15-853:Algorithms in the Real World

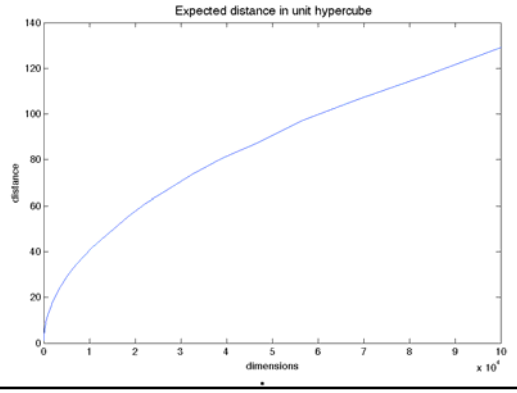
Nearest Neighbors in **High Dimensions**

- Curse of dimensionality
- Representing Documents and Products as Sets, Set similarity
- Minhash for compact set signatures
- Locality sensitive hashing

More "Big Data"

Curse of Dimensionality

High-dimension spaces are lonely places.



Curse of Dimensionality

High-dimension spaces are also weird places.

There exist exp(N) many points at distance almost 1 from each other. (in the homework.)

Take a ball in N dimensions:

Most of the volume of the ball is very near the surface

Most of the volume of the ball is very near the equator.

In this and the next two lectures, see way to deal with this.

Challenges

1. Presenting high dimensional objects compactly, so that they can be stored in the RAM and quickly compared for similarity.
   – Today: Min-hash signatures for sets
2. Finding similar items from a collection of high dimensional objects.
   – Today: Locality sensitive hashing based on min-hash

Material based largely on "Mining of Massive Datasets" book by Rajaraman and Ullman (available free for download!)

High Dimensional Data

Examples of high dimensional data:

Representing **documents** as vectors (or sets)

- "bag of words" (TF-IDF weighting)
- shingles (k-substrings)

Bag of Words

"The course will cover both the theory behind the algorithms and case studies…"

→ {the: 3, course: 1, will: 1, …}

→ [0,0, …., 1 …, 3,0,0,… 1,0,..]

→ For representing **sets**, only **binary values**

Note: In practice stop-words like "the" would be removed.

Extremely sparse, so we use *sparse vectors*

→ [(118,1), (107872,1), (200938, 1) ….]

Shingles

Take the text and break into k character long strings.

abacbdaeacf -> abac, bacb, acbd, cbda, bdae,…

Choose k so that any shingle is unlikely to occur in any doc.

E.g., k = 5 would mean 27^5 ~ 14M possible shingles.

Actually, think of having only 20 letters (x, q, z, etc.)

Say choose k=8 or so, so $20^8 \sim 2^{32}$

Finally, hash shingles to 32 bit words ("cheap compression").

Applications

Applications of finding Similar (nearest) Items

- Filter duplicate docs in search engine
- Plagiarism, mirror pages
- Recommend similar products, movies

Collaborative Filtering

– represent movie as a vector of ratings by users

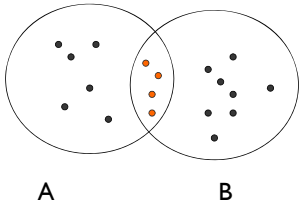
– represent product by binary vector x: x(j) = 1 if user j bought the item, 0 otherwise

Defining Similarity

Many ways to define similarity.

One similarity metric, "distance", for **sets**

Jaccard similarity: $\mathrm{SIM}(A,B) := \frac{A \cap B}{A \cup B}$



4 common

18 total

SIM(A,B) = 4/18 = 2/9

Similarity-Preserving Signatures

Even sparse, the sets of words, shingles or users/ratings are too big to handle efficiently.

Goal: compute a "signature" for each set, so that similar documents have similar signatures (and dissimilar docs are unlikely to have similar signatures). (Note: "hashes" are one type of signature)

Trade-off: length of signature vs. accuracy

Could we use **cryptographic signatures?**

Characteristic Matrix of Sets

Element num	Set1	Set2	Set3	Set4
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
…				

Stored as a sparse matrix in practice.

Example from "Mining of Massive Datasets" book by Rajaraman and Ullman

Minhashing

Minhash(π) of a set is the number of the row (element) with first non-zero in the **permuted order π**.

