Finding similar items I

Krzysztof Dembczyński

Intelligent Decision Support Systems Laboratory (IDSS)
Poznań University of Technology, Poland

Bachelor studies, eighth semester
Academic year 2018/19 (summer semester)

Review of the previous lectures

- Processing of massive datasets
- Evolution of database systems
- OLTP and OLAP systems
- ETL
- Dimensional modeling
- Data processing
- MapReduce in Spark
- Approximate query processing

Outline

Outline

Nearest neighbor search

- Find similar elements to the query element.

Applications of nearest neighbor search

- Similarity of documents
  - Plagiarism
  - Mirror pages
  - Articles from the same source
- Machine learning
  - k-nearest neighbors
  - Collaborative filtering
- Computational geometry
- Computer vision
- Geographic Information Systems (GIS)

Nearest neighbor search

- Brute force search:

Nearest neighbor search

- Brute force search:
  - Given a query point q scan through each of n data points in database

Nearest neighbor search

- Brute force search:
  - Given a query point q scan through each of n data points in database
  - Computational complexity for 1-NN query:

Nearest neighbor search

- Brute force search:
  - Given a query point q scan through each of n data points in database
  - Computational complexity for 1-NN query: $\mathcal{O}(n)$.

Nearest neighbor search

- Brute force search:
  - Given a query point q scan through each of n data points in database
  - Computational complexity for 1-NN query: $\mathcal{O}(n)$.
  - Computational complexity for k-NN query:

Nearest neighbor search

- Brute force search:
  - Given a query point q scan through each of n data points in database
  - Computational complexity for 1-NN query: $\mathcal{O}(n)$.
  - Computational complexity for k-NN query: $\mathcal{O}(n \log k)$ or

Nearest neighbor search

- Brute force search:
  - Given a query point q scan through each of n data points in database
  - Computational complexity for 1-NN query: $\mathcal{O}(n)$.
  - Computational complexity for k-NN query: $\mathcal{O}(n \log k)$ or $\mathcal{O}(n + k)$

Nearest neighbor search

- Brute force search:
  - Given a query point q scan through each of n data points in database
  - Computational complexity for 1-NN query: $\mathcal{O}(n)$.
  - Computational complexity for k-NN query: $\mathcal{O}(n \log k)$ or $\mathcal{O}(n + k)$
- With large databases linear complexity can be too costly.

Nearest neighbor search

- Brute force search:
  - Given a query point q scan through each of n data points in database
  - Computational complexity for 1-NN query: $\mathcal{O}(n)$.
  - Computational complexity for k-NN query: $\mathcal{O}(n \log k)$ or $\mathcal{O}(n + k)$
- With large databases linear complexity can be too costly.
- Can we do better?

Nearest neighbor search

- Brute force search:
  - Given a query point q scan through each of n data points in database
  - Computational complexity for 1-NN query: $\mathcal{O}(n)$.
  - Computational complexity for k-NN query: $\mathcal{O}(n \log k)$ or $\mathcal{O}(n + k)$
- With large databases linear complexity can be too costly.
- Can we do better?
- Data structures for exact search: not robust to curse of dimensionality

Nearest neighbor search

- Brute force search:
  - Given a query point q scan through each of n data points in database
  - Computational complexity for 1-NN query: $\mathcal{O}(n)$.
  - Computational complexity for k-NN query: $\mathcal{O}(n \log k)$ or $\mathcal{O}(n + k)$
- With large databases linear complexity can be too costly.
- Can we do better?
- Data structures for exact search: not robust to curse of dimensionality
- Approximate algorithms

Outline

Motivation

- Consider an application of finding near-duplicates of Web pages, like plagiarisms or mirrors.

Motivation

- Consider an application of finding near-duplicates of Web pages, like plagiarisms or mirrors.
- We can represents pages as sets of character k-grams (or k-shingles) and formulate a problem as finding sets with a relatively large intersection.

Motivation

- Consider an application of finding near-duplicates of Web pages, like plagiarisms or mirrors.
- We can represents pages as sets of character k-grams (or k-shingles) and formulate a problem as finding sets with a relatively large intersection.
- Storing large number of sets and computing their similarity in naive way is not sufficient.

