Finding Similar Items

Hung Le

University of Victoria

January 11, 2019

Misc

- Programming languages: Python 3 is a default choice. If you are not comfortable with Python, you can use one of three languages: C, C++ or Java. However, labs and homework instructions will use Python 3.
- Since projects for SENG 474 and CSC 578D have different expectations, a SENG student should NOT be in the same group with a CSC student for the project. However, if you are a SENG student and insist on being in a group with CSC students, then the expectation for the project is CSC.
- You can pair with different students on different homeworks. Make sure you have the names of all the students in your group on paper.

Problem

Finding similar items

Given a collection of items $\mathcal{I} = \{I_1, I_2, \ldots, I_m\}$ and an arbitrary item I, find items in $\mathcal{I}$ that are similar to I.

Questions that we explore:

- What does it mean for two items to be similar?
- How can we (quickly) find similar items?

Applications

Question suggestion in Quora.

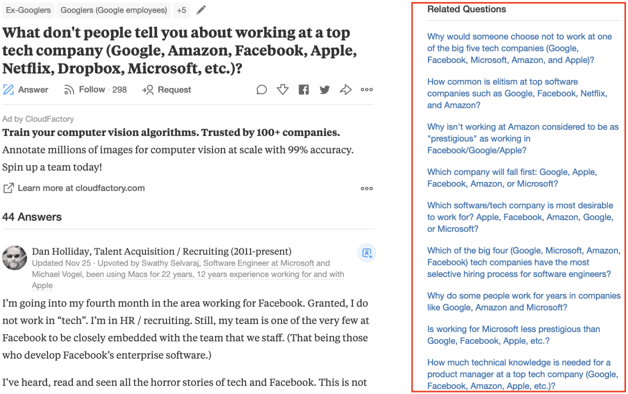

Figure: Quora

Applications

Recommendation system

Figure: Amazon

Applications

Amazon Recommendation System in a 2003 paper[1] (cited more than 5000 times according to Google scholar).

AMAZONALG(a customer C who bought item I_1)
    foreach item I_2
        Compute the similarity between I_1 and I_2
    Recommend most similar items of I_1 to C.

Figure: Amazon recommedation system

[1] https://www.cs.umd.edu/~samir/498/Amazon-Recommendations.pdf

Applications

Many other applications.

- Web page deduplication.
- Plagiarism detection.
- News deduplication.
- And more.

Back to our problem

Finding similar items

Given a collection of items $\mathcal{I} = \{I_1, I_2, \ldots, I_m\}$ and an arbitrary item I, find items in $\mathcal{I}$ that are similar to I.

Each item is typically a set of elements drawn from a ground set U.

- Each document is a set of words. U here is a set of all words that appears in the collection of documents.
- Each Amazon item can be represented by a set of customers who bought the item. U here is a set of all customers who ever bought something on Amazon.

Jaccard similarity measure for sets

Jaccard similarity

$$\texttt{Sim}(I_1, I_2) = \frac{|I_1 \cap I_2|}{I_1 \cup I_2} \tag{1}$$

For example, $I_1 = \{a, b, d\}$ and $I_2 = \{a, d, e\}$. Then $I_1 \cap I_2 = \{a, d\}$ and $I_1 \cup I_2 = \{a, b, d, e\}$. Thus:

$$\texttt{Sim}(I_1, I_2) = \frac{|\{a, d\}|}{|\{a, b, d, e\}|} = \frac{2}{4} = \frac{1}{2}$$

Jaccard similarity measure for sets

Jaccard similarity

$$\texttt{Sim}(I_1, I_2) = \frac{|I_1 \cap I_2|}{I_1 \cup I_2} \tag{1}$$

For example, $I_1 = \{a, b, d\}$ and $I_2 = \{a, d, e\}$. Then $I_1 \cap I_2 = \{a, d\}$ and $I_1 \cup I_2 = \{a, b, d, e\}$. Thus:

$$\texttt{Sim}(I_1, I_2) = \frac{|\{a, d\}|}{|\{a, b, d, e\}|} = \frac{2}{4} = \frac{1}{2}$$

How to calculate Jaccard similarity efficiently?

