

# LSML #7

Трюки с хэшированием

# Хэширование полезно

- Уже знаем:
  - Хэширование признаков (vowpal wabbit)
- Узнаем сегодня для **больших** множеств:
  - Определение принадлежности множеству (Bloom Filter)
  - Оценка частот элементов множества в потоке (Count-min sketch)
  - Оценка похожести двух множеств (MinHash)
  - Отбор самых похожих множеств на заданное (LSH)

**Будем решать приближенно!**

# Bloom Filter (1970)

- **Определение принадлежности множеству (в память не лезет)**
- Bloom Filter – это массив из  $m$  бит, который хранит информацию о множестве  $S = \{x_1, x_2, \dots, x_n\}$ . Начинаем с массива нулей.
- Для работы фильтра нам нужно  $k$  независимых хэш-функций  $h_1, \dots, h_k$  с  $m$  корзинками (равномерно раскидывают по корзинкам).
- При вставке каждого элемента  $x \in S$  выставаем биты  $h_i(x)$  в 1 для всех  $i \in [1, k]$ .

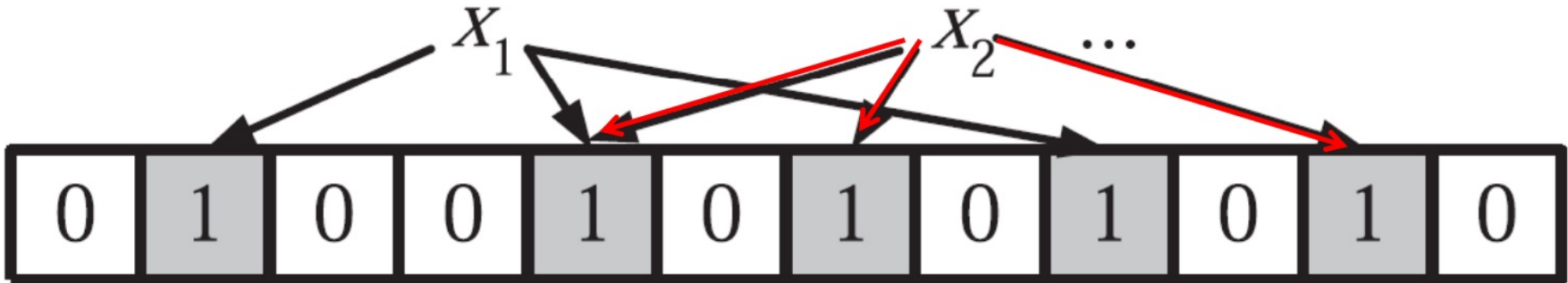
# Вставка и проверка принадлежности

$m = 12$

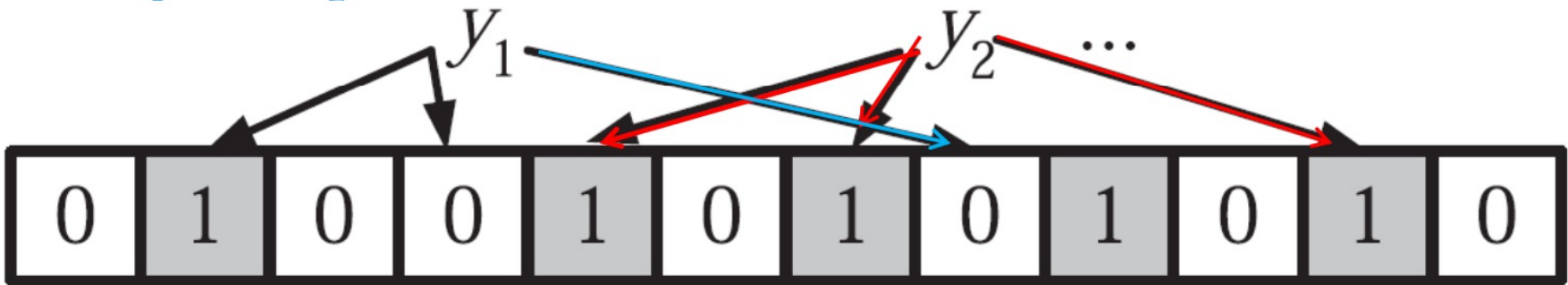


Insert  $X_1$  and  $X_2$

$k = 3$



Check  $Y_1$  and  $Y_2$



$y_1$  **ТОЧНО** нет  
 $y_2$  **ВОЗМОЖНО** есть

# То есть может быть только false positive

- Вероятность того, что  $h_i(x)$  не выставит бит  $j$ :  $1 - \frac{1}{m}$
- Вероятность того, что ни одна из  $k$  хэш-функций не выставит:  $\left(1 - \frac{1}{m}\right)^k$
- Мы вставили  $n$  элементов, значит вероятность не выставить:  $\left(1 - \frac{1}{m}\right)^{kn}$
- Вероятность того, что бит  $j$  в фильтре выставлен:  $1 - \left(1 - \frac{1}{m}\right)^{kn}$
- Вероятность того, что  $k$  бит будут уже выставлены:

$$\left(1 - \frac{1}{m}\right)^m \approx e^{-1} \quad \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \approx \left(1 - e^{-kn/m}\right)^k$$

# Оптимальные параметры для заданной ошибки

- Минимизируем  $(1 - e^{-kn/m})^k$  по  $k$ :  $k = \frac{m}{n} \ln 2$

substituting the optimal value of  $k$  in the probability expression above:

$$p = \left(1 - e^{-\left(\frac{m}{n} \ln 2\right) \frac{n}{m}}\right)^{\frac{m}{n} \ln 2}$$

which can be simplified to:

