Approximative Speicherung von Daten: "Sketching"

Bloom-Filter

Ziel: Eine große Menge $S \subseteq U$ kompakt repräsentieren, so dass bei Eingabe eines Elements $u \in U$ schnell getestet werden kann, ob das Element u

- garantiert nicht zu S gehört

  oder
- möglicherweise zu S gehört

Hierbei wollen wir natürlich möglichst wenige "falsche positive Antworten" (d.h. Antworten der Form "gehört möglicherweise zu S", obwohl das betrachtete Element tatsächlich nicht zu S gehört) bekommen.

Anwendungsbeispiel:

S ist eine "white list" von Email-Adressen, deren Emails der Spam-Filter an die Adressaten ausliefern soll.

Szenario: (aus Leskovec, Rajaraman, Ullman "Mining of Massive Datasets" 2014)

- S enthält 10^9 Email-Adressen
- zum Speichern einer einzelnen Email-Adresse werden ca 20 Bytes benötigt

$\Rightarrow$ S kann nicht im Hauptspeicher abgelegt werden

Aber: Bei Ankunft einer Email soll <u>schnell</u> (ohne Zugriffe auf externen Speicher) entschieden werden, ob die Mail vom Spam-Filter aussortiert oder ob sie an den Adressaten weitergeleitet werden soll.

<u>Lösung</u>:

- Lege im Hauptspeicher eine geeignete "Skizze" (engl: "Sketch") von S ab
- Nimm in Kauf, dass es "falsche positive Entscheidungen" gibt — aber sorge dafür, dass das möglichst selten vorkommt.
- Gestalte die "Skizze" so, dass es garantiert <u>nicht</u> zu "falschen negativen Entscheidungen" kommt (dh: dass Mails von Adressen, die auf der "white list" S stehen, stets an den Adressaten weitergeleitet werden).

Genau dies leisten Bloom-Filter!

Der Bloom-Filter

Szenario: • Im Hauptspeicher steht ein Bit-Array $A[1..n]$ der Länge n zur Verfügung (für eine feste Zahl n).

• Wir wollen eine "Skizze" einer Menge $S = \{s_1, \ldots, s_N\} \subseteq U$ speichern.

Speicherung der "Skizze":

- Das Bit-Array wird auf 0 gesetzt:
  $A[j] := 0$ für alle $j \in \{1, \ldots, n\}$
- Wähle zufällig und unabhängig voneinander k Hash-Funktionen $h_1, \ldots, h_k$, wobei jedes h_i eine Funktion von U nach $\{1, \ldots, n\}$ ist.
  Die Zahl k wird dazu geeignet gewählt (siehe unten).
- Für jedes $s \in S$ tue Folgendes:

  Für jedes $i \in \{1, \ldots, k\}$
  berechne $h_i(s)$ und
  setze $A[h_i(s)] := 1$.

Abfragen der "Skizze":

Bei Eingabe eines $u \in U$:

- berechne $h_1(u), \ldots, h_k(u)$
- Teste, ob $A[h_i(u)] = 1$ für <u>alle</u> $i \in \{1, \ldots, k\}$ gilt.
- Falls "ja": gib aus "u gehört möglicherweise zu S"
  Sonst: gib aus "u gehört nicht zu S".

Man sieht leicht, dass die Ausgabe korrekt ist.

Frage: Wie sollen wir k wählen?

Anschaulich scheint klar zu sein:

- Je kleiner k, desto weniger Array-Einträge sind auf 1 gesetzt, und desto geringer scheint die Wahrscheinlichkeit dafür zu sein, dass für ein $u \notin S$ die Array-Einträge an allen Positionen $h_1(u), \ldots, h_k(u)$ auf 1 gesetzt sind (und daher eine "falsche positive Antwort" gegeben wird).
- Je größer k, desto geringer scheint die Wahrscheinlichkeit dafür zu sein, dass für ein $u \notin S$ die Array-Einträge an allen Positionen $h_1(u), \ldots, h_k(u)$ auf 1 gesetzt sind. (und daher eine "falsche positive Antwort" gegeben wird).

Was ist also ein guter Kompromiss zwischen "kleinem k" und "großem k"?

Ziel: Für festes n und N wähle k so, dass die erwartete Anzahl falscher positiver Antworten möglichst klein ist.

Um die Analyse zu ermöglichen gehen wir im Folgenden davon aus, dass die von uns genutzten Hash-Funktionen die Werte aus U zufällig, gleichverteilt und unabhängig voneinander auf die Bits $1,\dots,n$ verteilen.

(In der Praxis genutzte Hash-Funktionen werden diese idealisierte Eigenschaft i.d.R. nicht haben; unsere Analyse wird dadurch aber überhaupt erst möglich. Bisher sind keine Ergebnisse bekannt, die theoretische Garantien für Bloom-Filter geben, wenn (streng) k-universelle Familien von Hash-Funktionen genutzt werden.

