортировать хэши и найти все одинаковые. Далее сравнить объекты, соответствующие одинаковым хэшам.

В любом случае нужно хранить целевые данные - нужно много места - *O(N)* + сортировка.

Поиск похожих объектов

Если ищем не точные совпадения, а похожие объекты, просто посчитать хэши не получится. Небольшое изменение объекта -> кардинальное изменение хэша.

Вопрос к аудитории: что можно сделать?

Техники нахождения похожих объектов

- **bag of words** – сравнение множества слов в одном документе с множеством слов в другом;
- **shingling** – улучшает "bag of words" за счет сравнения коротких фраз, обеспечивая у слов контекст;
- **hashing** – улучшает процесс за счет отсутствия необходимости хранить копии контента. Фразы хэшируются, полученные хэши затем можно сравнивать для нахождения дубликатов;
- **MinHash** – улучшает процесс хранения хэшей контента;
- **SimHash** – еще улучшает процесс хранения хэшей и детекта дубликатов.

Bag of words

Bag of words - это просто множество слов документа. Например, предложение

"*a bump on the log in the hole in the bottom of the sea*"

будет представлено как множество

{a, in, of, on, the, log, sea, bump, hole, bottom}.

Для определения похожести предложений просто сравним множества их слов.

Как сравнивать два множества?

Нужна мера, которая определяет сходство между множествами.

Желательно определить функцию из множеств S, T, U, ... в число.

Ваши предложения?

Похожесть по Жаккару (Jaccard)

Рассмотрим понятие "похожесть по Жаккару", которая работает с множествами.

Похожесть по Жаккару применяется для 2 или нескольких множеств.

Обозначим через

$$|S|$$

мощность множества S (для конечных множеств это число элементов).

Для множеств S и T мера похожести по Жаккару это

$$\frac{|S \cap T|}{|S \cup T|}$$

то есть, отношение размера пересечения множеств S и T к их объединению. Будем обозначать похожесть по Жаккару множеств S и T через $SIM(S, T)$.

$SIM(S, T) = SIM(T, S)$;

$SIM(S, S) = 1$;

$SIM(S, \varnothing) = 0$;

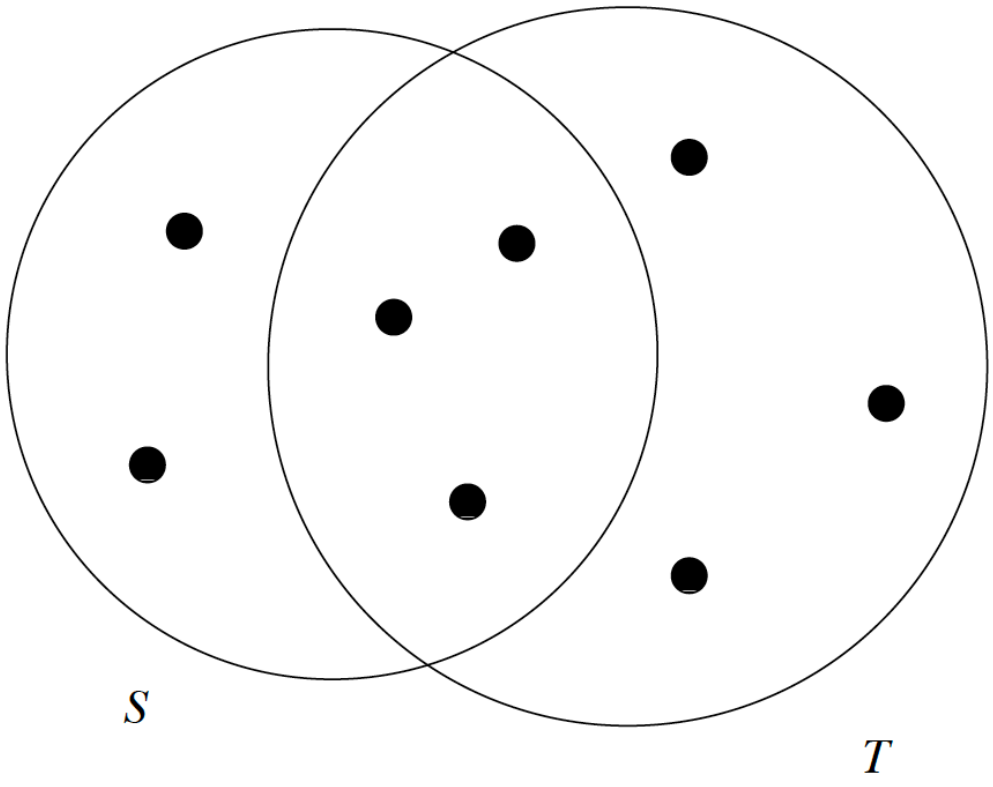
$SIM(\varnothing, \varnothing) = ? ? ?$.

Чем больше мера Жаккара, тем больше похожи множества.

Использование похожести по Жаккару для сравнения по методу Bag of words

Посчитаем меру Жаккара для двух множеств на примерах.

Пример 1



Два множества S и T. Пересечение содержит 3 элемента, объединение 8 элементов. Таким образом, $SIM(S, T) = \frac{3}{8}$.

Упражнение 1

Найдите меру похожести по Жаккару у множеств *{a, a, a, b}* и *{a, a, b, b, c}* (например, это могут быть две корзины покупателей интернет-магазина). В данном случае повторяющиеся элементы возможны.

- пересечение = ?
- объединение = ?
- похожесть по Жаккару = ?

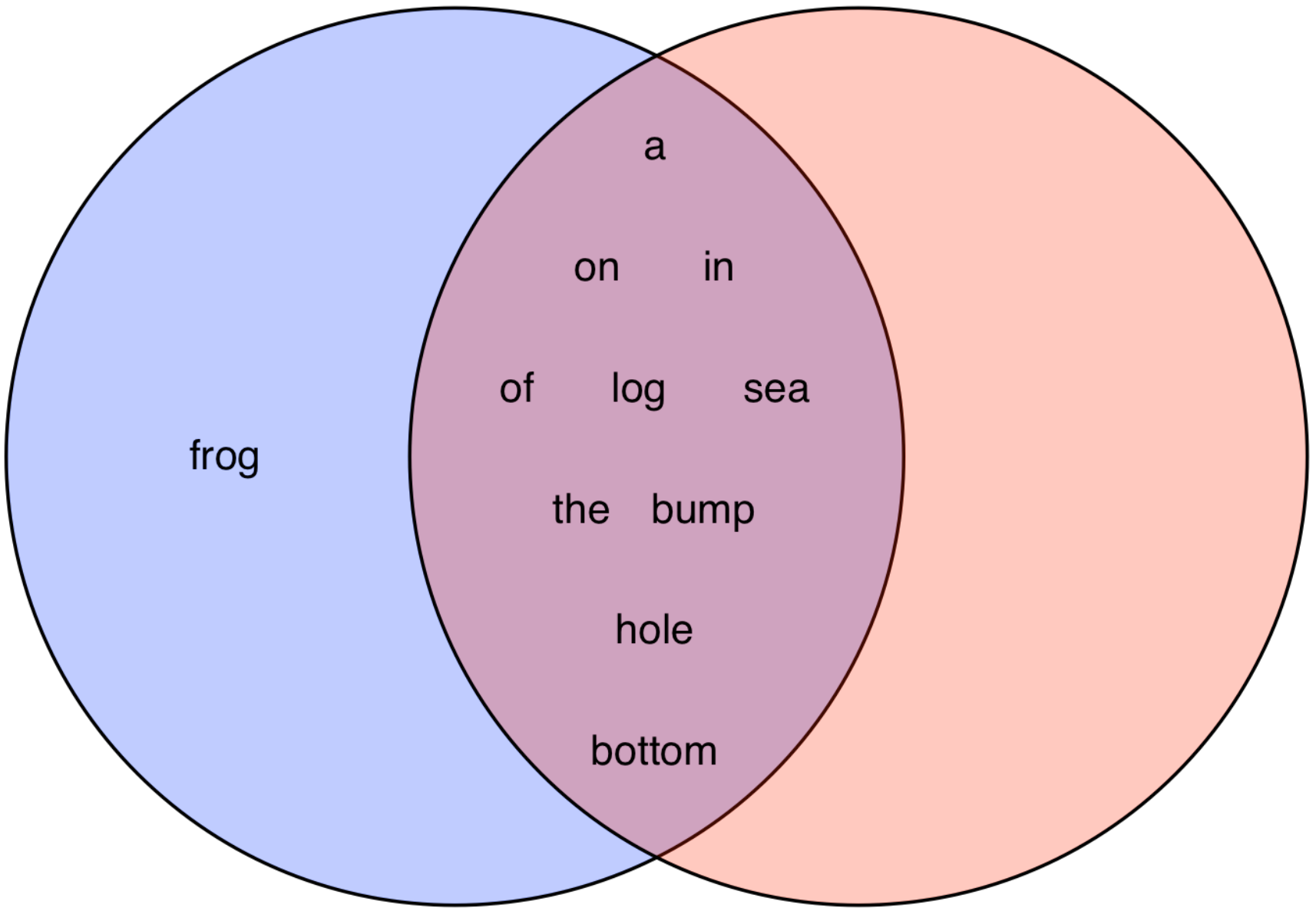
Упражнение 2

Сравним по Жаккару предложение

"*a bump on the log in the hole in the bottom of the sea*"

с предложением

"*a frog on the bump on the log in the hole in the bottom of the sea*"



Чему равно SIM?

