COMP336 — Big Data

Week 7 Lecture 1: Finding Similar Items

Diego Mollá

COMP348 2018H1

Abstract

This will be the first lecture of a series of lectures focusing on algorithms and approaches for mining massive data sets which do not fit on memory. This lecture will look at the problem of identifying similar items. We will see how we can trade-off accuracy for scalability by applying clever hashing techniques.

Update April 6, 2018

Contents

1 Mining of Massive Datasets **2**
1.1 Data Mining . . . 2
1.2 Finding Similar Items . . . 3

2 Shingling and Minhashing **3**
2.1 Shingling . . . 3
2.2 Minhashing . . . 5

3 Locality-Sensitive Hashing **9**

Reading

- Leskovec, Rajaraman, Ullman (2014): Mining of Massive Datasets, Chapter 3. *http://www.mmds.org/*

Who am I

Diego Mollá Aliod
Email: diego.molla-aliod@mq.edu.au
Office: E6A331

Webpage: http://web.science.mq.edu.au/ diego/
Research interests: Text processing, machine learning: question answering, named entity recognition, text summarisation.

What We've Seen So Far

Amin Beheshti

1. Data and Big Data
2. Organizing Big Data
3. Curating Big Data
4. Processing Big Data: Cloud computing
5. Processing Big Data: MapReduce (I)
6. Processing Big Data: MapReduce (II)

These are techniques for organising, curating, and *general techniques* for processing Big Data.

What We're Yet to See

Diego Molla

7. Finding Similar Items
8. Mining Data Streams
9. Link Analysis
10. Frequent Itemsets
11. Large-scale Machine Learning (I)
12. Large-scale Machine Learning (II)

These are *specific techniques* for particular problems that require the processing of big data.

1 Mining of Massive Datasets

1.1 Data Mining

Mining of Big Data

- MapReduce is a technique that manages parallel processing of big data.
- It can be very effective for tasks that require massive parallelisation.
- But there is a trade-off between computation and resources.
  - We can reduce computation time by increasing the number of computation nodes.
  - If the number of computation nodes is limited, MapReduce may take long to complete.
- Sometimes we need specific methods for particular tasks.

1.2 Finding Similar Items

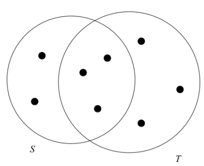
Applications of Near-Neighbour Search

- Find plagiarised documents.
- Detect mirror pages.
- Find news articles from the same source.
- Collaborative Filtering.
  - On-line purchases.
  - Movie ratings.

Jaccard Similarity

- The Jaccard similarity of two sets A and B is:

$$\frac{|A \cap B|}{|A \cup B|}$$



- We want to find efficient methods to:
  1. Represent documents as sets.
  2. Compute the Jaccard similarity between A and B, where A and B can be large sets.
  3. Find the pairs with highest Jaccard similarity in a large collection of sets.

2 Shingling and Minhashing

2.1 Shingling

Shingling

- A document can be characterised by the set of words appearing in it.
- But this would ignore ordering of words.
- Shingles are an alternative that can represent documents ...
  - ... as sets of something ...
  - ... that account for word ordering (of some sort).

k-Shingles

k-shingle
Define a k-shingle for a document to be any substring of length k found within the document.

Example
A document D is the string $abcdabd$. If $k = 2$, then the set of 2-shingles for D is:

$$\{ab, bc, cd, da, bd\}$$

Note that the substring ab appears twice within D but appears only once as a shingle.

Matrix Representation of Sets

A (sparse) matrix can represent a collection of sets by using the following convention:

- Each row indicates a possible element.
- Each column represents a set.
- A value of 1 in row r and column c indicates that the element r is in set c.

Example

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

This matrix represents the following sets:

1. $S_1 = \{a, d\}$
2. $S_2 = \{c\}$
3. $S_3 = \{b, d, e\}$
4. $S_4 = \{a, c, d\}$

How to Represent Shingles Efficiently?