Jaccard similarity measure for sets

Jaccard similarity

$$\texttt{Sim}(I_1, I_2) = \frac{|I_1 \cap I_2|}{I_1 \cup I_2} \tag{1}$$

For example, $I_1 = \{a, b, d\}$ and $I_2 = \{a, d, e\}$. Then $I_1 \cap I_2 = \{a, d\}$ and $I_1 \cup I_2 = \{a, b, d, e\}$. Thus:

$$\texttt{Sim}(I_1, I_2) = \frac{|\{a, d\}|}{|\{a, b, d, e\}|} = \frac{2}{4} = \frac{1}{2}$$

How to calculate Jaccard similarity efficiently?

- We can compute Jaccard similarity in $O(|I_1| + |I_2|)$ time using hashing. See the board calculation.

Upon having a similarity measure

Finding similar items

Given a collection of items $\mathcal{I} = \{I_1, I_2, \ldots, I_m\}$ and an arbitrary item I_1, find all items I_2 in $\mathcal{I}$ such that $\texttt{Sim}(I_1, I_2) \geq x$ for some fixed threshold $x < 1$.

We can do like Amazon

$\textsc{FindSim}$(item I_1, threshold x)
 foreach item I_2
  Compute $\texttt{Sim}(I_1, I_2)$.
  Report I_2 if $\texttt{Sim}(I_1, I_2) \geq x$.

We can do like Amazon

$\text{FINDSIM}(\text{item } I_1, \text{threshold } x)$
  foreach item I_2
    Compute $\texttt{Sim}(I_1, I_2)$.
    Report I_2 if $\texttt{Sim}(I_1, I_2) \geq x$.

- The worst case running time is $O(m^2 n)$ where $m = |\mathcal{I}|$ and $n = |U|$ if you want to find similar items for **all** items. See the board calculation.

We can do like Amazon

$\textsc{FindSim}$(item I_1, threshold x)
  foreach item I_2
    Compute $\texttt{Sim}(I_1, I_2)$.
    Report I_2 if $\texttt{Sim}(I_1, I_2) \geq x$.

- The worst case running time is $O(m^2 n)$ where $m = |\mathcal{I}|$ and $n = |U|$ if you want to find similar items for **all** items. See the board calculation.
- Typically m, n is about $10M - 100M$ for a big system like Amazon, so the algorithm is terribly slow.

If you are curious what does Amazon do

They employed the following heuristics:

- Compute the similarity table offline: "This offline computation of the similar-items table is extremely time intensive, with $O(N^2M)$ as worst case. In practice, however, it's closer to $O(NM)$, as most customers have very few purchases."
- Online phase: only loop through items I_2 bought by customers who already bought I_1.
- Sampling: 'Sampling customers who purchase best-selling titles reduces run-time even further, with little reduction in quality."

This lecture

Finding similar items

Given a collection of items $\mathcal{I} = \{I_1, I_2, \ldots, I_m\}$ and an arbitrary item I_1, find all items I_2 in $\mathcal{I}$ such that $\texttt{Sim}(I_1, I_2) \geq x$ for some fixed threshold $x \leq 1$.

W will study a data structure $\mathcal{D}$ where:

- $\mathcal{D}$ can be built in $O(n + m)$ time (best case).
- For any item I_1, similar items can be retrieved by querying data structure $\mathcal{D}$ in times $O(1)$ per similar item. (If I_1 has p similar items, the querying time is $O(p)$.)

Let's simplify the problem a bit

Finding similar items

Given a collection of items $\mathcal{I} = \{I_1, I_2, \ldots, I_m\}$ and an arbitrary item I_1, find all items I_2 in $\mathcal{I}$ such that $\texttt{Sim}(I_1, I_2) = 1$.

Can you build the data structure $\mathcal{D}$?

Let's simplify the problem a bit

Finding similar items

Given a collection of items $\mathcal{I} = \{I_1, I_2, \ldots, I_m\}$ and an arbitrary item I_1, find all items I_2 in $\mathcal{I}$ such that $\texttt{Sim}(I_1, I_2) = 1$.