$$\ln p = -\frac{m}{n} (\ln 2)^2.$$

This results in:

$$m = -\frac{n \ln p}{(\ln 2)^2}$$

So the optimal number of bits per element is

$$\frac{m}{n} = -\frac{\log_2 p}{\ln 2} \approx -1.44 \log_2 p$$

with the corresponding number of hash functions  $k$  (ignoring integrality):

$$k = -\frac{\ln p}{\ln 2} = -\log_2 p.$$

*Хотим:*

$$n = 10^6 \quad p = 0.01$$

*Рассчитываем:*

$$m = 10^7 \quad k = 7$$

# А профит вообще есть?

- Предыдущий пример:

$$n = 10^6 \quad p = 0.01 \quad m = 10^7 \quad k = 7$$

- Пусть множество из строк длины 1024 (например URL страниц)
- Для хранения множества нужно  $1024 \times 10^6$  байт, то есть **1 ГБ**
- Для хранения фильтра:  $10^7$  бит, то есть **1.2 МБ**

# Примеры применения

- Medium uses Bloom filters to avoid recommending articles a user has previously read.
- Bitcoin uses Bloom filters to speed up wallet synchronization.
- Google Bigtable, Apache HBase and Apache Cassandra, and PostgreSQL use Bloom filters to reduce the disk lookups for non-existent rows or columns.



# Пример из ML практики

- Нужно сделать **join**:
  - огромной таблицы с логами юзеров
  - и набором интересующих юзеров (*не влезает в память*)
- Простой join на MapReduce будет шафлить (куча пересылок по сети) огромное количество логов зря → медленно
- Сделаем Map шаг (без пересылок по сети) с фильтрацией огромной таблицы при помощи Bloom Filter → суммарно в 4 раза быстрее

# Операции над множествами

- Bloom Filter заменяет нам само множество с небольшими потерями
- Для объединения множеств нужно применить побитовый OR
- Для пересечения множеств – побитовый AND
- Вычитать сложнее – нужен Counting Bloom Filter...
- Создание Bloom Filter легко распараллелить!

# Count-min sketch (2003)

- Оценка частот элементов множества в потоке (поток и множество большие)
- Заведем двумерный массив из  $d$  строк и  $w$  столбцов.
- Положим  $w = \lceil e/\varepsilon \rceil$  и  $d = \lceil \ln 1/\delta \rceil$ , где ошибка в запросе частоты в пределах аддитивной добавки  $\varepsilon$  с вероятностью  $1 - \delta$ .