Motivation

- Consider an application of finding near-duplicates of Web pages, like plagiarisms or mirrors.
- We can represents pages as sets of character k-grams (or k-shingles) and formulate a problem as finding sets with a relatively large intersection.
- Storing large number of sets and computing their similarity in naive way is not sufficient.
- We compress sets in a way that enables to deduce the similarity of the underlying sets from their compressed versions.

Jaccard similarity

- We focus on similarity of sets by looking at the relative size of their intersection.

Jaccard similarity

- We focus on similarity of sets by looking at the relative size of their intersection.
- The Jaccard similarity of sets S and T is defined as:

$$SIM(S,T) = \frac{|S \cap T|}{|S \cup T|}$$

Jaccard similarity

- We focus on similarity of sets by looking at the relative size of their intersection.
- The Jaccard similarity of sets S and T is defined as:

$$SIM(S,T) = \frac{|S \cap T|}{|S \cup T|}$$

- **Example**: Let $S = \{a, b, c, d\}$ and $T = \{c, d, e, f\}$, then

$$SIM(S,T) = 2/6.$$

k-shingles

- A document is a string of characters.

k-shingles

- A document is a string of characters.
- A k-shingle (or k-gram) for a document is any substring of length k found within the document.

k-shingles

- A document is a string of characters.
- A k-shingle (or k-gram) for a document is any substring of length k found within the document.
- Each document may be represented as a **set** of k-shingles that appear one or more times within that document.

k-shingles

- A document is a string of characters.
- A k-shingle (or k-gram) for a document is any substring of length k found within the document.
- Each document may be represented as a **set** of k-shingles that appear one or more times within that document.
- **Example**: The set of all 3-shingles for the first sentence on this slide:

$$\{\text{“A d”}, \text{“ do”}, \text{“doc”}, \text{“ocu”}, \text{“cum”}, \text{“ume”}, \text{“men”}, \ldots, \text{“ers”}\}$$

k-shingles

- A document is a string of characters.
- A k-shingle (or k-gram) for a document is any substring of length k found within the document.
- Each document may be represented as a **set** of k-shingles that appear one or more times within that document.
- **Example**: The set of all 3-shingles for the first sentence on this slide:

$$\{\text{“A d”}, \text{“ do”}, \text{“doc”}, \text{“ocu”}, \text{“cum”}, \text{“ume”}, \text{“men”}, \ldots, \text{“ers”}\}$$

- Several options regarding white spaces:

k-shingles

- A document is a string of characters.
- A k-shingle (or k-gram) for a document is any substring of length k found within the document.
- Each document may be represented as a **set** of k-shingles that appear one or more times within that document.
- **Example**: The set of all 3-shingles for the first sentence on this slide:

$$\{\text{“A d”}, \text{“ do”}, \text{“doc”}, \text{“ocu”}, \text{“cum”}, \text{“ume”}, \text{“men”}, \ldots, \text{“ers”}\}$$

- Several options regarding white spaces:
  - Replace any sequence of one or more white spaces by a single blank.

k-shingles

- A document is a string of characters.
- A k-shingle (or k-gram) for a document is any substring of length k found within the document.
- Each document may be represented as a **set** of k-shingles that appear one or more times within that document.
- **Example**: The set of all 3-shingles for the first sentence on this slide:

$$\{\text{“A d”}, \text{“ do”}, \text{“doc”}, \text{“ocu”}, \text{“cum”}, \text{“ume”}, \text{“men”}, \ldots, \text{“ers”}\}$$

- Several options regarding white spaces:
  - Replace any sequence of one or more white spaces by a single blank.
  - Remove all white spaces.

Size of shingles

- For small k we would expect most sequences of k characters to appear in most documents.

Size of shingles

- For small k we would expect most sequences of k characters to appear in most documents.
- For $k = 1$ most documents will have most of the common characters and few other characters, so almost all documents will have high similarity.

Size of shingles

- For small k we would expect most sequences of k characters to appear in most documents.
- For $k = 1$ most documents will have most of the common characters and few other characters, so almost all documents will have high similarity.
- k should be picked large enough that the probability of any given shingle appearing in any given document is low.