Can you build the data structure $\mathcal{D}$?

- $\texttt{Sim}(I_1, I_2) = 1$ if and only if $I_1 = I_2$ (proof?)

Let's simplify the problem a bit

Finding similar items

Given a collection of items $\mathcal{I} = \{I_1, I_2, \ldots, I_m\}$ and an arbitrary item I_1, find all items I_2 in $\mathcal{I}$ such that $\texttt{Sim}(I_1, I_2) = 1$.

Can you build the data structure $\mathcal{D}$?

- $\texttt{Sim}(I_1, I_2) = 1$ if and only if $I_1 = I_2$ (proof?)
- A hash table would work!

The general problem

Finding similar items

Given a collection of items $\mathcal{I} = \{I_1, I_2, \ldots, I_m\}$ and an arbitrary item I_1, find all items I_2 in $\mathcal{I}$ such that $\texttt{Sim}(I_1, I_2) \geq x$ for $x < 1$.

Idea: design a hash function $h(.)$ such that:

- $h(I_1) = h(I_2)$ when $\texttt{Sim}(I_1, I_2) \geq x$ (w.h.p[2]).
- $h(I_1) \neq h(I_2)$ when $\texttt{Sim}(I_1, I_2) < x$ (w.h.p) .

[2] w.h.p = with high probability

The general problem

Finding similar items

Given a collection of items $\mathcal{I} = \{I_1, I_2, \ldots, I_m\}$ and an arbitrary item I_1, find all items I_2 in $\mathcal{I}$ such that $\texttt{Sim}(I_1, I_2) \geq x$ for $x < 1$.

Idea: design a hash function $h(.)$ such that:

- $h(I_1) = h(I_2)$ when $\texttt{Sim}(I_1, I_2) \geq x$ (w.h.p[2]).
- $h(I_1) \neq h(I_2)$ when $\texttt{Sim}(I_1, I_2) < x$ (w.h.p) .

But, similarity is not transitive relation.

- $I_1 = \{a, b, d\}, I_2 = \{b, d, e\}, I_3 = \{d, e, f\}$ and $x = 0.5$. $\texttt{Sim}(I_1, I_2) \geq x, \texttt{Sim}(I_2, I_3) \geq x$ but $\texttt{Sim}(I_1, I_3) < x$.

[2] w.h.p = with high probability

The general problem

Finding similar items

Given a collection of items $\mathcal{I} = \{I_1, I_2, \ldots, I_m\}$ and an arbitrary item I_1, find all items I_2 in $\mathcal{I}$ such that $\texttt{Sim}(I_1, I_2) \geq x$ for $x < 1$.

Idea: design a hash function $h(.)$ such that:

- $h(I_1) = h(I_2)$ when $\texttt{Sim}(I_1, I_2) \geq x$ (w.h.p[2]).
- $h(I_1) \neq h(I_2)$ when $\texttt{Sim}(I_1, I_2) < x$ (w.h.p) .

But, similarity is not transitive relation.

- $I_1 = \{a, b, d\}, I_2 = \{b, d, e\}, I_3 = \{d, e, f\}$ and $x = 0.5$.
  $\texttt{Sim}(I_1, I_2) \geq x, \texttt{Sim}(I_2, I_3) \geq x$ but $\texttt{Sim}(I_1, I_3) < x$.

It suggests that we may need to use many hash tables to have a robust data structure. (We will go into details of this point later.)

[2] w.h.p = with high probability

Overview of the data structure

BUILD($\mathcal{D}$)
    $\mathcal{D} \leftarrow \{H_1, H_2, \ldots, H_b\}$ `// use` b `hash tables`
    foreach hash table H_i
        Let $h_i(.)$ be the corresponding hash function.
        foreach item $I \in \mathcal{I}$
            $j \leftarrow h_i(I)$
            $H_i[j] \leftarrow H_i[j] \cup \{I\}$ `// put item` I `to location` j

Overview of the data structure

$\text{BUILD}(\mathcal{D})$

$\mathcal{D} \leftarrow \{H_1, H_2, \ldots, H_b\}$ `// use` b `hash tables`

foreach hash table H_i

Let $h_i(.)$ be the corresponding hash function.

foreach item $I \in \mathcal{I}$

$j \leftarrow h_i(I)$

$H_i[j] \leftarrow H_i[j] \cup \{I\}$ `// put item` I `to location` j

Running time? $O(b \times m \times \{\text{hashing time per item}\})$.