# Модель потока

- В момент времени  $t$  счетчики для всех  $n$  элементов:

$$\vec{a}(t) = (a_1(t), \dots, a_i(t), \dots, a_n(t))$$

- Начинаем с нулей:

$$a_i(0) = 0 \quad \forall i$$

- На шаге  $t$  увеличиваем счетчик элемента  $i_t$  на  $c_t$ :

$$a_{i'}(t) = a_{i'}(t-1) \quad \forall i' \neq i_t$$

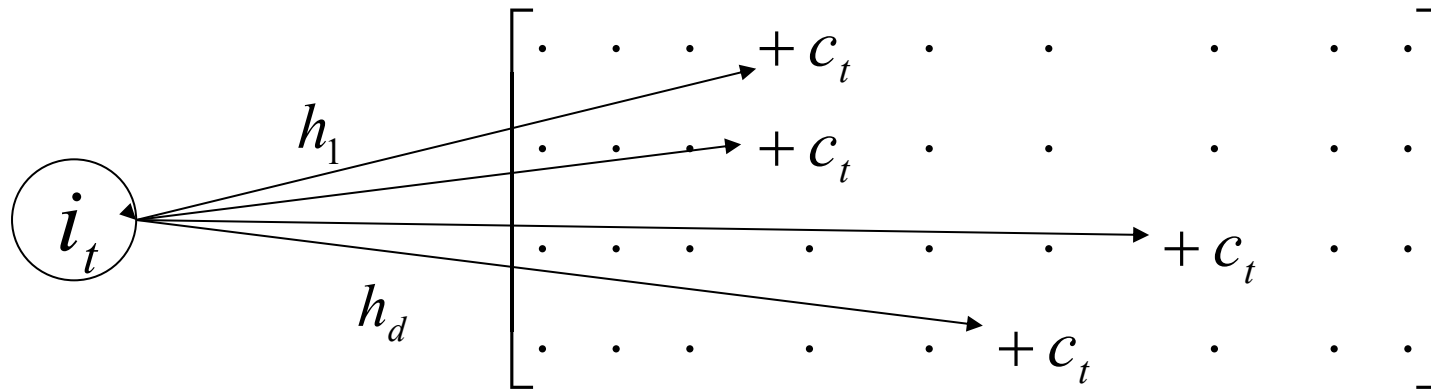
$$a_{i_t}(t) = a_{i_t}(t-1) + c_t$$

# Обновление массива

- Пусть  $h_i(x), i \in [1, d]$  – попарно независимые хэш-функции с  $w$  корзинами

When  $(i_t, c_t)$  arrives, set  $\forall 1 \leq j \leq d$

$$count[j, h_j(i_t)] \leftarrow count[j, h_j(i_t)] + c_t$$



Будем хранить счетчики, а не частоту

# Какие запросы можно делать

• point query  $Q(i)$   $\xrightarrow{\text{approx.}}$   $a_i$

• range queries  $Q(l, r)$   $\xrightarrow{\text{approx.}}$   $\sum_{i=l}^r a_i$

• inner product queries  $Q(\vec{a}, \vec{b})$   $\xrightarrow{\text{approx.}}$   $\vec{a} \cdot \vec{b} = \sum_{i=1}^n a_i b_i$

# Point query ( $a_{i_t}(t) > 0$ )

$$Q(i) \longrightarrow \hat{a}_i = \min_j \text{count}[j, h_j(i)]$$

• Теорема:

$$a_i \leq \hat{a}_i \quad P[\hat{a}_i > a_i + \varepsilon \|\vec{a}\|_1] \leq \delta$$

$$P\left[\frac{\hat{a}_i}{\|\vec{a}\|_1} > \frac{a_i}{\|\vec{a}\|_1} + \varepsilon\right] \leq \delta$$

То есть частота оценивается хорошо

# Point query ( $a_{i_t}(t) > 0$ )

$$Q(i) \longrightarrow \hat{a}_i = \min_j \text{count}[j, h_j(i)]$$

• Теорема: ★  $a_i \leq \hat{a}_i$        $P[\hat{a}_i > a_i + \varepsilon \|\vec{a}\|_1] \leq \delta$

$$I_{i,j,k} = \begin{cases} 1 & \text{if } (i \neq k) \wedge (h_j(i) = h_j(k)) \\ 0 & \text{otherwise} \end{cases}$$

$$w = \lceil e/\varepsilon \rceil$$

$$E(I_{i,j,k}) \leq \Pr[h_j(i) = h_j(k)] = \frac{1}{w} = \frac{\varepsilon}{e}$$

$$X_{i,j} = \sum_{k=1}^n I_{i,j,k} a_k \implies \text{count}[j, h_j(i)] = a_i + X_{i,j} \implies \min \text{count}[j, h_j(i)] \geq a_i$$