Size of shingles

- For small k we would expect most sequences of k characters to appear in most documents.
- For $k = 1$ most documents will have most of the common characters and few other characters, so almost all documents will have high similarity.
- k should be picked large enough that the probability of any given shingle appearing in any given document is low.
- **Example**: Let us check two words *document* and *monument*:

$$\begin{aligned} SIM(\{d, o, c, u, m, e, n, t\}, \{m, o, n, u, m, e, n, t\}) &= 6/8 \\ SIM(\{doc, ocu, cum, ume, men, ent\}, & \\ \{mon, onu, num, ume, men, ent\}) &= 3/9 \end{aligned}$$

Size of shingles

- **Example**:

Size of shingles

- **Example**:
  - For corpus of emails setting $k = 5$ should be fine.

Size of shingles

- **Example**:
  - For corpus of emails setting $k = 5$ should be fine.
  - If only English letters and a general white-space character appear in emails, then there would be $27^5 = 14348907$ possible shingles.

Size of shingles

- **Example**:
  - For corpus of emails setting $k = 5$ should be fine.
  - If only English letters and a general white-space character appear in emails, then there would be $27^5 = 14348907$ possible shingles.
  - Since typical email is much smaller than 14 million characters long, this can be right value.

Size of shingles

- **Example**:
  - For corpus of emails setting $k = 5$ should be fine.
  - If only English letters and a general white-space character appear in emails, then there would be $27^5 = 14348907$ possible shingles.
  - Since typical email is much smaller than 14 million characters long, this can be right value.
  - Since distribution of characters is not uniform, the above estimate should be corrected, for example, by assuming that there are only 20 characters.

Hashing shingles

- Instead of using substrings directly as shingles, we can pick a hash function that maps strings of length k to some number of buckets.

Hashing shingles

- Instead of using substrings directly as shingles, we can pick a hash function that maps strings of length k to some number of buckets.
- Then, the resulting bucket number can be treated as the shingle.

Hashing shingles

- Instead of using substrings directly as shingles, we can pick a hash function that maps strings of length k to some number of buckets.
- Then, the resulting bucket number can be treated as the shingle.
- The set representing a document is then the set of integers that are bucket numbers of one or more k-shingles that appear in the document.

Hashing shingles

- Instead of using substrings directly as shingles, we can pick a hash function that maps strings of length k to some number of buckets.
- Then, the resulting bucket number can be treated as the shingle.
- The set representing a document is then the set of integers that are bucket numbers of one or more k-shingles that appear in the document.
- **Example**:

Hashing shingles

- Instead of using substrings directly as shingles, we can pick a hash function that maps strings of length k to some number of buckets.
- Then, the resulting bucket number can be treated as the shingle.
- The set representing a document is then the set of integers that are bucket numbers of one or more k-shingles that appear in the document.
- **Example**:
  - Each 9-shingle from a document can be mapped to a bucket number in the range from 0 to $2^{32} - 1$.

Hashing shingles

- Instead of using substrings directly as shingles, we can pick a hash function that maps strings of length k to some number of buckets.
- Then, the resulting bucket number can be treated as the shingle.
- The set representing a document is then the set of integers that are bucket numbers of one or more k-shingles that appear in the document.
- **Example**:
    - Each 9-shingle from a document can be mapped to a bucket number in the range from 0 to $2^{32} - 1$.
    - Instead of **nine** we use then **four** bytes and can manipulate (hashed) shingles by single-word machine operations.

Hashing shingles

- Short shingles vs. hashed shingles

Hashing shingles

- Short shingles vs. hashed shingles
  - If we use 4-shingles, most sequences of four bytes are unlikely or impossible to find in typical documents.

Hashing shingles

- Short shingles vs. hashed shingles
  - If we use 4-shingles, most sequences of four bytes are unlikely or impossible to find in typical documents.
  - The effective number of different shingles is approximately $20^4 = 160000$ much less than 2^{32}.

Hashing shingles

- Short shingles vs. hashed shingles
  - If we use 4-shingles, most sequences of four bytes are unlikely or impossible to find in typical documents.
  - The effective number of different shingles is approximately $20^4 = 160000$ much less than 2^{32}.
  - if we use 9-shingles, there are many more than 2^{32} likely shingles.