Element num	Set1	Set2	Set3	Set4
1	0	0	1	0
4	0	0	1	0
0	1	0	0	1
3	1	0	1	1
2	0	1	0	1
…				

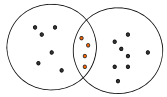
ΠΠ=(1,4,0,3,2)

Example from "Mining of Massive Datasets" book by Rajaraman and Ullman

Minhash and Jaccard similarity

Theorem:

P(minhash(S) = minhash(T)) = SIM(S,T)



Proof:

X = rows with 1 for both S and T

Y = rows with either S or T have 1, but not both

Z = rows with both 0

Probability that row of type X is before type Y in a random permuted order is _____

Minhash signature

Let h_1, h_2, …, h_n be different minhash functions (i.e., independent permutations).

Then **signature** for set S is:

$SIG(S) = [h_1(S), h_2(S), \ldots, h_n(S)]$

Now how to compute estimate of the Jaccard similarity between S and T using minhash-signatures?

SIM(S,T) ≈ fraction of coordinates where SIG(S) and SIG(T) are the same

Approximating Minhashes

Storing the hash function is a pain
(storing huge permutations is infeasible).

Solution: use a **good hash function**
that maps rows to positions.
If the rows mapped to distinct positions,
perhaps behaves like random permutation.

Properties of random hashes?
We assume the # collisions is small vs. number of items.

Algorithm

Pick n independent hash functions.

For each row r = 0, 1, …, N-1 of the characteristic matrix:

1. Compute $h_1(r), h_2(r), \ldots, h_n(r)$
2. For each column c:
   1. If column c has 0 in row r, do nothing
   2. Otherwise, for each i = 1,2, …, n set
      $SIG(i, c) = \min(h_i(r), SIG(i, c))$

Note: in practice we need to only iterate through the non-zero elements.

Worked example (on blackboard)

Element num	Set1	Set2	Set3	Set4	x + 1 mod 5	3x +1 mod 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
…						

Signature matrix

	Set1	Set2	Set3	Set4
H1	∞	∞	∞	∞
H2	∞	∞	∞	∞

Worked example (on blackboard)

Element num	Set1	Set2	Set3	Set4	x + 1 mod 5	3x +1 mod 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
…						

Signature matrix

	Set1	Set2	Set3	Set4
H1	1	3	0	1
H2	0	2	0	0

LOCALITY SENSITIVE HASHING USING MINHASH

Nearest Neighbors

Assume that we construct a 1,000 byte minhash signature for each document.

Million documents can now fit into 1 gigabyte of RAM.

But how much does it cost to find the nearest neighbor of a document?

- Brute force: N signature-signature matches.

(Closest pair takes N^2 time.)

→ Need a way to reduce the number of comparisons.

LSH requirements

A good LSH hash function will divide input into large number of **buckets.**

To find nearest neighbors for a query item q, we want to only compare with items in the bucket hash(q): "candidates".

If two A and B are similar, we want the probability that hash(A) = hash(B) be high.

- *False positives*: sets that are not similar, but are hashed into same bucket.
- *False negatives*: sets that are similar, but hashed into different buckets.

LSH based on minhash

(do not get confused about the different "hashes")

Idea:

divide the signature matrix rows into **b** bands of **r** rows

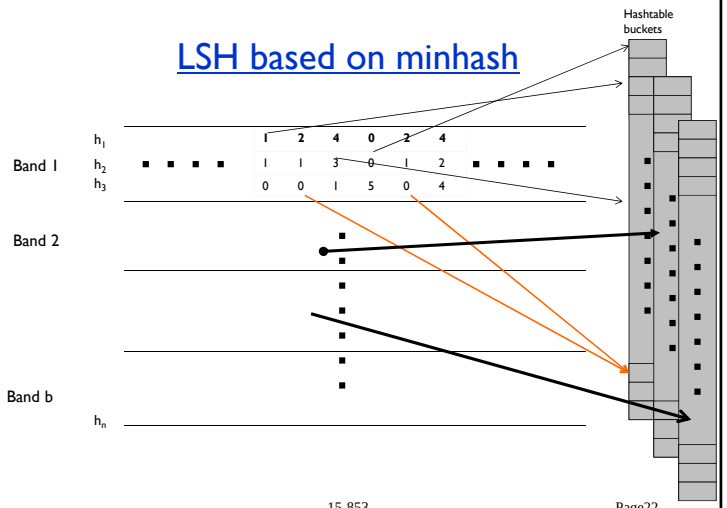
hash the columns in each band with a basic hash-function

→ each band divided to buckets [i.e., a hashtable for each band]

If sets S and T have same values in a band,
they will be hashed into the same bucket in that band.

For nearest-neighbor, the candidates are
the items in the same bucket as query item, in each band.