# Point query ( $a_{i_t}(t) > 0$ )

$$Q(i) \longrightarrow \hat{a}_i = \min_{j \in [1, d]} \text{count}[j, h_j(i)]$$

- Теорема:  $\star a_i \leq \hat{a}_i$   $\star P[\hat{a}_i > a_i + \varepsilon \|\vec{a}\|_1] \leq \delta$

$$E(X_{i,j}) = E\left(\sum_{k=1}^n I_{i,j,k} a_k\right) = \sum_{k=1}^n a_k E(I_{i,j,k}) \leq \frac{\varepsilon}{e} \|\vec{a}\|_1$$

$$\Pr[\hat{a}_i > a_i + \varepsilon \|\vec{a}\|_1] = \Pr[\forall j. \text{count}[j, h_j(i)] > a_i + \varepsilon \|\vec{a}\|_1]$$

для всех  $j \in [1, d]$   
одновременно

$$= \Pr[\forall j. a_i + X_{i,j} > a_i + \varepsilon \|\vec{a}\|_1]$$

$$\leq \Pr[\forall j. X_{i,j} > eE(X_{i,j})] < e^{-d} \leq \delta$$

Markov inequality

$$\Pr[X \geq t] \leq \frac{E(X)}{t} \quad \forall t > 0$$

$$d = \lceil \ln 1/\delta \rceil$$



# Оценки сложности

$$w = \lceil e/\varepsilon \rceil$$

$$d = \lceil \ln 1/\delta \rceil$$

Time to produce the estimate

$$O(\ln \frac{1}{\delta})$$

Space used

$$O(\frac{1}{\varepsilon} \ln \frac{1}{\delta})$$

Time for updates

$$O(\ln \frac{1}{\delta})$$

# Inner Product Query

Есть счетчики для двух потоков: a и b

$$\text{Set } (\vec{a} \cdot \vec{b})_j = \sum_{k=1}^w \text{count}_{\vec{a}}[j, k] * \text{count}_{\vec{b}}[j, k]$$

$$Q(\vec{a}, \vec{b}) \quad \longrightarrow \quad (\vec{a} \cdot \vec{b}) = \min_j (\vec{a} \cdot \vec{b})_j$$

Например, для оценки размера **join** двух таблиц:

`x join y on x.a=y.b`

# Inner Product Query

- Теорема:  $(\vec{a} \cdot \vec{b}) \leq (\vec{a} \cdot \vec{b})^\wedge \quad \Pr[(\vec{a} \cdot \vec{b})^\wedge > \vec{a} \cdot \vec{b} + \varepsilon \|\vec{a}\|_1 \|\vec{b}\|_1] \leq \delta$

$$(\vec{a} \cdot \vec{b})^\wedge_j = \sum_{k=1}^w \text{count}_{\vec{a}}[j, k] * \text{count}_{\vec{b}}[j, k]$$

$$(\vec{a} \cdot \vec{b})^\wedge = \min_j (\vec{a} \cdot \vec{b})^\wedge_j$$

$$(\vec{a} \cdot \vec{b})^\wedge_j = \sum_{i=1}^n a_i b_i + \sum_{p \neq q, h_j(p)=h_j(q)} a_p b_q \rightarrow (\vec{a} \cdot \vec{b}) \leq (\vec{a} \cdot \vec{b})^\wedge$$

$$E(\vec{a} \cdot \vec{b}_j^\wedge - \vec{a} \cdot \vec{b}) = \sum_{p \neq q} \Pr[h_j(p) = h_j(q)] a_p b_q \leq \sum_{p \neq q} \frac{\varepsilon a_p b_q}{e} \leq \frac{\varepsilon \|\vec{a}\|_1 \|\vec{b}\|_1}{e}$$