Hashing shingles

- Short shingles vs. hashed shingles
  - If we use 4-shingles, most sequences of four bytes are unlikely or impossible to find in typical documents.
  - The effective number of different shingles is approximately $20^4 = 160000$ much less than 2^{32}.
  - if we use 9-shingles, there are many more than 2^{32} likely shingles.
  - When we hash them down to four bytes, we can expect almost any sequence of four bytes to be possible.

Outline

Similarity-preserving summaries of sets

- Sets of shingles are large!

Similarity-preserving summaries of sets

- Sets of shingles are large!
- Even if we hash them to four bytes each, the space needed to store a set is still roughly four times the space taken by the document.

Similarity-preserving summaries of sets

- Sets of shingles are large!
- Even if we hash them to four bytes each, the space needed to store a set is still roughly four times the space taken by the document.
- If we have millions of documents, it may well not be possible to store all the shingle-sets in main memory.

Similarity-preserving summaries of sets

- Sets of shingles are large!
- Even if we hash them to four bytes each, the space needed to store a set is still roughly four times the space taken by the document.
- If we have millions of documents, it may well not be possible to store all the shingle-sets in main memory.
- We would like to replace large sets by much smaller representations called **signatures**.

Similarity-preserving summaries of sets

- Sets of shingles are large!
- Even if we hash them to four bytes each, the space needed to store a set is still roughly four times the space taken by the document.
- If we have millions of documents, it may well not be possible to store all the shingle-sets in main memory.
- We would like to replace large sets by much smaller representations called **signatures**.
- The signatures, however, should preserve (at least to some extent) the similarity between sets.

Matrix representation of sets

- **Characteristic matrix**

Matrix representation of sets

- **Characteristic matrix**
  - The columns of the matrix correspond to the sets.

Matrix representation of sets

- **Characteristic matrix**
  - The columns of the matrix correspond to the sets.
  - The rows correspond to elements of the universal set from which elements of the sets are drawn.

Matrix representation of sets

- **Characteristic matrix**
  - The columns of the matrix correspond to the sets.
  - The rows correspond to elements of the universal set from which elements of the sets are drawn.
  - There is a 1 in row r and column c if the element for row r is a member of the set for column c.

Matrix representation of sets

- **Characteristic matrix**
  - The columns of the matrix correspond to the sets.
  - The rows correspond to elements of the universal set from which elements of the sets are drawn.
  - There is a 1 in row r and column c if the element for row r is a member of the set for column c.
  - Otherwise the value in position (r, c) is 0.

Matrix representation of sets

- **Example**:
  - Let the universal set be $\{a, b, c, d, e\}$.
  - Let $S_1 = \{a, d\}$, $S_2 = \{c\}$, $S_3 = \{b, d, e\}$, $S_4 = \{a, c, d\}$.

Element	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

- It is important to remember that the characteristic matrix is unlikely to be the way the data is stored, but it is useful as a way to visualize the data!

Minhashing

- The signatures we desire to construct for sets are composed of the results of some number of calculations (say several hundred) each of which is a **minhash** of the characteristic matrix.

Minhashing

- The signatures we desire to construct for sets are composed of the results of some number of calculations (say several hundred) each of which is a **minhash** of the characteristic matrix.
- To minhash a set represented by a column of the characteristic matrix, pick a permutation of the rows.

Minhashing

- The signatures we desire to construct for sets are composed of the results of some number of calculations (say several hundred) each of which is a **minhash** of the characteristic matrix.
- To minhash a set represented by a column of the characteristic matrix, pick a permutation of the rows.
- The minhash value of any column is the number of the first row, in the permuted order, in which the column has a 1 (or, the first element of the set in the given permutation).

Minhashing

- The signatures we desire to construct for sets are composed of the results of some number of calculations (say several hundred) each of which is a **minhash** of the characteristic matrix.
- To minhash a set represented by a column of the characteristic matrix, pick a permutation of the rows.
- The minhash value of any column is the number of the first row, in the permuted order, in which the column has a 1 (or, the first element of the set in the given permutation).
- The index of the first row is 0 in the following.