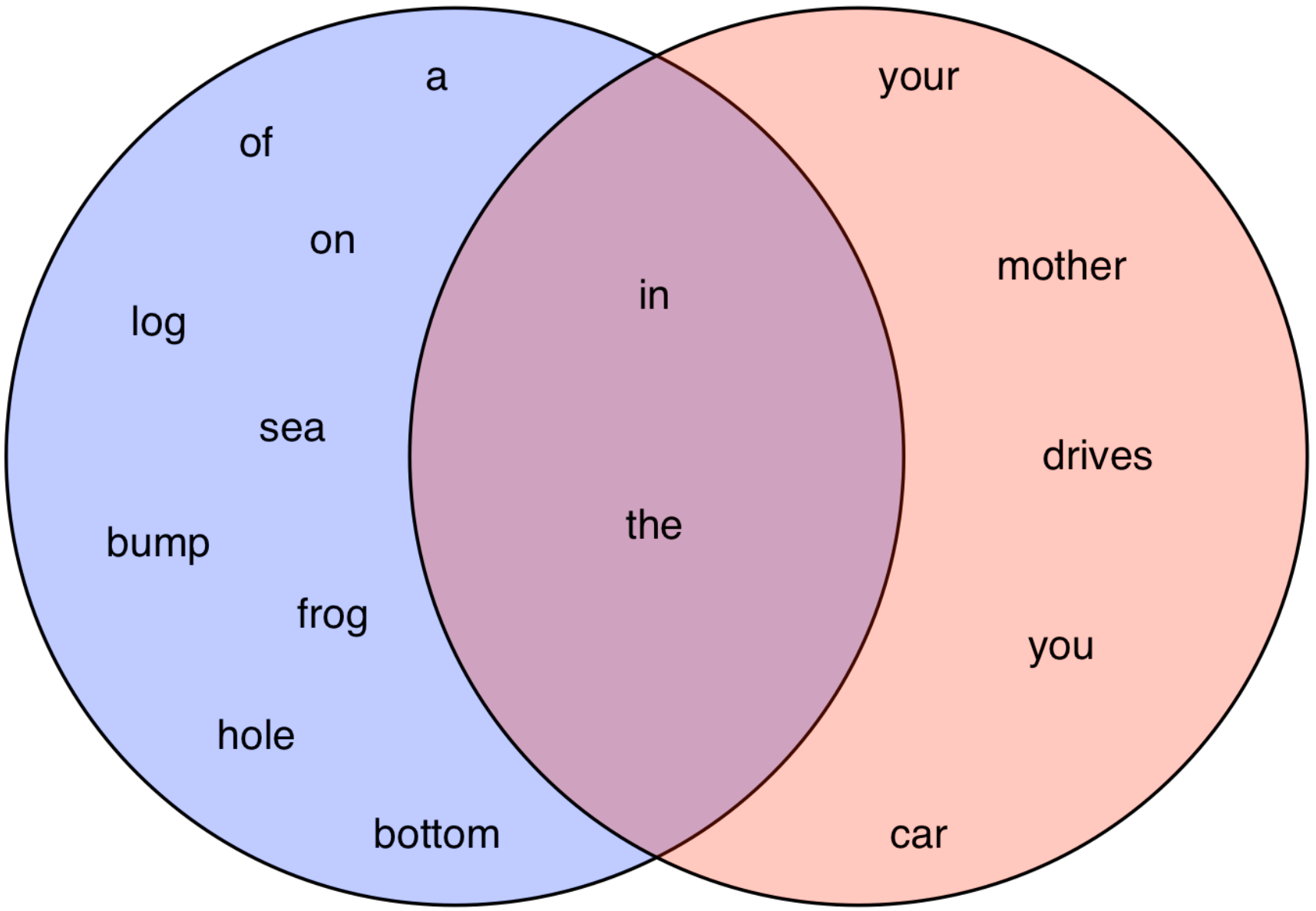
Упражнение 3

Сравним по Жаккару предложение

"*a bump on the log in the hole in the bottom of the sea*"

с предложением

"*your mother drives you in the car*"



***SIM* = ?**

Проблема

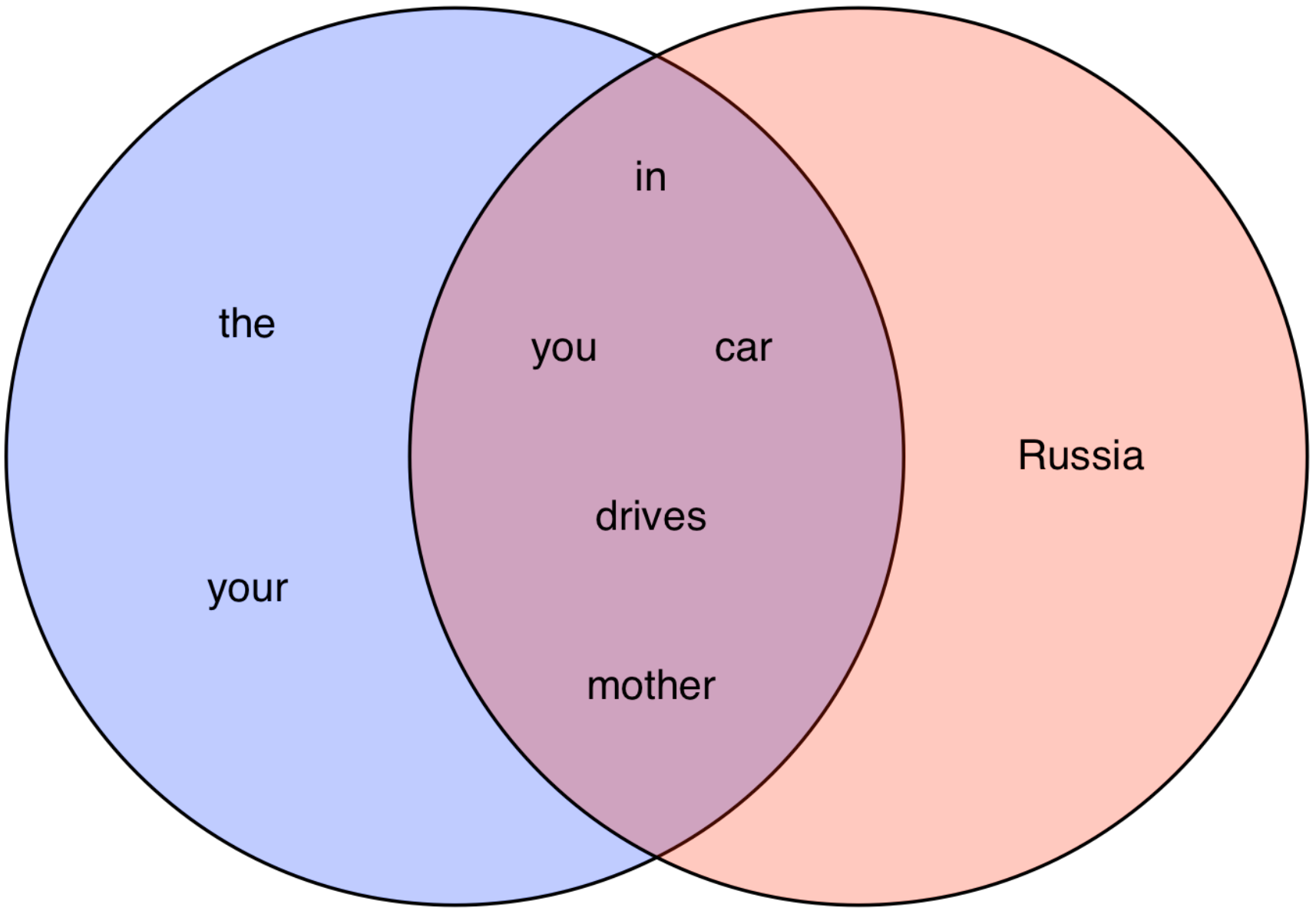
С bag of words есть проблема: какая? (вопрос к аудитории)

Сравним по Жаккару предложение

"*your mother drives you in the car*"

с предложением

"*In mother Russia, car drives you!*"



Пересечение содержит 5 элементов, объединение 8 элементов. Таким образом, $SIM = \frac{5}{8}$, хотя предложения с точки зрения человека полностью различны.