Так же, как в прошлый раз

Markov inequality

$$\Pr[X \geq t] \leq \frac{E(X)}{t} \quad \forall t > 0$$

$$\Pr[\vec{a} \cdot \vec{b}^\wedge - \vec{a} \cdot \vec{b} > \varepsilon \|\vec{a}\|_1 \|\vec{b}\|_1] \leq \delta$$

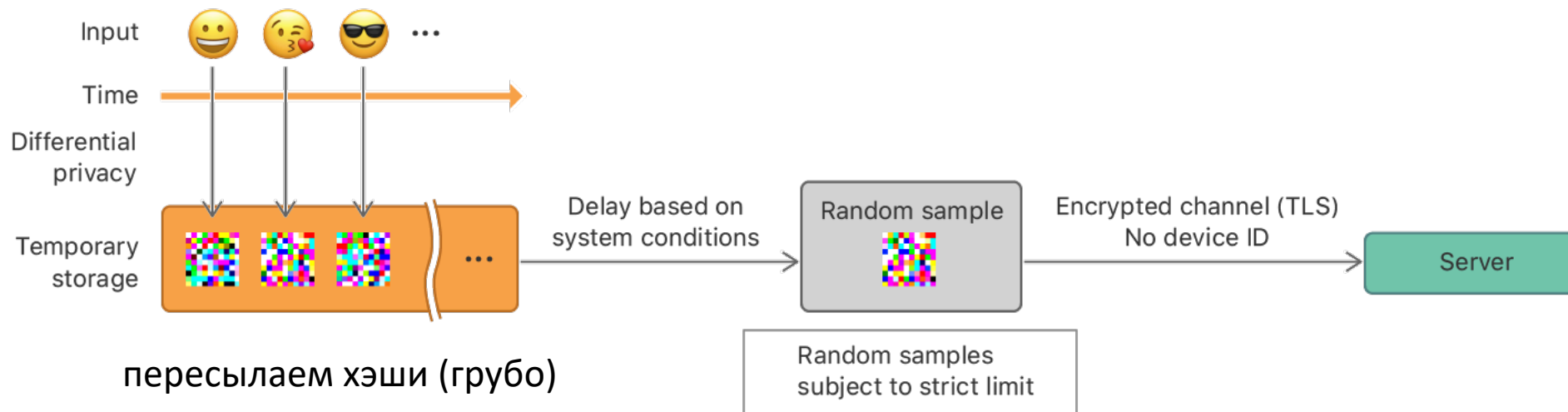
минимум из d штук

# Примеры применений

- Статистики для больших корпусов в NLP
  - <http://www.aclweb.org/anthology/D12-1100>
- Приближенные статистики в Apache Spark
  - <https://databricks.com/blog/2016/05/19/approximate-algorithms-in-apache-spark-hyperloglog-and-quantiles.html>
  - <https://mapr.com/blog/some-important-streaming-algorithms-you-should-know-about/>