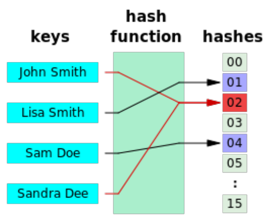
- Assuming that there are 32 characters, then there are 32^9 different 9-shingles.
- A matrix with 32^9 rows will not fit in memory of a regular computer.
- But most shingles will never occur, so we can represent a shingle using a *hash function*.

Hashing Shingles

- Set a hash function that maps a shingle to a number within a limited range, e.g. 0 to $2^{32} - 1$.
- The shingle is represented by its hash number.
- We can now represent all shingles in 4 bytes.

What is a Hash Function?

- A hash function maps an item (string, number, list, etc) to a number within a predefined range.
- A good hash function will minimise the number of *collisions*.
  - Two different items that map to the same value.
- Hashes are used in programming languages to speed up the search in a list.



Exercise

Why not using, say, 4-shingles?

- 4-shingles may not have the power to differentiate documents.
- *Represent two documents using 4-shingles and compute the Jaccard difference.*
- *Represent the same two documents using 9-shingles and compute the Jaccard difference.*

Why using 9-shingles hashed down to 4 bytes?

- The set of 9-shingles hashed down to 4 bytes takes the same space as the set of 4-shingles.
- But we use most of the hash space to represent 9-shingles and we underuse much of the space to represent 4-shingles.
- *Represent the same two documents using 9-shingles hashed down to 4 bytes and compute the Jaccard difference.*

Trade-off
Different shingles may map to the same hash number (hash collisions) so we trade accuracy for scalability.

2.2 Minhashing

Motivation for Minhashing and Locality Sensitive Hashing

- Even if we can represent a document by its k-shingles, how do we find all similar documents?
- Imagine you have $N = 1$ million documents.
- Naively, if we compute pairwise Jaccard similarities for every pair of docs, you would need $N \times (N - 1)/2 \approx 5 \times 10^{11}$ comparisons.
- At 10^5 secs per day and 10^6 comparisons per sec, it would take 5 days.
- For $N = 10$ million it would take more than a year!

Minhashing

- To minhash a set represented by a column of the characteristic matrix, pick a permutation of the rows.
- The minhash value of a column is the first row, in the permuted order, in which the column has a 1.

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

$h(S_1) = a$, $h(S_2) = c$, $h(S_3) = b$, $h(S_4) = a$

Minhashing and Jaccard Similarity

Minhashing and Jaccard Similarity
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 is a good hash function for computing the Jaccard similarity.
- If the Jaccard similarity between two sets C_1 and C_2 is high, then with high probability $h(C_1) = h(C_2)$.
- If the Jaccard similarity between two sets C_1 and C_2 is low, then with high probability $h(C_1) \neq h(C_2)$.

Minhash Signatures

1. Use several (e.g. 100) independent permutations of the characteristic matrix.
2. Compute the minhashing according to each permutation.
3. Now we have 100 independent signatures of each document.
4. These 100 signatures form the rows of a matrix called *signature matrix*.
5. This new signature matrix has many less rows than the original characteristic matrix!

Computing Minhash Signatures

- It is not practical to permute a large characteristic matrix explicitly.
- Instead, we will simulate a random permutation by defining a random *hash function* that maps row numbers to their new row positions.

Procedure

1. Initialise $SIG(r, c) \leftarrow \infty$
2. Scan row $r = 1 \cdots M$ of the characteristic matrix.
3. Compute $h_1(r), h_2(r), \ldots, h_n(r)$
4. For each column c do the following:
   (a) If c has 0 in row r, do nothing.
   (b) If c has 1 in row r, $SIG(r, c) \leftarrow \min(SIG(r, c), h_i(r)$

Example

Let's define two hash functions (on the right of the characteristic matrix below).