Нужен лучший подход!

Вопрос к аудитории: что можно сделать?

Шинглинг (shingling) документа

Нужен контекст, в котором находится фраза. Для контекста берем соседние элементы-слова.

A shingle - "галька", "место, покрытое галькой", "короткая женская стрижка", "кусочек материала для покрытия крыши".

To shingle - "выкладывать так, чтобы кусочки перекрывались", "покрывать крышу шинглами".

Как работает шинглинг?

- из документа конструируется набор коротких строк, которые в нем содержатся;
- если документы имеют общие фразы или предложения, то в их наборах коротких строк будет много общих элементов;
- определим k-шингл для документа как любую подстроку длиной k, находящуюся в документе;
- сопоставим каждому документу его набор k-шинглов.

Пример 1

Документ D - строка $abcdabd$, берем $k = 2$. Тогда набор 2-шинглов для D будет *{ab, bc, cd, da, bd}*.

Упражнение

"*your mother drives you in the car*", $k = 3$.

Представьте в виде набора шинглов!

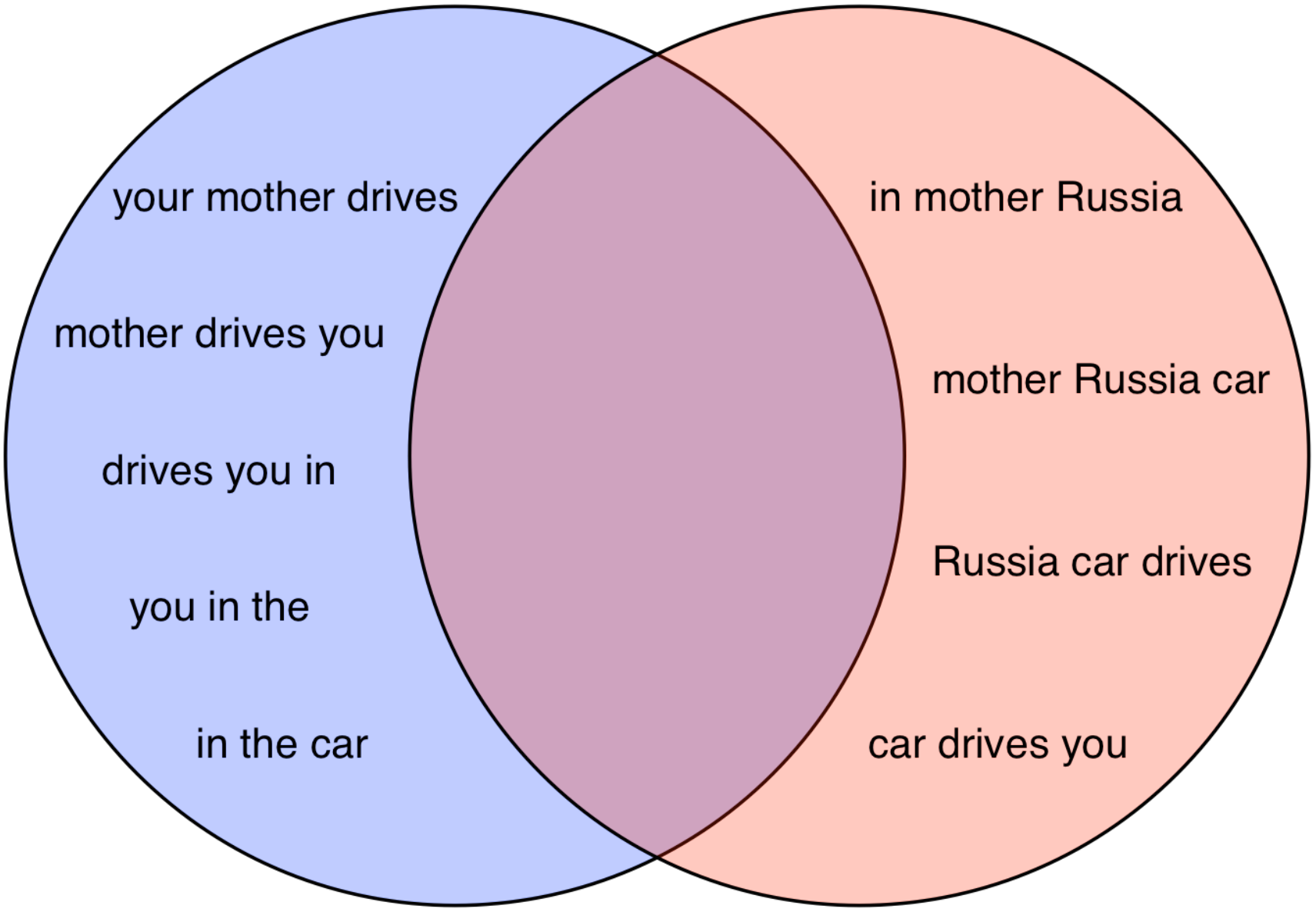
Похожесть по Жаккару наборов шинглов

Еще раз сравним предложение

"*your mother drives you in the car*"

с предложением

"*in mother Russia, car drives you!*"



$SIM = 0$

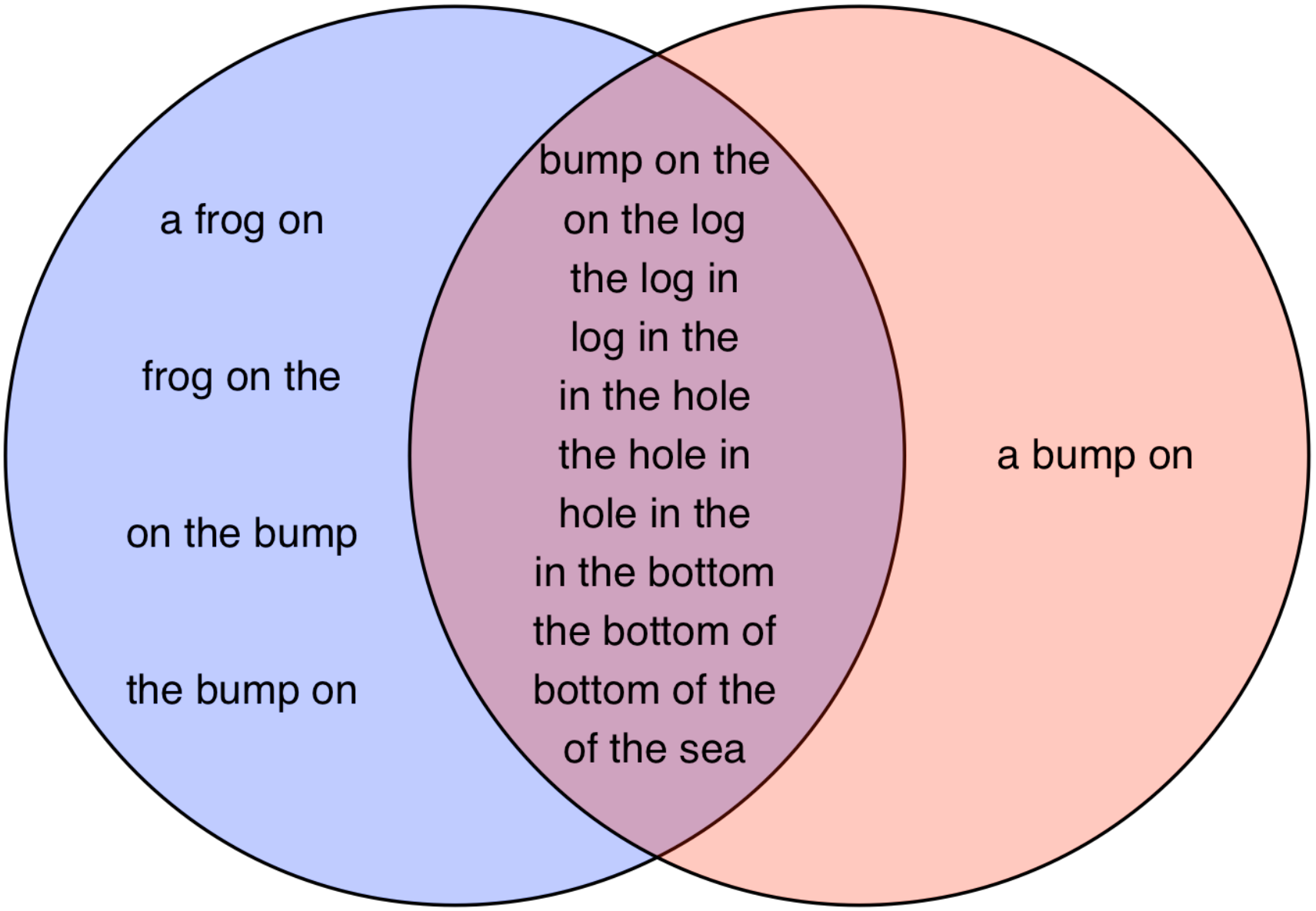
Вернемся к примеру с лягушкой.