LSH based on minhash



Analysis

Consider the probability that we find T with query document Q

Let

s = SIM(Q,T) = P{ $h_i(Q) = h_i(T)$ }

b = # of bands

r = # rows in one band

What is the probability that rows of signature matrix agree for columns Q and T in one band?

Analysis

s = SIM(Q,T)
b = # of bands
r = # rows in one band

Probability that Q and T agree on all rows in a band

s^r

Probability that disagree on at least one row

$1 - s^r$

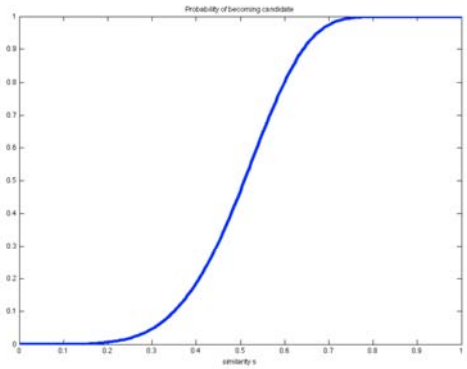
Probability that signatures do not agree on any of the bands:

$(1 - s^r)^b$

Probability that T will be chosen as candidate: ____

S-curve

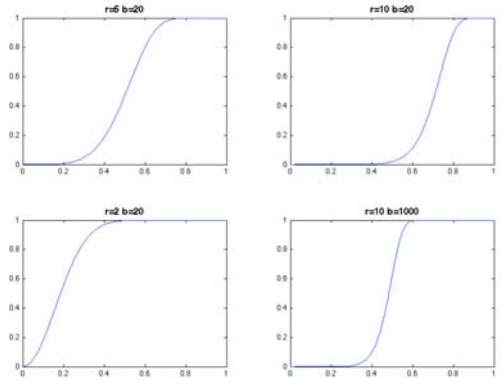
r = 5
b = 20



Approx. value of the threshold: (1/b)^{1/r}

S-curves

r and b are parameters of the system: trade-offs?



Summary

To build a system that quickly finds similar documents from a corpus:

1. Decide a vector presentation of your documents (bag of words, shingles, etc…)
2. Generate minhash signature matrix for the corpus.
3. Divide signature matrix into bands
4. Store each band-column into a hashtable
5. To find similar documents, compare to candidate documents for each band only in the same bucket (using minhash signatures or the docs themselves) .

More About Locality Sensitive Hashing

Active research area.

Different distance metrics and compatible locality sensitive hash functions:

- Euclidean distance → random projections
- Cosine distance
- Edit distance (strings)
- Hamming distance
- Jaccard distance (= 1 – Jaccard similarity)

On the board…

(d_1, d_2, p_1, p_2)-LSH

A hash function mapping elements from some distance space d(.,.) to some set [M] such that:

If $d(x,y) \le d_1$, want $Pr[f(x) = f(y)] \ge p_1$

If $d(x,y) \ge d_2$, want $Pr[f(x) = f(y)] \le p_2$

So $d_1 < d_2$, and $p_1 > p_2$. Want $p_1 >> p_2$.

Example: suppose we have a hash function such that

$Pr[f(x) = f(y)] = g(d(x,y))$

for some monotone decreasing function g()

Then we get $(d_1, d_2, g(d_1), g(d_2))$-LSH.

Hamming distance

Points: vertices of the hypercube $\{0,1\}^n$.

Distance $d_H(x,y)$ = #(coordinates x,y differ).

To use above fact, define $d(x,y) = d_H(x,y)/n$.

Hence distances are in [0,1].

Now hash function:

$h(x) = x_k$ for k chosen uniformly at random in {1..n}

Pr[h(x) = h(y)] = fraction of coordinates of x,y that agree

= 1 – d(x,y).

Cosine distance

Points: unit vectors in R^n.

Distance $d_{cos}(x,y) = 1 - \langle x,y \rangle = 1 - \cos(\text{angle})$.

Hence distances are in [0,2].

Now hash function:

$h(x) = \text{sign}(\langle x,r \rangle)$ for a uniformly random unit vector r

Pr[h(x) = h(y)]

= Pr[x,y on the same side of the hyperplane normal to r]

$= 1 - \text{angle}/\pi = 1 - \arccos(\langle x,y \rangle)/\pi$.

More About Locality Sensitive Hashing

Leskovec, Rajaraman, Ullman:

Mining of Massive Datasets (available for download)

CACM technical survey article by Andoni and Indyk

and an implementation by Alex Andoni.