Element	S_1	S_2	S_3	S_4	$(x+1)\%5$	$(3x+1)\%5$
a	1	0	0	1	1	1
b	0	0	1	0	2	4
c	0	1	0	1	3	2
d	1	0	1	1	4	0
e	0	0	1	0	0	3

Example - Initialisation

Element	S_1	S_2	S_3	S_4	$(x+1)\%5$	$(3x+1)\%5$
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

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

Example - After Scanning Row 0

	Element	S_1	S_2	S_3	S_4	$(x+1)\%5$	$(3x+1)\%5$
$\Rightarrow$	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

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

Example - After Scanning Row 1

	Element	S_1	S_2	S_3	S_4	$(x+1)\%5$	$(3x+1)\%5$
	0	1	0	0	1	1	1
$\Rightarrow$	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

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

Example - After Scanning Row 2

	Element	S_1	S_2	S_3	S_4	$(x+1)\%5$	$(3x+1)\%5$
	0	1	0	0	1	1	1
	1	0	0	1	0	2	4
$\Rightarrow$	2	0	1	0	1	3	2
	3	1	0	1	1	4	0
	4	0	0	1	0	0	3

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

Example - After Scanning Row 3

	Element	S_1	S_2	S_3	S_4	$(x+1)\%5$	$(3x+1)\%5$
	0	1	0	0	1	1	1
	1	0	0	1	0	2	4
	2	0	1	0	1	3	2
$\Rightarrow$	3	1	0	1	1	4	0
	4	0	0	1	0	0	3

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

Example - After Scanning Row 4

	Element	S_1	S_2	S_3	S_4	$(x+1)\%5$	$(3x+1)\%5$
	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
$\Rightarrow$	4	0	0	1	0	0	3

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

Exercise: Compare Jaccard Similarities

Fill the following tables showing the pairwise Jaccard similarities using the characteristic matrix and the signature matrix.

Characteristic Matrix

	S_1	S_2	S_3	S_4
S_1		0	1/4	2/3
S_2				
S_3				
S_4				

Signature Matrix

	S_1	S_2	S_3	S_4
S_1		0	1/2	1
S_2				
S_3				
S_4				

3 Locality-Sensitive Hashing

What we Know So Far

- How to represent documents as sets.
- How to compress the sets to small signatures.
- How to use the small signatures to approximate the Jaccard Similarity.

But
How do we efficiently find the most similar documents?

Locality-Sensitive Hashing

- It is very time-consuming to find the expected similarity of all pairs of documents.
  - *Would you use MapReduce for this...?*
- But often we only need to find the pairs of documents that are most similar.
- We will study an approach that can be used for documents represented by shingle-sets and then minhashed to short signatures.

⇒ Read Section 3.6 of the textbook if you want to learn the general theory of locality-sensitive hashing.

LSH for Minhashing — The Key Idea

- Hash items several times, in such a way that similar items are more likely to be hashed to the same bucket than dissimilar items are.
- We then consider any pairs that hashed to the same bucket for any of the hashings to be candidate pairs.
- The key is to find hash functions that reduce errors:

  False positives: Dissimilar pairs that hash to the same bucket.

  False negatives: Similar pairs that do not hash to the same bucket.
- *What is worse, a false positive or a false negative...?*

LSH Using the Signature Matrix

- We want to find hash functions that put similar columns of the signature matrix in the same buckets.
- Remember that we are using the Jaccard similarity metric, so columns that share many values in their rows are more similar.

Big Idea
Define hash functions for *bands* of the signature matrix.

Approach

1. Divide signature matrix into b bands of r rows.
2. For each band, hash its portion of each column to a hash table with k buckets.
   - Make k as large as possible.
   - We want to avoid collisions which would lead to false positives.
   - We are using hashing as a quick method to compare column fragments.
3. Candidate column pairs are those that hash to the same bucket for at least one band.
4. Tune b and r to catch most similar pairs, but few non-similar pairs.

Example — Signature Matrix

This is an artificial example! The signature matrix would have many more rows and columns.

	S_1	S_2	S_3	S_4
r_1	1	0	1	1
r_2	2	1	0	2
r_3	1	2	2	0
r_4	0	1	1	2
r_5	3	0	3	1
r_6	1	0	1	0

Example — After Dividing Matrix

This is an artificial example! The signature matrix would have many more rows and columns.