- **Example**:
  - Let us pick the order of rows $beadc$ for the matrix from the previous example.

Element	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

  - In this matrix, we can read off the values of minhash (mh) by scanning from the top until we come to a 1.
  - Thus, we see that $mh(S_1) = 2$ (a), $mh(S_2) = 4$ (c), $mh(S_3) = 0$ (b), and $mh(S_4) = 2$ (a).

Minhashing and Jaccard similarity

- There is a remarkable connection between minhashing and Jaccard similarity of the sets that are minhashed:

Minhashing and Jaccard similarity

- There is a remarkable connection between minhashing and Jaccard similarity of the sets that are minhashed:
  - The probability that the minhash function for a random permutation of rows produces the same value for two sets equals the Jaccard similarity of those sets.

Minhashing and Jaccard similarity

- Let us consider two sets, i.e., two columns of the characteristic matrix.

Element	S_1	S_4
b	0	0
e	0	0
a	1	1
d	1	1
c	0	1

Minhashing and Jaccard similarity

- Let us consider two sets, i.e., two columns of the characteristic matrix.

Element	S_1	S_4
b	0	0
e	0	0
a	1	1
d	1	1
c	0	1

- The rows can be divided into three classes:

Minhashing and Jaccard similarity

- Let us consider two sets, i.e., two columns of the characteristic matrix.

Element	S_1	S_4
b	0	0
e	0	0
a	1	1
d	1	1
c	0	1

- The rows can be divided into three classes:
  - Type X rows have 1 in both columns,

Minhashing and Jaccard similarity

- Let us consider two sets, i.e., two columns of the characteristic matrix.

Element	S_1	S_4
b	0	0
e	0	0
a	1	1
d	1	1
c	0	1

- The rows can be divided into three classes:
  - Type X rows have 1 in both columns,
  - Type Y rows have 1 in one of the columns and 0 in the other,

Minhashing and Jaccard similarity

- Let us consider two sets, i.e., two columns of the characteristic matrix.

Element	S_1	S_4
b	0	0
e	0	0
a	1	1
d	1	1
c	0	1

- The rows can be divided into three classes:
  - Type X rows have 1 in both columns,
  - Type Y rows have 1 in one of the columns and 0 in the other,
  - Type Z rows have 0 in both columns.

Minhashing and Jaccard similarity

- Since the matrix is sparse, most rows are of type Z.

Minhashing and Jaccard similarity

- Since the matrix is sparse, most rows are of type Z.
- The ratio of the numbers of type X and type Y rows determine both $SIM(S, T)$ and the probability that $mh(S) = mh(T)$.

Minhashing and Jaccard similarity

- Since the matrix is sparse, most rows are of type Z.
- The ratio of the numbers of type X and type Y rows determine both $SIM(S, T)$ and the probability that $mh(S) = mh(T)$.
- Let there be x rows of type X and y rows of type Y.

Minhashing and Jaccard similarity

- Since the matrix is sparse, most rows are of type Z.
- The ratio of the numbers of type X and type Y rows determine both $SIM(S,T)$ and the probability that $mh(S) = mh(T)$.
- Let there be x rows of type X and y rows of type Y.
- Then, the Jaccard similarity is:

Minhashing and Jaccard similarity

- Since the matrix is sparse, most rows are of type Z.
- The ratio of the numbers of type X and type Y rows determine both $SIM(S,T)$ and the probability that $mh(S) = mh(T)$.
- Let there be x rows of type X and y rows of type Y.
- Then, the Jaccard similarity is:

$$SIM(S,T) = \frac{x}{x+y} \,.$$

Minhashing and Jaccard similarity

- Since the matrix is sparse, most rows are of type Z.
- The ratio of the numbers of type X and type Y rows determine both $SIM(S,T)$ and the probability that $mh(S) = mh(T)$.
- Let there be x rows of type X and y rows of type Y.
- Then, the Jaccard similarity is:

$$SIM(S,T) = \frac{x}{x+y}\,.$$

- If we imagine the rows permuted randomly, and we proceed from the top, the probability that we shall meet a type X row before we meet a type Y row is