Сравним предложение

"*a bump on the log in the hole in the bottom of the sea*"

с предложением

"*a frog on the bump on the log in the hole in the bottom of the sea*"



$$SIM = \frac{11}{16}$$

Выбор размера шингла

- если выбрать k слишком маленьким, во многих документах будут одни и те же шинглы - точность определения похожести снизится;
- если выбрать k слишком большим, не найдутся похожие документы, так как набор шинглов будет отличаться даже для похожих документов;
- **k следует выбрать настолько большим, чтобы вероятность появления любого заданного шингла в любом документе была небольшой**.

Грубо: если у нас корпус е-майлов, выбор k = 5 достаточен.

Почему так? Пусть имеем 20 наиболее употребимых символов в е-майл. Имеем $20^5 \approx$ 3 млн возможных шинглов. Так как обычно е-майл гораздо короче 3 млн симв, k = 5 подходит.

Для обычных текстов можно брать k = 9.

Проблема: шинглы занимают слишком много места

Можно хранить наборы шинглов для документов и сравнивать их между собой, но это занимает много места.

Что можно предложить - вопрос к аудитории?

Хэширование шинглов

- не будем использовать сами шинглы, а выберем хэш-функцию, которая мапит шинглы длины k в число - номер корзины;
- результат будем интерпретировать как шингл;
- набор шинглов, представляющий документ, - это набор чисел - номеров корзин;

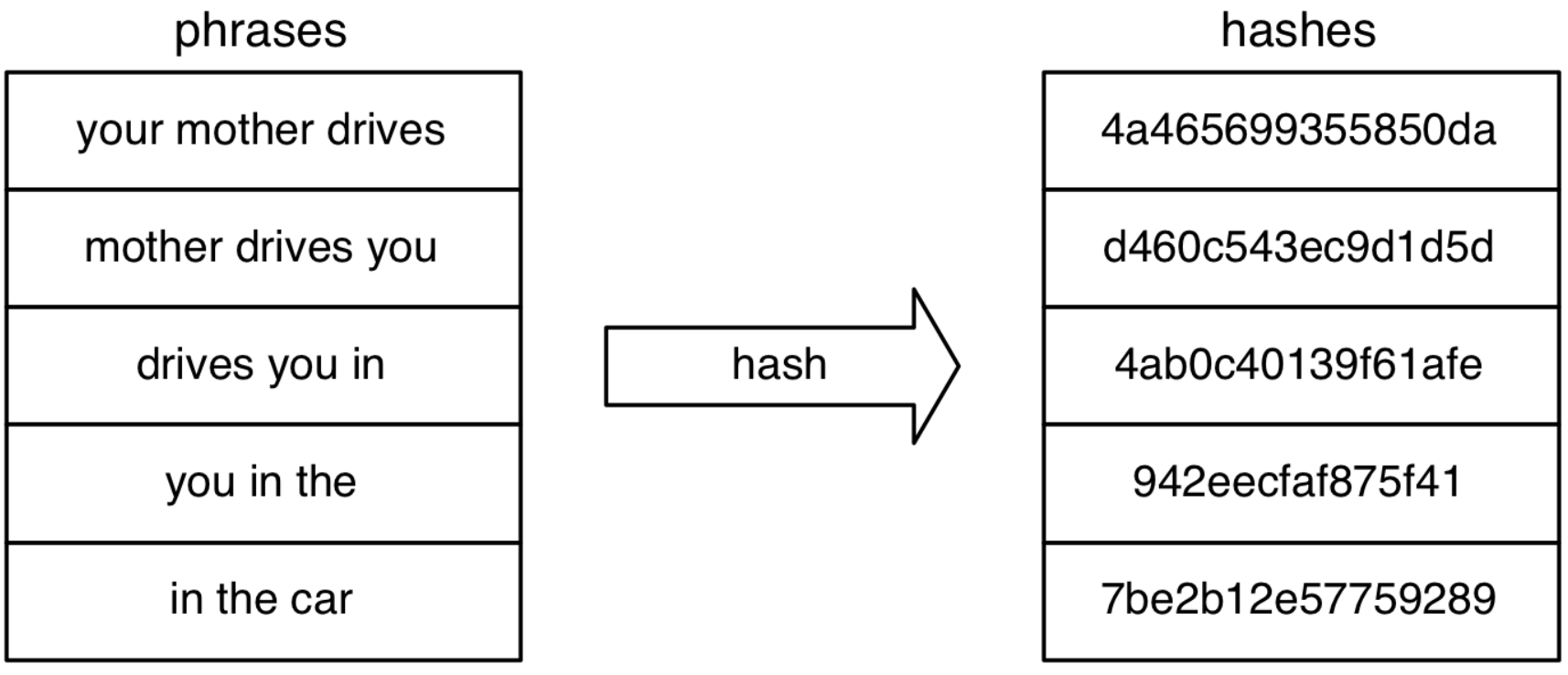
Например, делаем 9-шинглы по документу (посимвольно) и хэшируем их в диапазон 0 - 2^{32} - каждый шингл умещается в 4 байта вместо 9.

Бонусы:

- данные сжимаются;
- после хэширования значения распределяются более равномерно;
- можем манипулировать хэшированными шинглами как числами, что удобно.

Пример

"*your mother drives you in the car*"



Проблема: даже после хэширования большой размер

Хэши могут занять значительно больше места, чем занимал исходный документ.

Если документов миллионы -> хэши не поместятся в оперативную память -> трудно работать с ними. Даже если поместятся, очень ресурсоемко будет сравнивать все пары документов, вычисляя для каждой пары меру Жаккара.

Идеально:

- для всего документа иметь одну "подпись", а не множество хэшей;
- хорошо бы, чтобы мера похожести по Жаккару между двумя документами легко вычислялась по двум "подписям" документов.

Цель: найти способ задать короткую "подпись" для документа, при помощи которой можно было бы вычислить расстояние Жаккара

Есть ли предложения как это сделать?

Матричное представление множества шинглов

Визуализируем множество шинглов с помощью характеристической матрицы.

- матрица представляет собой визуализацию множества шинглов для одного документа;
- столбцы матрицы соответствуют множествам шинглов, строки соответствуют элементам "универсального множества", откуда взяты элементы наших множеств;
- универсальное множество - общее для всех документов;
- если элемент, соответствующий строке r является членом множества для колонки c, в соответствующей ячейке матрицы стоит 1. Иначе 0;
- матрицы часто разреженные **(почему? вопрос к аудитории)**;
- это только для визуализации, в реальности не будем хранить так данные.

Пример

Универсальное множество: *{a, b, c, d, e}*.

S_1 = *{a, d}*;

S_2 = *{c}*;

S_3 = *{b, d, e}*;

S_4 = *{a, c, d}*.

Элемент/Множество	S_1	S_2	S_3	S_4
a	1	0	0	1
b	0	0	1	0
c	0	1	0	1
d	1	0	1	1
e	0	0	1	0

Подписи для множества шинглов

Разработаем идею "подписей" множества шинглов.

- множества шинглов большие: шингл может быть 16 байт, а хэш - 4 байта;
- заменим большие множества шинглов на небольшие подписи.

Бонусы:

- можем посчитать подписи у документов и просто сравнить их для оценки похожести по Жаккару.

Введем понятие ***minhash*** характеристической матрицы - это и будет подписью множества шинглов документа.

MinHash

Чтобы посчитать **minhash** множества, представленного любой колонкой характеристической матрицы (то есть любого из множеств S_i), выберем какую-либо пермутацию (случайную перестановку) строк этой матрицы.

MinHash - *это функция отображения множества в число*.

MinHash любой колонки - это первый по счету элемент, в перемешанном порядке, в которой колонка содержит единицу.