Beobachtung 1:

Nach dem Einfügen von S ins Array $A[1..n]$ gilt für jedes feste Bit $j \in \{1,\dots,n\}$:

$$\Pr\left(A[j]=0\right) = \left(1-\frac{1}{n}\right)^{k\cdot N}$$

Zur Erinnerung: $N = |S|$

Die Wahrscheinlichkeit läuft hier darüber, dass die Hash-Funktionen $h_1,\dots,h_k$ zufällig und unabhängig voneinander gewählt werden.

Begründung: Es gilt: $A[j]=0 \iff \forall s\in S$ gilt: $h_1(s)\neq j$ und $\cdots$ und $h_k(s)\neq j$.

Für jedes $s\in S$ und jedes $i\in\{1,\dots,k\}$ ist

$$\Pr\left(h_i(s)=j\right) = \frac{1}{n} \quad \text{und daher} \quad \Pr\left(h_i(s)\neq j\right) = 1-\frac{1}{n}.$$

Also gilt für jedes feste $s\in S$:

$$\Pr\left(h_1(s)\neq j \text{ und} \cdots \text{und } h_k(s)\neq j\right) = \left(1-\frac{1}{n}\right)^{k}$$

$h_1,\dots,h_k$ sind unabhängig voneinander gewählt

Auf Grund unserer idealisierenden Annahme, dass die Hash-Funktionen die Werte aus U zufällig, gleichverteilt und <u>unabhängig</u> voneinander auf die Bits $1, \dots, n$ verteilen, erhalten wir:

$$\Pr\Big(\forall s \in S \text{ gilt: } h_1(s) \neq j \text{ und } \dots \text{ und } h_k(s) \neq j\Big) = \left(1-\frac{1}{n}\right)^{k \cdot |S|}$$

Somit gilt: $\Pr\big(A[j]=0\big) = \left(1-\frac{1}{n}\right)^{k \cdot N}$

□ Beobachtung 1

Wir setzen $\boxed{p(k) := \left(1-\frac{1}{n}\right)^{k\cdot N} \underset{\text{Beob.1}}{=} \Pr\big(A[j]=0\big)}$ (für jedes feste $j \in \{1..n\}$)

Wir gehen im Folgenden vereinfachend davon aus, dass für paarweise verschiedene Werte $j_1, \dots, j_k \in \{1, \dots, n\}$ die Ereignisse "$A[j_1] \neq 0$", ..., "$A[j_k] \neq 0$" unabhängig voneinander sind. (Unter Verwendung der Chernoff-Schranke kann man zeigen, dass diese Annahme das Ergebnis nicht allzu sehr verfälscht.)

<u>Beobachtung 2</u>:

Für jedes feste $u \in U \setminus S$ gilt:

$$\Pr\Big(\underbrace{\text{"u liefert eine falsche positive Antwort"}}_{\text{d.h. } A[h_i(u)]=1 \text{ für alle } i \in \{1,\dots,k\}}\Big) = \big(1-p(k)\big)^k$$

<u>Begründung</u>: Für jedes $i \in \{1, \dots, k\}$ setze $j_i := h_i(u)$. Gemäß Beobachtung 1 gilt $\Pr\big(A[j_i]=1\big) = 1-p(k)$.

Auf Grund unserer vereinfachenden Annahme, dass die Ereignisse "$A[j_1] \neq 0$", …, "$A[j_k] \neq 0$" alle unabhängig voneinander sind, erhalten wir:

$$\Pr\left(\text{f.a. } i \in \{1,..,k\} \text{ gilt: } A[j_i] = 1\right) = \left(1 - p(k)\right)^k$$

$\square$ Beobachtung 2

<u>Ziel:</u> Wähle k so, dass $\left(1-p(k)\right)^k$ möglichst klein ist.

<u>Lösungsansatz:</u>
"Kurvendiskussion" für die Funktion
$\tilde{f} : \mathbb{R} \to \mathbb{R}$ mit

$$\tilde{f}(x) := \left(1 - p(x)\right)^x = \left(1 - \left(1-\frac{1}{n}\right)^{N \cdot x}\right)^x$$

<u>Typischer Trick</u>, um die Rechnung zu vereinfachen, ohne das Ergebnis allzusehr zu verfälschen:

$$\boxed{\text{Ersetze } \left(1-\frac{1}{n}\right) \text{ durch } e^{-\frac{1}{n}}.}$$

<u>Rechtfertigung:</u> Es ist bekannt, dass

$$e = \lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n \quad \text{und} \quad e^{-1} = \lim_{n\to\infty}\left(1-\frac{1}{n}\right)^n$$

Für hinreichend große n ist also

$$e \approx \left(1+\frac{1}{n}\right)^n \quad \text{und} \quad e^{-1} \approx \left(1-\frac{1}{n}\right)^n$$

Anstatt $\tilde{f}(x) := \Big(1 - \underbrace{\left(1-\frac{1}{n}\right)}_{\approx e^{-\frac{1}{n}}}{}^{N\cdot x}\Big)^x$ betrachten

wir daher die Funktion $f: \mathbb{R} \to \mathbb{R}$ mit

$$f(x) := \left(1 - e^{-\frac{N}{n}\cdot x}\right)^x$$

Als unser gesuchtes k wählen wir dasjenige x, für das $f(x)$ kleinstmöglich ist.