Minhashing and Jaccard similarity

- Since the matrix is sparse, most rows are of type Z.
- The ratio of the numbers of type X and type Y rows determine both $SIM(S,T)$ and the probability that $mh(S) = mh(T)$.
- Let there be x rows of type X and y rows of type Y.
- Then, the Jaccard similarity is:

$$SIM(S,T) = \frac{x}{x+y}\,.$$

- If we imagine the rows permuted randomly, and we proceed from the top, the probability that we shall meet a type X row before we meet a type Y row is, as before,

$$P(mh(S) = mh(T)) = \frac{x}{x+y}\,.$$

Minhash signatures

- For a given collection of sets represented by their characteristic matrix M, the signatures are produced in the following way:

Minhash signatures

- For a given collection of sets represented by their characteristic matrix M, the signatures are produced in the following way:
  - Pick at random some number n of permutations of the rows of M (let say, around 100 or 1000).

Minhash signatures

- For a given collection of sets represented by their characteristic matrix M, the signatures are produced in the following way:
  - Pick at random some number n of permutations of the rows of M (let say, around 100 or 1000).
  - Call the minhash functions determined by these permutations mh_1, mh_2, ..., mh_n.

Minhash signatures

- For a given collection of sets represented by their characteristic matrix M, the signatures are produced in the following way:
  - Pick at random some number n of permutations of the rows of M (let say, around 100 or 1000).
  - Call the minhash functions determined by these permutations mh_1, mh_2, ..., mh_n.
  - From the column representing set S, construct the minhash signature for S, the vector $(mh_1(S), mh_2(S), \ldots, mh_n(S))$ – represented as a column.

Minhash signatures

- For a given collection of sets represented by their characteristic matrix M, the signatures are produced in the following way:
  - Pick at random some number n of permutations of the rows of M (let say, around 100 or 1000).
  - Call the minhash functions determined by these permutations mh_1, mh_2, ..., mh_n.
  - From the column representing set S, construct the minhash signature for S, the vector $(mh_1(S), mh_2(S), \ldots, mh_n(S))$ – represented as a column.
  - Thus, we can form from matrix M a **signature matrix**, in which the i-th column of M is replaced by the minhash signature for (the set of) the i-th column.

Minhash signatures

- For a given collection of sets represented by their characteristic matrix M, the signatures are produced in the following way:
  - Pick at random some number n of permutations of the rows of M (let say, around 100 or 1000).
  - Call the minhash functions determined by these permutations mh_1, mh_2, ..., mh_n.
  - From the column representing set S, construct the minhash signature for S, the vector $(mh_1(S), mh_2(S), \ldots, mh_n(S))$ – represented as a column.
  - Thus, we can form from matrix M a **signature matrix**, in which the i-th column of M is replaced by the minhash signature for (the set of) the i-th column.
- The signature matrix has the same number of columns as M, but only n rows!

Minhash signatures

- For a given collection of sets represented by their characteristic matrix M, the signatures are produced in the following way:
  - Pick at random some number n of permutations of the rows of M (let say, around 100 or 1000).
  - Call the minhash functions determined by these permutations mh_1, mh_2, ..., mh_n.
  - From the column representing set S, construct the minhash signature for S, the vector $(mh_1(S), mh_2(S), \ldots, mh_n(S))$ – represented as a column.
  - Thus, we can form from matrix M a **signature matrix**, in which the i-th column of M is replaced by the minhash signature for (the set of) the i-th column.
- The signature matrix has the same number of columns as M, but only n rows!
- Even if M is not represented explicitly (but as a sparse matrix by the location of its ones), it is normal for the signature matrix to be much **smaller** than M.

Computing minhash signatures

- Unfortunately, it is **not** feasible to permute a large characteristic matrix explicitly.

Computing minhash signatures

- Unfortunately, it is **not** feasible to permute a large characteristic matrix explicitly.
- Even picking a random permutation of millions or billions of rows is time-consuming.

Computing minhash signatures

- Unfortunately, it is **not** feasible to permute a large characteristic matrix explicitly.
- Even picking a random permutation of millions or billions of rows is time-consuming.
- Fortunately, it is possible to simulate the effect of a random permutation by a **random hash function** that maps row numbers to as many buckets as there are rows.