Пример

Вернемся к предыдущей матрице

Элемент/Множество	S_1	S_2	S_3	S_4
a	1	0	0	1
b	0	0	1	0
c	0	1	0	1
d	1	0	1	1
e	0	0	1	0

Пусть мы выбрали порядок строк *beadc* для неё:

Элемент/Множество	S_1	S_2	S_3	S_4
b	0	0	1	0
e	0	0	1	0
a	1	0	0	1
d	1	0	1	1
c	0	1	0	1

Эта перестановка определяет **minhash**-функцию h, которая мапит наборы в строки.

Вычислим значение **minhash** множества S_1 в соответствии с h.

- первая колонка для множества S_1, имеет 0 в строке b, поэтому переходим к строке e;
- там также 0 в колонке для S_1;
- переходим к строке a, там 1.

Таким образом, $h(S_1) = a$.

Аналогично, опускаемся вниз по остальным колонкам, пока не дойдем до 1. Получаем:

$h(S_2) = c$;

$h(S_3) = b$;

$h(S_4) = a$.

Понятно ли это?

Minhash и похожесть по Жаккару

Вероятность того, что функция minhash для случайной перестановки строк выдает одно и то же значение для двух множеств, равна мере похожести по Жаккару между этими множествами.

$$p(h(S_i) = h(S_j)) = SIM(S_i, S_j)$$

Интуитивно это понятно (напр. на диаграмме S_1 и S_4). Докажем, почему это так в общем случае.

Нужно рассмотреть столбцы для двух множеств.

Рассмотрим столбцы для множеств S_i и S_j, тогда строки можно разделить на 3 категории:

- строки типа X имеют 1 в обоих столбцах;
- строки типа Y имеют 1 в одном из столбцов и 0 в другом;
- строки типа Z имеют 0 в обоих столбцах.

Так как матрица разреженная, большинство строк имеет тип Z. Однако, и $SIM(S_i, S_j)$, и вероятность того, что $h(S_i) = h(S_j)$ определяется соотношением числа строк типов X и Y.

Пусть имеется x строк типа X и y строк типа Y.

Тогда

$$SIM(S_i, S_j) = \frac{x}{x + y}$$

потому что x это размер $S_i \cap S_j$, а $x + y$ это размер $S_i \cup S_j$.

Рассмотрим вероятность того, что $h(S_i) = h(S_j)$.

Если строки были перемешаны случайно, и мы начинаем с начала, вероятность, что мы встретим строку типа X до строки типа Y равна

$$\frac{x}{x + y}$$

Но если первая строка сверху, отличная от Z, - это строка типа X, то $h(S_i) = h(S_j)$.

Почему? Вопрос к аудитории!

С другой стороны, если первая строка не типа Z имеет тип Y, то множество с 1 получает эту строку как его значение **minhash**.

Однако множество с 0 в этой строке точно получит некоторую строку далее в перемешанном списке.

Таким образом, мы знаем, что $h(S_i) \neq h(S_j)$, если мы первой встретим строку типа Y.

Делаем вывод, что вероятность того, что

$$p(h(S_i) = h(S_j)) = \frac{x}{x + y}$$

что одновременно является **мерой похожести по Жаккару между S_i и S_j**.

Получили результат для одной случайно перестановки. Но этого недостаточно, нужно сделать много перестановок, чтобы погрешность была меньше.

Подписи MinHash

Вернемся к набору множеств, представленных характеристической матрицей M. Чтобы представить множества, мы выбираем случайно некоторое число n случайных перестановок строк M, то есть сделаем рандомное число пермутаций.

Назовем **minhash**-функции, определяемые этими пермутациями через $h_1, h_2, \ldots, h_n$. Из колонки, представляющей множество S, сконструируем **minhash**-подпись для S, а именно вектор $[h_1(S), h_2(S), \ldots, h_n(S)]$.

Обычно мы представляем этот список hash-значений как колонку. Таким образом, мы можем сформировать матрицу подписей M, в которой i-я колонка M заменяется **minhash**-подписью для множества i-й колонки.

Заметим, что матрица подписей имеет такое же число колонок, как M, но только n строк.

Несмотря на то, что M можно представить не явно, а в сжатом виде (только с единицами как минимум в одной колонке), матрица подписей может быть гораздо меньше M.

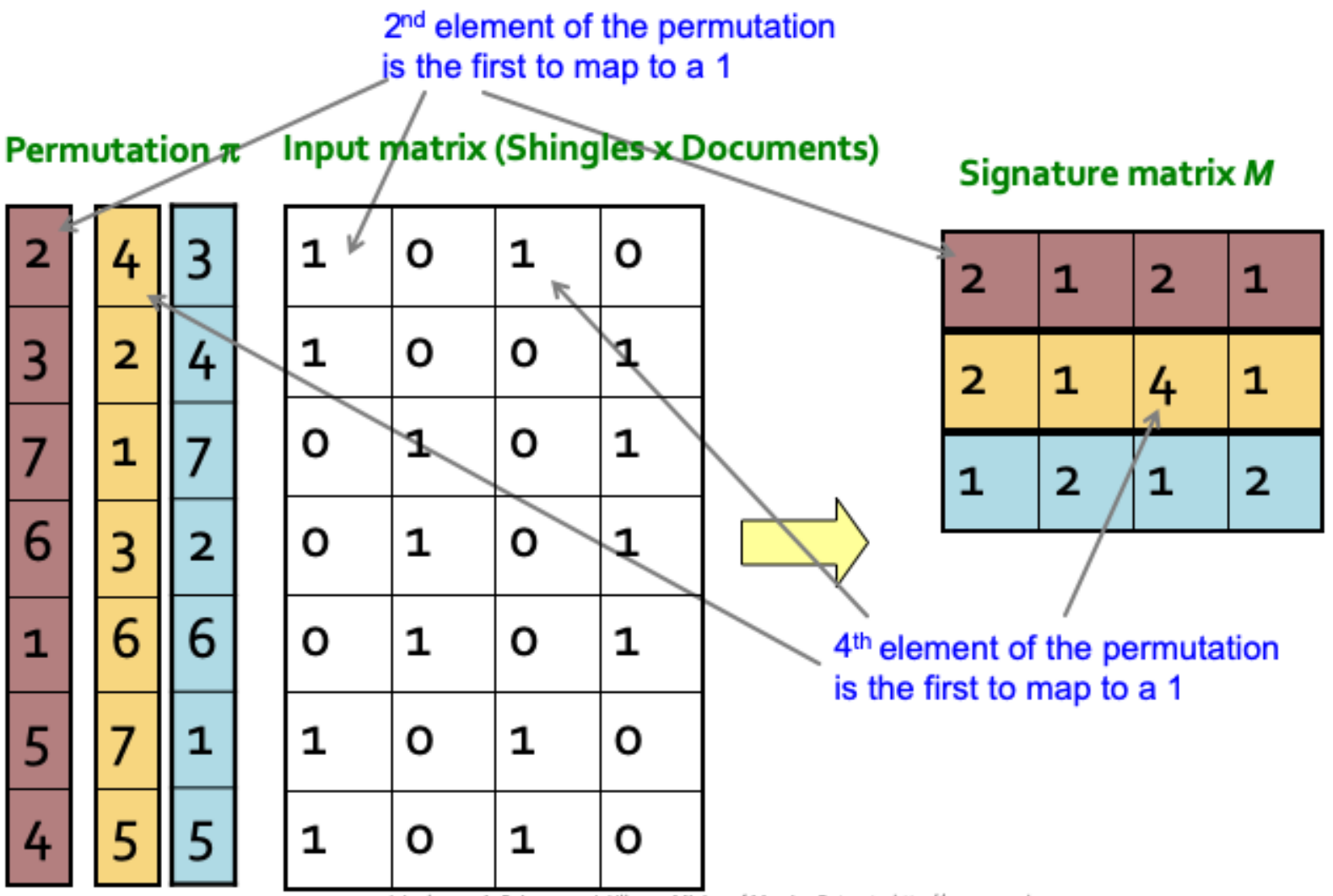
Пример

- входная матрица - не пермутированная;
- альтернатива для h - хранить не сами значения, а индексы.

Min-Hashing Example

Note: Another (equivalent) way is to store row indexes:

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



Похожесть по Жаккару через подписи

Мы знаем, что

$$p(h_k(S_i) = h_k(S_j)) = SIM(S_i, S_j)$$

Обобщим на несколько хэш-функций.

Похожесть двух подписей это доля хэш-функций, в которых они совпадают.

Похожесть колонок такая же, как ожидаемая похожесть их подписей.