# Пример от Apple

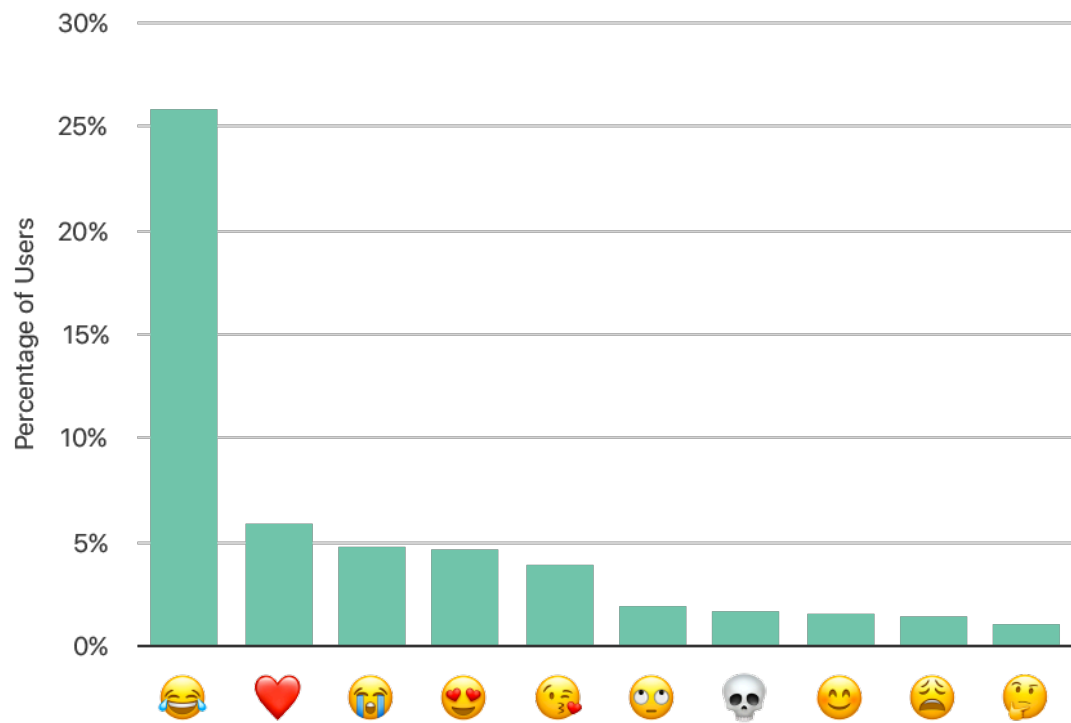
- Learning with Privacy at Scale (2017)
- Нельзя передавать в открытую все, что печатает юзер на клавиатуре
- Отошлем Count-**Mean** Sketch и проверим только интересные нам слова



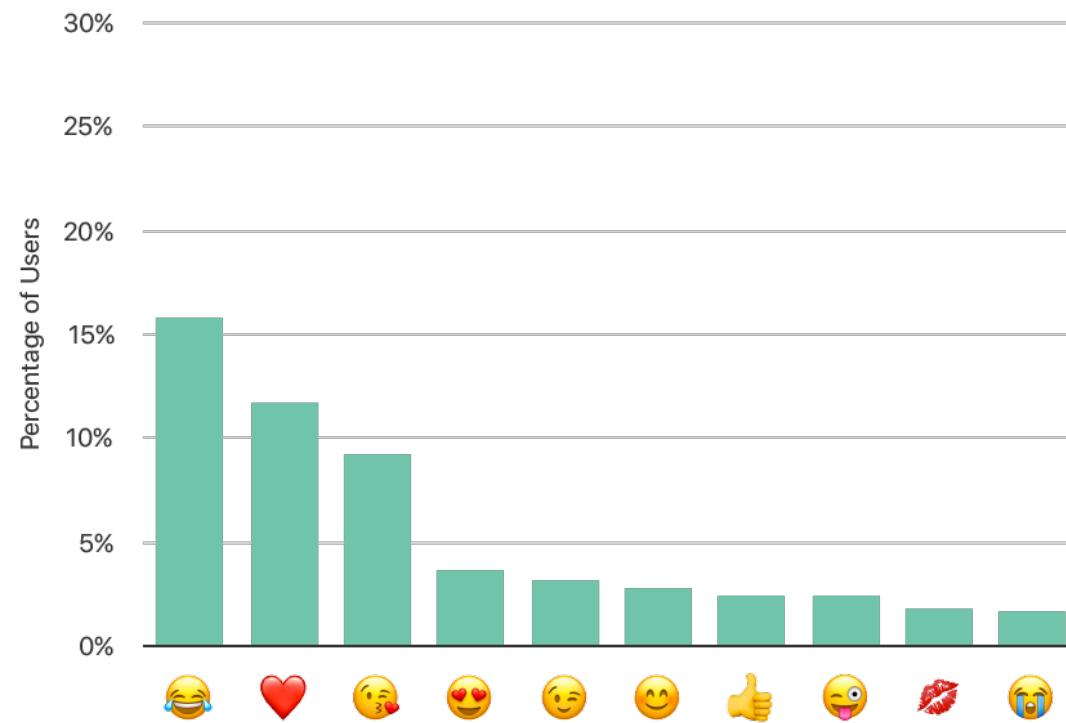
# Пример от Apple

- Можно восстановить распределение

English



French



Можем узнать частоту интересных нам смайликов