Computing minhash signatures

- A hash function that maps integers $0, 1, \ldots, k-1$ to bucket numbers 0 through $k-1$ typically will map some pairs of integers to the same bucket and leave other buckets unfilled.

Computing minhash signatures

- A hash function that maps integers $0, 1, \ldots, k-1$ to bucket numbers 0 through $k-1$ typically will map some pairs of integers to the same bucket and leave other buckets unfilled.
- This difference is unimportant as long as k is large and there are not too many collisions.

Computing minhash signatures

- A hash function that maps integers $0, 1, \ldots, k-1$ to bucket numbers 0 through $k-1$ typically will map some pairs of integers to the same bucket and leave other buckets unfilled.
- This difference is unimportant as long as k is large and there are not too many collisions.
- We can maintain the fiction that our hash function h **permutes** row r to position $h(r)$ in the permuted order.

Computing minhash signatures

- Instead of picking n random permutations of rows, we pick n randomly chosen hash functions $h_1, h_2, \ldots, h_n$ on the rows.

Computing minhash signatures

- Instead of picking n random permutations of rows, we pick n randomly chosen hash functions $h_1, h_2, \ldots, h_n$ on the rows.
- We construct the signature matrix by considering each row in their given order.

Computing minhash signatures

- Instead of picking n random permutations of rows, we pick n randomly chosen hash functions $h_1, h_2, \ldots, h_n$ on the rows.
- We construct the signature matrix by considering each row in their given order.
- Let $SIG(i, c)$ be the element of the signature matrix for the i-th hash function and column c defined by

$$SIG(i, c) = \min\{h_i(r) : \text{for such } r \text{ that } c \text{ has 1 in row } r\}$$

Computing minhash signatures

- **Example**:
  - Let us consider two hash functions h_1 and h_2:

$$h_1(r) = r + 1 \bmod 5 \quad h_2(r) = 3r + 1 \bmod 5$$

Row	S_1	S_2	S_3	S_4	$h_1(r)$	$h_2(r)$
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		

Computing minhash signatures

- **Example**:
  - Let us consider two hash functions h_1 and h_2:

$$h_1(r) = r + 1 \bmod 5 \quad h_2(r) = 3r + 1 \bmod 5$$

Row	S_1	S_2	S_3	S_4	$h_1(r)$	$h_2(r)$
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

Computing minhash signatures

- **Example**:
  - The signature matrix is:

	S_1	S_2	S_3	S_4
$SIG(1,c)$				
$SIG(2,c)$				

Computing minhash signatures

- **Example**:
  - The signature matrix is:

	S_1	S_2	S_3	S_4
$SIG(1, c)$	1	3	0	1
$SIG(2, c)$	0	2	0	0

- We can estimate the Jaccard similarities of the underlying sets from this signature matrix:

Computing minhash signatures

- **Example**:
  - The signature matrix is:

	S_1	S_2	S_3	S_4
$SIG(1,c)$	1	3	0	1
$SIG(2,c)$	0	2	0	0

- We can estimate the Jaccard similarities of the underlying sets from this signature matrix:

$$SIM(S_1, S_2) = 0 \quad SIM(S_1, S_3) = 1/2 \quad SIM(S_1, S_4) = 1$$

Computing minhash signatures

- **Example**:
  - The signature matrix is:

	S_1	S_2	S_3	S_4
$SIG(1,c)$	1	3	0	1
$SIG(2,c)$	0	2	0	0

- We can estimate the Jaccard similarities of the underlying sets from this signature matrix:

$$SIM(S_1, S_2) = 0 \quad SIM(S_1, S_3) = 1/2 \quad SIM(S_1, S_4) = 1$$

while the true similarities are:

$$SIM(S_1, S_2) = 0 \quad SIM(S_1, S_3) = 1/4 \quad SIM(S_1, S_4) = 2/3$$

Outline

Summary

- Similarity of documents.
- Jaccard similarity.
- Minhash technique.

Bibliography

- J. Leskovec, A. Rajaraman, and J. D. Ullman. *Mining of Massive Datasets*. Cambridge University Press, 2014