Min-Hashing Example

Permutation π

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

Input matrix (Shingles x Documents)

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

Signature matrix *M*

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

Similarities:

	1-3	2-4	1-2	3-4
Col/Col	0.75	0.75	0	0
Sig/Sig	0.67	1.00	0	0

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org 37

Цель достигнута?

- выберем n=100 случайных пермутаций строк;
- пусть sig(C) - колонка-вектор;
- sig(C)[i] = по i-той пермутации индекс (значение) первой строки, в которой колонка C содержит 1;
- подпись (sketch) документа всего ~**100** байт!

Мы сжали длинные битовые вектора в короткие подписи.

Эффективное вычисление подписей MinHash

Проблема

- делать явные пермутации характеристической матрицы слишком затратно;
- даже выбор случайной пермутации из миллионов или миллиардов элементов ресурсоемкая операция, а требуемая сортировка строк (исходной матрицы в соответствии с пермутацией) займет еще больше ресурсов.

Таким образом, пермутированные матрицы аналогичные тем, что мы рассмотрели ранее, сделать концептуально привлекательно, но малореалистично.

Решение

К счастью, возможно симулировать эффект случайной перестановки с помощью случайной хэш-функции, которая мапит номера строк на такое же к-во корзин. Хэш-функция, которая мапит целые числа $0, 1, \ldots, k-1$ в номера корзин от 0 до $k-1$, может замапить некоторые пары чисел в одну и ту же корзину, а другие корзины оставит незаполненными. Однако, это не имеет значения, если k большое и таких коллизий не слишком много.

Таким образом, вместо выбора n случайных пермутаций строк, мы случайно выбираем n хэш-функций $h_1, h_2, \ldots, h_n$ над строками.

Вычислим подпись для матрицы, обсчитывая каждую строку следующим образом.

Пусть $SIG(i, c)$ это элемент матрицы подписей для i-той хэш-функции и столбца c.

Проинитим $SIG(i, c)$ в ∞ для всех i и c. Обсчитываем строку r следующим образом:

- вычислить $h_1(r), h_2(r), \ldots, h_n(r)$;
- для каждой колонки c сделать следующее:
  - если колонка c имеет 0 в строке r, ничего не делаем.
  - если колонка c имеет 1 в строке r, то для каждого $i = 1, 2, \ldots, n$ пусть $SIG(i, c)$ будет наименьшим значением из текущего $SIG(i, c)$ и $h_i(r)$.

Как выбрать случайную хэш-функцию $h(x)$? Универсальное хэширование:

$$h_{a,b}(x) = ((a \cdot x + b)\%P)\%N$$

где:

- a, b - случайные целые числа;
- P - простое число ($P > N$).

Пример вычисления матрицы подписей

Строка	S_1	S_2	S_3	S_4
a	1	0	0	1
b	0	0	1	0
c	0	1	0	1
d	1	0	1	1
e	0	0	1	0

- заменим значения a, b, c, d, e числами 0 - 4;

Строка	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

- выберем хэш-функции $h_1(x) = (x + 1)\%5$ и $h_2(x) = (3x + 1)\%5$
- впишем их результат в новые колонки

Строка	S_1	S_2	S_3	S_4	h_1	h_2
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

- заметим, что эти простые хэш-функции дают настоящие пермутации строк, но это возможно, потому что число строк 5 - простое. В общем случае будут коллизии.

Вычислим матрицу подписей

Изначально матрица состоит из одних ∞

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

Смотрим строку 0 предыдущей матрицы. $h_1(0) = 1$ и $h_2(0) = 1$.

Строка 0 имеет единицы в столбцах для множеств S_1 и S_4, поэтому только эти колонки в матрице подписей могут меняться. Так как 1 меньше ∞, меняем оба значения в колонках для S_1 и S_4.

Текущее состояние матрицы:

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

Теперь переходим к строке 1 предыдущей матрицы. Эта строка имеет 1 только в S_3, и ее значения хэша $h_1(1) = 2$ и $h_2(1) = 4$. Таким образом, $SIG(1, 3) = 2$ и $SIG(2, 3) = 4$. Все другие ячейки остаются как есть, так как для них в строке 1 содержится 0.

Новое состояние матрицы:

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

Строка 2 содержит единицы в колонках для S_2 и S_4, и ее значения хэша $h_1(2) = 3$ и $h_2(2) = 2$. Мы можем поменять значения в подписи для S_4, но значения в этой колонке матрицы подписей каждое меньше, чем соответствующее значение хэша. Однако, так как колонка для S_2 содержит бесконечности, мы заменяем ее на (3, 2).

Результат:

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

Затем рассмотрим строку 3. Все колонки, кроме S_2, содержат 1, и значения хэша $h_1(3) = 4$ и $h_2(3) = 0$. Значение 4 для h_1 уже превышает то, что содержится в матрице подписей для всех колонок, поэтому мы не будем менять значения в первой строке матрицы подписей. Однако, значение 0 для h_2 меньше, чем то, что уже присутствует, так что мы уменьшаем $SIG(2, 1)$, $SIG(2, 3)$ и $SIG(2, 4)$ до 0. Заметим, что мы не можем уменьшить $SIG(2, 2)$, потому что колонка для S_2 содержит 0 в строке, которую мы рассматриваем.

Получается матрица:

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

Наконец, рассмотрим строку 4. $h_1(4) = 0$ и $h_2(4) = 3$. Так как строка 4 имеет 1 только в колонке для S_3 , мы сравниваем текущую колонку матрицы подписей (2, 0) только со значениями хэша (0, 3). Так как $0 < 2$, мы меняем $SIG(1, 3)$ на 0, но так как $3 > 0$, мы не меняем $SIG(2, 3)$.

Окончательная матрица подписей:

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

Оценим похожесть по Жаккару

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

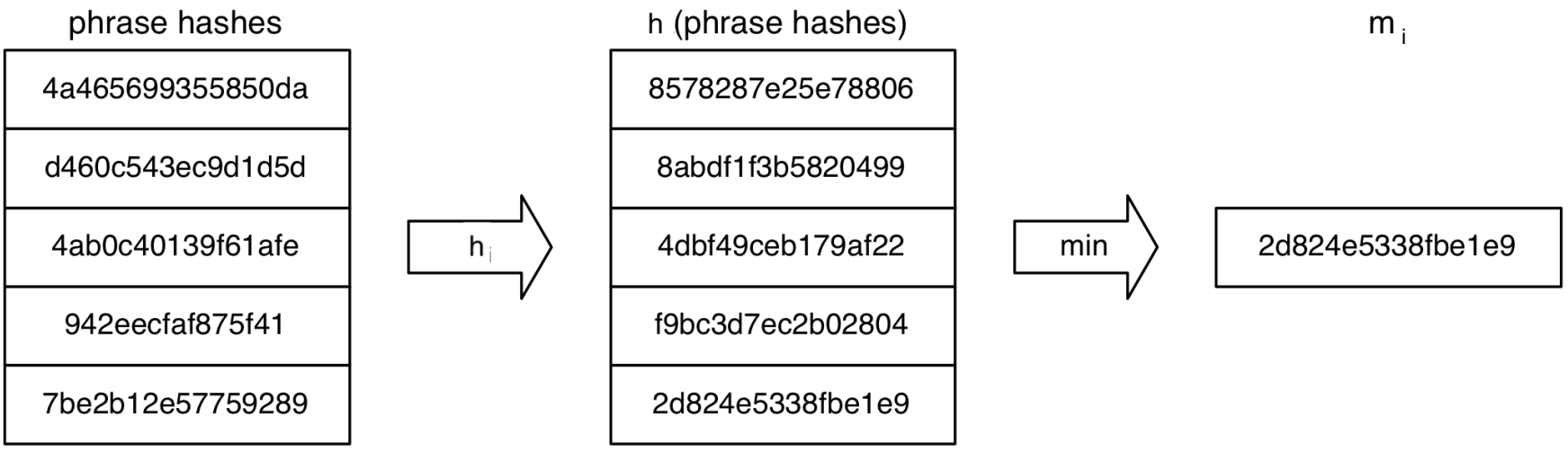
Оценим похожесть по Жаккару между множествами по матрице подписей.

Затемим, что колонки 1 и 4 идентичны, так что мы догадываемся, что $SIM(S_1, S_4) = 1.0$. По предыдущей матрице мы видим, что истинная похожесть по Жаккару между S_1 и S_4 равна 2/3. На самом деле по ЗБЧ приближение будет более точным с увеличением размерности.