- Typically use $b \in [20, 128]$ (will see later).
- The hashing time is proportional to the size of items I. In practice, $|I|$ is roughly a small order of 100.

So the (best case) running time is $O(n + m)$, (recall $n = |U|$)

Overview of the data structure

FindSim($\mathcal{D}$, item I_1)
$\mathcal{S} \leftarrow \emptyset$ `// set of similar items to` I_1
foreach hash table H_i
Let $h_i(.)$ be the corresponding hash function.
$j \leftarrow h_i(I_1)$
$\mathcal{S} \leftarrow \mathcal{S} \cup H_i[j]$ `// collect all items in location` j
return $\mathcal{S}$

Overview of the data structure

FindSim($\mathcal{D}$, item I_1)
  $\mathcal{S} \leftarrow \emptyset$ `// set of similar items to` I_1
  foreach hash table H_i
    Let $h_i(.)$ be the corresponding hash function.
    $j \leftarrow h_i(I_1)$
    $\mathcal{S} \leftarrow \mathcal{S} \cup H_i[j]$ `// collect all items in location` j
  return $\mathcal{S}$

Running time? $O(b \times \{\text{hashing time of } I\} + |\texttt{SimSet}(I)|)$ where $\texttt{SimSet}(I_1)$ is the set of all items similar to I_1.

- Typically use $b \in [20, 128]$ (will see later).
- The hashing time is proportional to the size of items I. In practice, $|I|$ is roughly a small order of 100.

So the best case running time is $O(|\texttt{SimSet}(I_1)|)$.

Are we done?

Design a hash function $h(.)$ such that:

- $h(I_1) = h(I_2)$ when $\texttt{Sim}(I_1, I_2) \geq x$ (w.h.p).
- $h(I_1) \neq h(I_2)$ when $\texttt{Sim}(I_1, I_2) < x$ (w.h.p) .

Let's have some fun with probability!

Random Permutation

- Pick a random permutation π of elements in the ground set U.
- Let $h_\pi(I) =$ minimum index of elements of I in π.

Random Permutation

- Pick a random permutation π of elements in the ground set U.
- Let $h_\pi(I) =$ minimum index of elements of I in π.

For example: $U = \{a, b, c, d, e, f\}$, and $\pi = 2, 5, 1, 4, 6, 3$, that is $\pi[a] = 2, \pi[b] = 4, \ldots, \pi[f] = 6$. Then what is $h(I)$ for $I = \{a, d, f\}$

Random Permutation

- Pick a random permutation π of elements in the ground set U.
- Let $h_\pi(I) =$ minimum index of elements of I in π.

For example: $U = \{a, b, c, d, e, f\}$, and $\pi = 2, 5, 1, 4, 6, 3$, that is $\pi[a] = 2, \pi[b] = 4, \ldots, \pi[f] = 6$. Then what is $h(I)$ for $I = \{a, d, f\}$

- $h(I) = \min(2, 1, 6) = 1$

Random Permutation

- Pick a random permutation π of elements in the ground set U.
- Let $h_\pi(I)$ = minimum index of elements of I in π.

Lemma

$$\Pr[h_\pi(I_1) = h_\pi(I_2)] = \frac{|I_1 \cap I_2|}{|I_1 \cup I_2|} = \mathtt{Sim}(I_1, I_2). \tag{2}$$

Random Permutation

- Pick a random permutation π of elements in the ground set U.
- Let $h_\pi(I)$ = minimum index of elements of I in π.