	S_1	S_2	S_3	S_4
r_1	1	0	1	1
r_2	2	1	0	2
r_3	1	2	2	0
r_4	0	1	1	2
r_5	3	0	3	1
r_6	1	0	1	0

Example — After Hashing Columns

This is an artificial example! The signature matrix would have many more rows and columns.

	S_1	S_2	S_3	S_4
r_1	1	0	1	1
r_2	2	1	0	2
r_3	1	2	2	0
r_4	0	1	1	2
r_5	3	0	3	1
r_6	1	0	1	0

In this example, column fragments with the same colour indicate that they were hashed to the same bucket. We are actually using hashing as a fast way to compare the column fragments. If the number of buckets is large enough we can assume there would not be too many collisions.

Example — Final Candidate Pairs

This is an artificial example! The signature matrix would have many more rows and columns.

	S_1	S_2	S_3	S_4
r_1	1	0	1	1
r_2	2	1	0	2
r_3	1	2	2	0
r_4	0	1	1	2
r_5	3	0	3	1
r_6	1	0	1	0

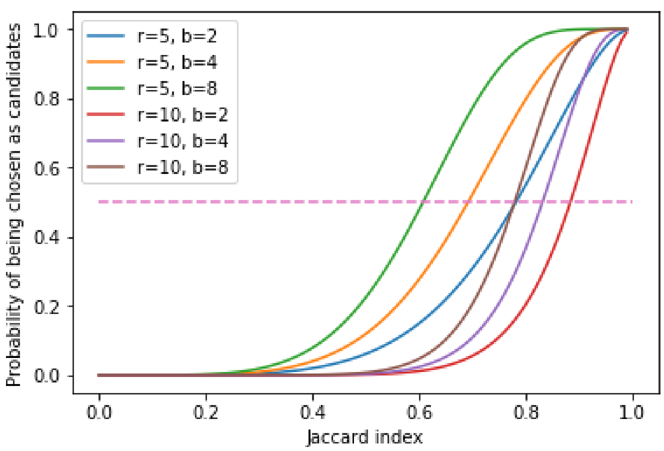
Candidate pairs are: $(S_1, S_4), (S_2, S_3), (S_1, S_3)$

Analysis of This Technique

Suppose that we use b bands of r rows each, and suppose that a particular pair of documents has Jaccard similarity s.

- The probability that the signatures agree in all rows of one particular band is s^r.
- The probability that the signatures do not agree in at least one row of a particular band is $1 - s^r$.
- The probability that the signatures do not agree in all rows of any of the bands is $(1 - s^r)^b$.
- The probability that the signatures agree in all rows of at least one band is $1 - (1 - s^r)^b$.
- The formula $(1/b)^{1/r}$ approximates the value of s where the probability is 0.5.

Picking r and b: the S-curve



Locality Search Hashing: The Complete Procedure

1. Pick a value of k and construct the set of k-shingles of each document.
   - Optionally, hash the k-shingles to shorter bucket numbers.
2. Sort the document-shingle pairs to order them by shingle.
3. Pick a length n for the minhash signature matrix. Compute the minhash signature matrix.

4. Choose a threshold t that decides the jaccard similarity of two documents to be to be considered "similar". Select values of b and r so that t is approximately $(1/b)^{1/r}$.

5. Construct candidate pairs by applying LSH.

6. Examine each candidate pairs more in detail.

Which of these steps can be parallelised?

Take-home Messages

- *Shingling*: A way to convert documents to sets.
- *Minhashing*: A way to convert large sets to short signatures.
- *Locality-Sensitive Hashing (LSH)*: An efficient way to detect pairs of similar documents from a large collection.
- Both minhashing and LSH trade efficiency for accuracy.

What's Next

Week 8

- RECESS: 16-29 April.
- Assignment 2 deadline: Workshop week 9.
- Topic week 8: Mining Data Streams.