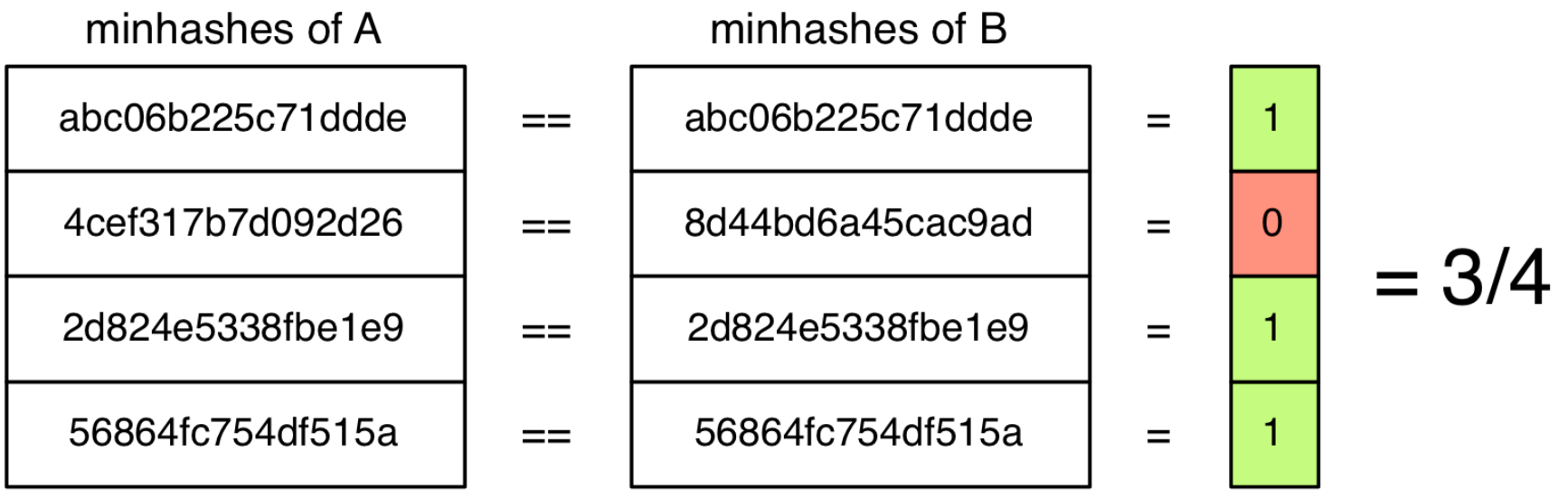
Колонки в подписях S_1 и S_3 совпадают в половине строк (точное расстояние 1/4), колонки S_1 и S_2 отличаются на 100%, похожесть по Жаккару - 0 (это верная оценка).

Для нашего примера выше

Документу соответствует набор хэшей фиксированного размера, вычисленных с помощью n рандомизирующих хеш-функций $h(i)$.



Подпись документа - это сортированный список минимальных хэшей m_0... m_{n-1}. Этот метод достигает приближения к похожести по Жаккару попарным сравнением минимальных хешей каждого документа.



Теперь требуемое место для каждого документа $O(1)$, сложность сравнения двух документов $O(1)$.

Реализация алгоритма

Рассмотрите дома.

https://github.com/chrisjmccormick/MinHash/blob/master/runMinHashExample.py (https://github.com/chrisjmccormick/MinHash/blob/master/runMinHashExample.py)

Проблема алгоритма MinHash

Проблема сложности сравнения документов устранена, но осталась проблема нахождения всех дубликатов из набора документов. С помощью MinHash мы вынуждены сканировать все документы за время $O(N)$. Что делать, если у нас сотни миллионов документов?

Алгоритм SimHash

Алгоритм **SimHash** помогает решить эту проблему. По набору входных хэшей (хэшированные шинглы) **SimHash** строит единственный хэш со свойством — два похожих множества входных хэшей дают такие же результирующие хэши. Большинство хэширующих функций обладают свойством, что немного отличающиеся input-ы производят очень разные output-ы, что делает **SimHash** уникальным.

Для каждой битовой позиции мы считаем число входных хешей, в которых этот бит установлен и вычитаем количество входных хешей, в которых этот бит не установлен. После этого каждая позиция с отрицательным счетчиком сбрасывается в 0, а все остальные в 1.

	1	0	1	1	1	0	1	1
	0	0	1	0	1	1	1	0
	0	1	1	0	0	0	1	1
inputs	0	1	0	0	0	0	1	0
	1	1	1	1	0	0	1	1
	1	0	0	1	1	1	0	0
	0	0	0	0	1	0	1	1
counters	-1	-1	1	-1	1	-3	6	1
result	0	0	1	0	1	0	1	1

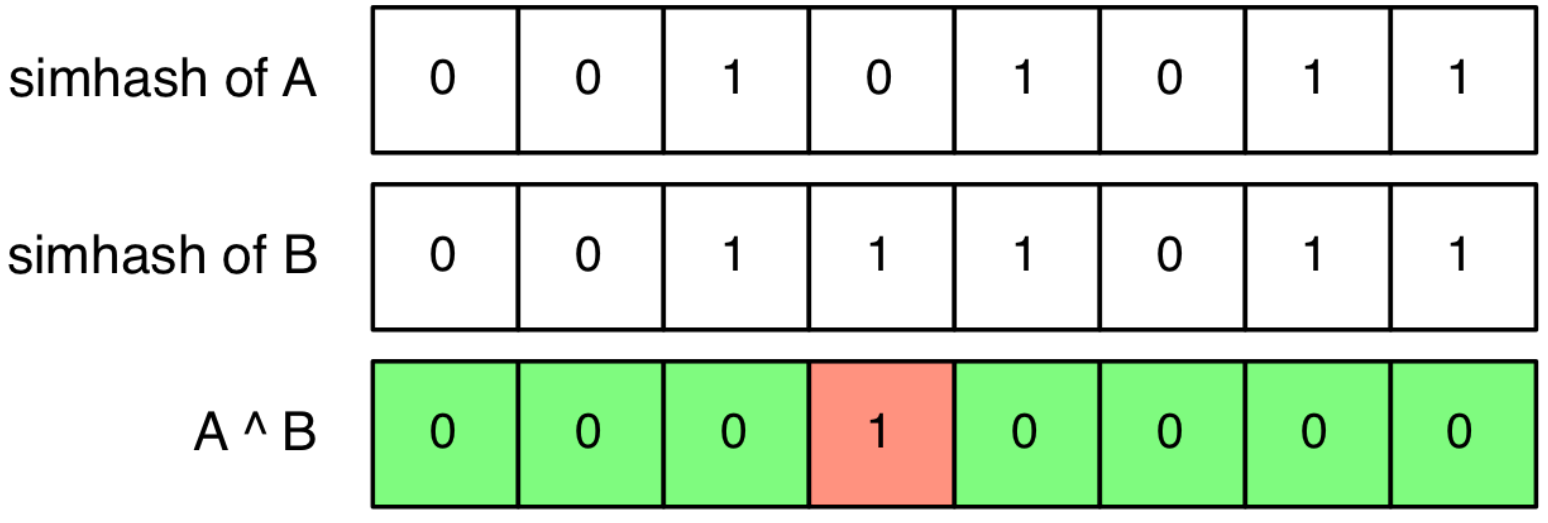
Вычисление расстояния между SimHashes

Чтобы измерить похожесть двух simhashes, мы подсчитываем количество бит, которые отличаются между двумя запросами (**расстояние Хэмминга**). Наоборот, к-во бит, которые одинаковы, это мера похожести.

Как вычислить число бит, в которых значения отличаются? - вопрос к аудитории

Расстояние Хэмминга

bits population(a ^ b).



Bit population(A ^ B) = 1

Выводы

- вместо того, чтобы хранить много хэшей для каждого документа, можно хранить единственное целое число;
- удобнее стало находить набор почти одинаковых документов.

#Сортировка SimHashes

Зачем сортировать документы по их SimHashes?