Lemma

$$\Pr[h_\pi(I_1) = h_\pi(I_2)] = \frac{|I_1 \cap I_2|}{|I_1 \cup I_2|} = \mathtt{Sim}(I_1, I_2). \quad (2)$$

Question: when $I_1 = I_2$, is the lemma true?

Random Permutation

- Pick a random permutation π of elements in the ground set U.
- Let $h_\pi(I)$ = minimum index of elements of I in π.

Lemma

$$\Pr[h_\pi(I_1) = h_\pi(I_2)] = \frac{|I_1 \cap I_2|}{|I_1 \cup I_2|} = \mathtt{Sim}(I_1, I_2). \tag{2}$$

Question: when $I_1 = I_2$, is the lemma true?

Proof.

Idea: for any subset $X \subseteq U$, the probability that a particular element $x \in X$ has minimum index in π is $\frac{1}{|X|}$. □

Minhashing[3]

- Pick a random permutation π of elements in the ground set U.
- Let $h_\pi(I) =$ minimum index of elements of I in π.

Lemma

$$\Pr[h_\pi(I_1) = h_\pi(I_2)] = \frac{|I_1 \cap I_2|}{|I_1 \cup I_2|} = \mathtt{Sim}(I_1, I_2). \tag{3}$$

What we get is:

- $\Pr[\mathcal{D}(I_1) = \mathcal{D}(I_2)]$ is at least $1 - (1-x)^b$ if $\mathtt{Sim}(I_1, I_2) \geq x$.
- $\Pr[\mathcal{D}(I_1) \neq \mathcal{D}(I_2)]$ is at least $(1-x)^b$ if $\mathtt{Sim}(I_1, I_2) < x$.

Recall we use b hash functions in $\mathcal{D}$ and by $\mathcal{D}(I_1) = \mathcal{D}(I_2)$, we means I_1 and I_2 are hashed to the same location in at least one hash table in $\mathcal{D}$.

[3] By Brorder, see `https://www.cs.princeton.edu/courses/archive/spring13/cos598C/broder97resemblance.pdf`

Still not done yet

- $\Pr[\mathcal{D}(I_1) = \mathcal{D}(I_2)]$ is at least $1 - (1-x)^b$ if $\texttt{Sim}(I_1, I_2) \geq x$.
- $\Pr[\mathcal{D}(I_1) \neq \mathcal{D}(I_2)]$ is at least $(1-x)^b$ if $\texttt{Sim}(I_1, I_2) < x \Rightarrow$ too many false positives.

Still not done yet

- $\Pr[\mathcal{D}(I_1) = \mathcal{D}(I_2)]$ is at least $1 - (1-x)^b$ if $\texttt{Sim}(I_1, I_2) \geq x$.
- $\Pr[\mathcal{D}(I_1) \neq \mathcal{D}(I_2)]$ is at least $(1-x)^b$ if $\texttt{Sim}(I_1, I_2) < x \Rightarrow$ too many false positives.

For example: when $b = 10, x = 0.5$ then $(1-x)^b \sim \frac{1}{1000}$.

MinHash Signature

- Pick r random permutations $\pi_1, \pi_2, \ldots, \pi_r$ of elements in the ground set U.
- Let $\sigma_i(I) =$ minimum index of elements of I in π_i for $i = 1, 2, \ldots, r$.
- $h(I) =< \sigma_1(I), \sigma_2(I), \ldots, \sigma_r(I) >$. This is called a *MinHash* signature of I.
  - $h(I_1) = h(I_2)$ if and only if $\sigma_i(I_1) = \sigma_i(I_2)$ for all $i = 1, 2 \ldots, r$.

MinHash Signature

- Pick r random permutations $\pi_1, \pi_2, \ldots, \pi_r$ of elements in the ground set U.
- Let $\sigma_i(I) =$ minimum index of elements of I in π_i for $i = 1, 2, \ldots, r$.
- $h(I) =< \sigma_1(I), \sigma_2(I), \ldots, \sigma_r(I) >$. This is called a *MinHash* signature of I.
    - $h(I_1) = h(I_2)$ if and only if $\sigma_i(I_1) = \sigma_i(I_2)$ for all $i = 1, 2 \ldots, r$.