<u>klar</u>: $f(x)$ minimal $\Longleftrightarrow$ $g(x) := \ln(f(x))$ minimal.

Wir minimieren also

$$g(x) := \ln(f(x)) = x \cdot \ln\left(1 - e^{-\frac{N}{n}\cdot x}\right).$$

Dazu suchen wir Nullstellen der ersten Ableitung von $g(x)$, also der Funktion

$$g'(x) \underset{\text{Standard-Ableitungsregeln}}{=} \ln\left(1 - e^{-\frac{N}{n}\cdot x}\right) + \frac{N}{n}\cdot x \cdot \frac{e^{-\frac{N}{n}\cdot x}}{1 - e^{-\frac{N}{n}\cdot x}}$$

Man kann (leicht) nachprüfen, dass Folgendes gilt:

1) $g'(x) = 0 \Longleftrightarrow x = (\ln 2)\cdot\frac{n}{N}$, und

2) Der Wert $x = (\ln 2)\cdot\frac{n}{N}$ liefert ein globales Minimum der Funktion $g(x)$ und somit auch ein globales Minimum der Funktion $f(x)$.

(6)

Wir wählen also $\boxed{k := \left\lceil \underbrace{(\ln 2)}_{\approx 0{,}7} \cdot \frac{n}{N} \right\rceil} \approx 0{,}7 \cdot \frac{n}{N}$

Für jedes feste $u \in U \setminus S$ gilt dann:

$$\Pr\left(u \text{ liefert eine falsche positive Antwort}\right)$$

$$\underset{\text{Beob. 2}}{=} \left(1 - p(k)\right)^k$$

$$= \tilde{f}(k)$$

$$\approx f(k)$$

$$\approx f\left((\ln 2) \cdot \frac{n}{N}\right)$$

$$= \Big(1 - \underbrace{e^{-\frac{N}{n} \cdot (\ln 2) \cdot \frac{n}{N}}}_{= e^{-\ln 2} = \frac{1}{e^{\ln 2}} = \frac{1}{2}}\Big)^{(\ln 2) \cdot \frac{n}{N}}$$

$$= \left(\frac{1}{2}\right)^{(\ln 2) \cdot \frac{n}{N}}$$

Wegen $\left(\frac{1}{2}\right)^{\ln 2} \approx 0{,}6185$ gilt mit unserer Wahl von k also:

$$\Pr\left(\begin{array}{c} u \text{ liefert eine} \\ \text{falsche positive Antwort} \end{array}\right) \approx \left(0{,}6185\right)^{\frac{n}{N}}$$

Es gilt:

$y := \frac{n}{N}$	$\frac{1}{2}$	1	2	3	4	5	...	8	...	16
$0{,}6185^y$	$\approx 0{,}786 \approx 79\%$	$0{,}6185 \approx 62\%$	$\approx 0{,}3825 \approx 38\%$	$\approx 0{,}2366 \approx 24\%$	$\approx 0{,}1463 \approx 15\%$	$\approx 0{,}0905 \approx 9\%$	...	$\approx 0{,}0214 \approx 2\%$	...	$\approx 0{,}00045 \approx 0{,}05\%$

Insbesondere:

Um für ein gegebenes S der Größe N eine Fehlerwahrscheinlichkeit von $\approx 2\%$ zu bekommen, sollte man ein Bit-Array der Länge $n = 8 \cdot N$ verwenden und $k := \lceil (\ln 2) \cdot \frac{n}{N} \rceil = \lceil 8 \cdot \ln 2 \rceil = 6$ Hash-Funktionen nutzen.

Wenn man stattdessen ein Bit-Array der Länge $n = 16 \cdot N$ verwendet und $k := \lceil (\ln 2) \cdot \frac{n}{N} \rceil = \lceil 16 \cdot \ln 2 \rceil = 12$ Hash-Funktionen nutzt, verringert sich die Fehlerwahrscheinlichkeit sogar auf $\approx 0{,}05\%$.

Zurück zu unserem Anwendungsbeispiel:

Hier ist S eine "white list", die aus $N := 10^9$ Gmail-Adressen besteht, wobei zur Speicherung jeder einzelnen Gmail-Adresse ca 20 Bytes nötig sind.
Zur exakten Speicherung von S benötigen wir also $20 \cdot 10^9$ Byte $= 20$ GB.

Wenn wir stattdessen einen Bloom-Filter mit einem Bit-Array der Länge $n = 8 \cdot N$ und $k = 6$ Hash-Funktionen verwenden, erhalten wir falsche positive Antworten mit ca 2%.

Zur Speicherung des Arrays $A[1 \dots n]$ benötigen wir „nur" $8 \cdot N$ Bits $= 8 \cdot 10^9$ Bits $= 1 \cdot 10^9$ Bytes $= 1$ GB.