Несортированные документы

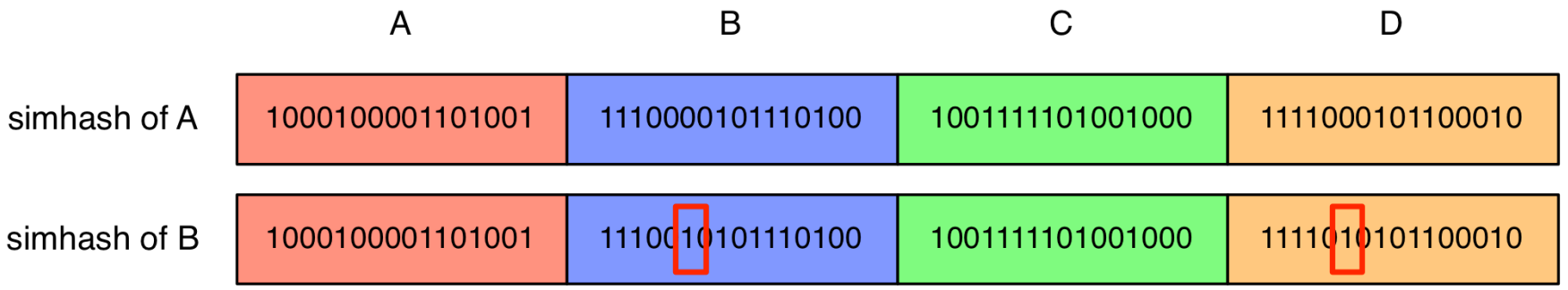
Номер	SimHash	Binary	Hamming from prev
1	37586	1001001011010010	
2	50086	1100001110100110	7
3	2648	0000101001011000	11
4	934	0000001110100110	9
5	40957	1001111111111101	9
6	2650	0000101001011010	9
7	64475	1111101111011011	7
8	40955	1001111111111011	4

Если отсортировать документы по SimHashes, то легко находятся документы с минимальным расстоянием по Хэммингу

Номер	SimHash	Binary	Hamming from prev
4	934	0000001110100110	
3	**2648**	**0000101001011000**	**9**
6	**2650**	**0000101001011010**	**1**
1	37586	1001001011010010	5
8	40955	1001111111111011	6
5	40957	1001111111111101	2
2	50086	1100001110100110	9
7	64475	1111101111011011	9

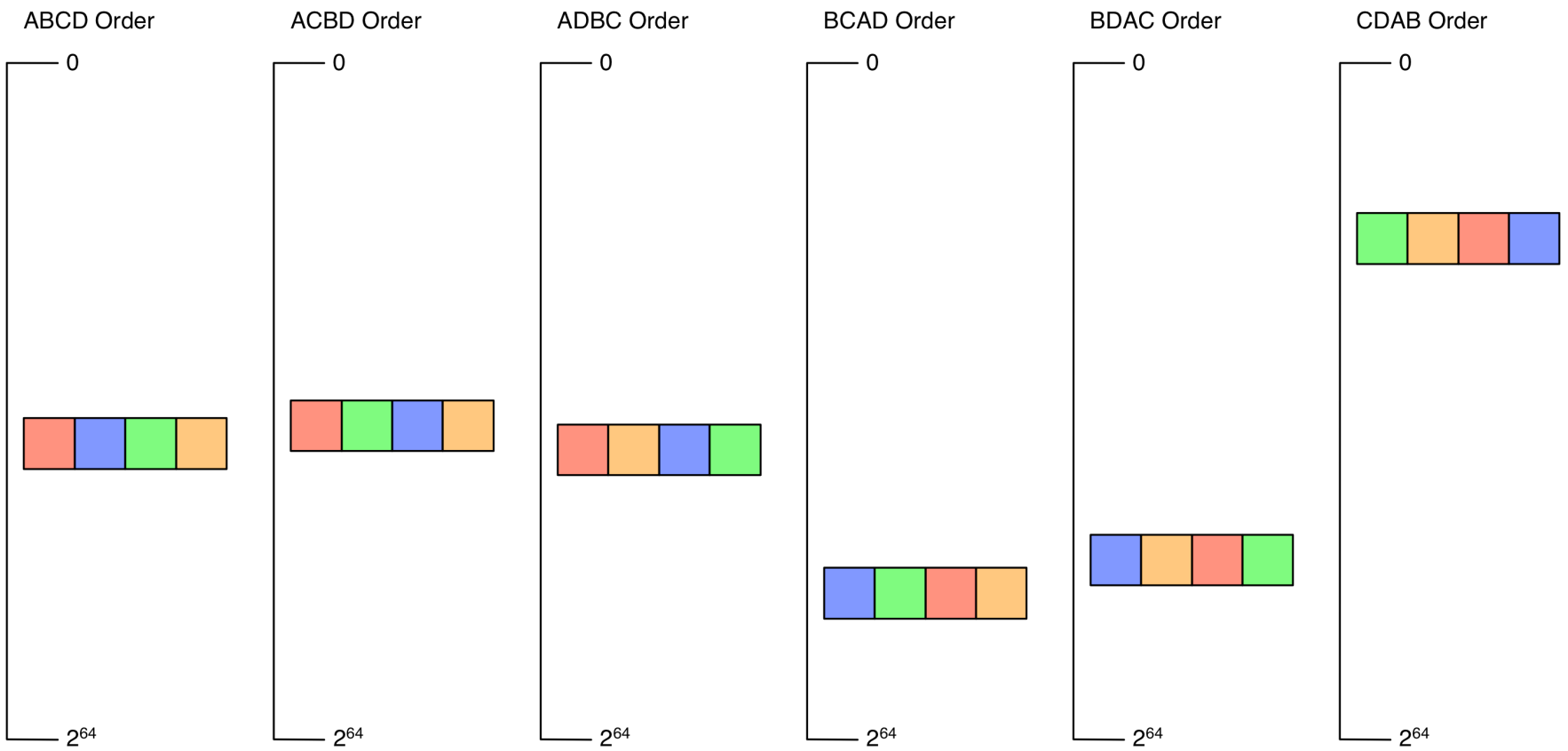
Нахождение набора почти одинаковых документов

Например, предположим, что чтобы документы считались дубликатами, они должны отличаться максимум двумя битами. Разделим наш 64-битный хэш на 4 части по 16 бит A, B, C и D. Если два документа отличаются максимум на 2 бита, то разные биты появятся максимум в двух этих частях. Соответственно, остальные части не будут отличаться.



Могут быть идентичны A и C, могут быть B и C, но представим, что мы знаем, что это всегда A и B. Будем хранить все simhash-и в отсортированном порядке, и затем можем найти множество элементов, у которых тот же префикс AB и сравнивать только с этими элементами. Можем найти это за логарифмическое время.

Так как мы заранее не знаем, какие диапазоны бит совпадут, будем хранить несколько разных комбинаций: AB, AC, AD, BC, BD, or CD. Итак, у нас будет 6 сортированных списков, каждый с simhash-ами: ABCD, ACDB, ADBC, BCAD, BDAC and CDAB.



Для любого заданного запроса мы вычисляем фиксированное число сортированных списков, делаем $O(d * ln(n))$ поисков и маленькое число сравнений, и находим все почти одинаковые документы по нашему запросу. Каждый из этих d запросов можно выполнять параллельно.

Домашнее задание

Будет после 3-его вебинара по вероятностным алг-мам и структурам данных.

Литература и полезные ссылки

Математика

MinHash

Broder, Andrei Z.; Charikar, Moses; Frieze, Alan M.; Mitzenmacher, Michael (1998), "Min-wise independent permutations"

SimHash

Similarity Estimation Techniques from Rounding Algorithms Moses S. Charikar
http://www.cs.princeton.edu/courses/archive/spring04/cos598B/bib/CharikarEstim.pdf (http://www.cs.princeton.edu/courses/archive/spring04/cos598B/bib/CharikarEstim.pdf)

Сравнение алгоритмов MinHash и SimHash

In Defense of MinHash Over SimHash Anshumali Shrivastava, Ping Li
http://proceedings.mlr.press/v33/shrivastava14.pdf (http://proceedings.mlr.press/v33/shrivastava14.pdf)

Описание алгоритмов MinHash, SimHash

WikiPedia https://en.wikipedia.org/wiki/Locality-sensitive_hashing (https://en.wikipedia.org/wiki/Locality-sensitive_hashing)

Mining of Massive Datasets Jure Leskovec Stanford Univ. Anand Rajaraman Milliway Labs Jeffrey D. Ullman Stanford Univ.

http://infolab.stanford.edu/~ullman/mmds/book.pdf (http://infolab.stanford.edu/~ullman/mmds/book.pdf)
http://www.mmds.org/ (http://www.mmds.org/) (есть слайды к книге и видео)

Пример реализации алгоритма MinHash:

- python: https://github.com/chrisjmccormick/MinHash/blob/master/runMinHashExample.py (https://github.com/chrisjmccormick/MinHash/blob/master/runMinHashExample.py)

Примеры реализации алгоритма SimHash:

- python: https://github.com/seomoz/simhash-py (https://github.com/seomoz/simhash-py)
- C++ https://github.com/seomoz/simhash-cpp (https://github.com/seomoz/simhash-cpp)

Спасибо
за внимание!