Lemma

$\Pr[h(I_1) = h(I_2)] = (\mathtt{Sim}(I_1, I_2))^r$.

Proof.

Your exercise!! □

MinHash Signature

Lemma

$\Pr[h(I_1) = h(I_2)] = (\mathtt{Sim}(I_1, I_2))^r$.

What we get is:

- $\Pr[\mathcal{D}(I_1) = \mathcal{D}(I_2)]$ is at least $1 - (1 - x^r)^b$ if $\mathtt{Sim}(I_1, I_2) \geq x$.
- $\Pr[\mathcal{D}(I_1) \neq \mathcal{D}(I_2)]$ is at least $(1 - x^r)^b$ if $\mathtt{Sim}(I_1, I_2) < x$.

No what?

MinHash Signature

- $\Pr[\mathcal{D}(I_1) = \mathcal{D}(I_2)]$ is at least $1 - (1 - x^r)^b$ if $\texttt{Sim}(I_1, I_2) \geq x$.
- $\Pr[\mathcal{D}(I_1) \neq \mathcal{D}(I_2)]$ is at least $(1 - x^r)^b$ if $\texttt{Sim}(I_1, I_2) < x$.

Draw the graph $y = 1 - 1(-x^r)^b$ [4].

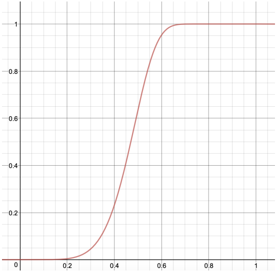

Figure: Graph for $y = 1 - (1 - x^6)^{64}$. Here $r = 6$ and $b = 64$.

[4] Go to `https://www.desmos.com/calculator` to play with drawing

MinHash Signature

How to choose r, b? Fix x then choose r, b such that $(1 - x^r)^b \sim 1/2$. For example, when $x = 0.5$, r and b would (approximately) be such that $b = 2^r$. (That's why I choose $r = 6$ and $b = 64$ in the graph.)

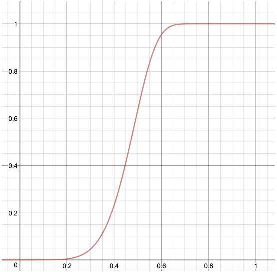

Figure: Graph for $y = 1 - (1 - x^6)^{64}$. Here $r = 6$ and $b = 64$.

Your exercise: if $x = 0.25$, how b and r would look like?

Some implementation issues

Q: How can I pick a random permutation?
A: Actually you don't. Choose a hash function that hash each element of U to a 32-bit number. Then the hash code of each item is the minimum hash code among all elements in I.
If you just want to do some experiments, hash functions like $h(x) = (ax \mod p) \mod n$ where p is a prime number bigger than n and a is chosen randomly from $[1, p-1]$ are reasonable.
If you build something serious, the tabulation hashing [5] is a good choice.

[5] `https://en.wikipedia.org/wiki/Tabulation_hashing`

Some implementation issues

Q: We talk about signatures in the lecture (which are a concatenation of integers). However, to use data structure $\mathcal{D}$, you need to have the output of the hash function to be a small numbers to index the hash tables. How could that be done?
A: Use a standard hash function to map to a range of small integers for indexing the hash table. But be careful in choosing the range (or load factor) so that there are not many collisions.

Cosine similarity

Cosine similarity

For two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$:

$$\texttt{Cosine}(\mathbf{x}, \mathbf{y}) = \frac{\sum_{i=1}^{d} x[i]y[i]}{\sqrt{\sum_{i=1}^{n} x[i]^2} \cdot \sqrt{\sum_{i=1}^{n} y[i]^2}} \tag{4}$$

- Require to represent each item as a vector. For document, TF-IDF is a popular choice.
- A hashing version for similarity search for (a closely related version of) cosine similarity, called SimHash[6], exists.

[6] `https://www.cs.princeton.edu/courses/archive/spr04/cos598B/bib/CharikarEstim.pdf`