Using Jaccard Coefficient

CS5154/6054

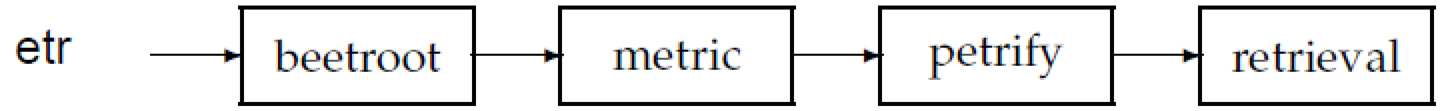
Yizong Cheng

9/1/2022

Spelling Correction using k-Gram Indexes

- Given a (possibly misspelt) word q, we can rank all words v in the dictionary against q, using some similarity measure sim(q, v) and make the highest ranked as the spelling correction.
- Words can be considered as sets of k-grams and sim(q, v) that of Jaccard coefficient.
    - A word is a sequence of characters.
    - Not just a set of characters. Same to documents, but later.
- Can be made very efficient with an inverted index over a sorted list of k-grams.
    - A k-gram is a sequence of k characters.

Postings for a 3-gram



▶ **Figure 3.4** Example of a postings list in a 3-gram index. Here the 3-gram etr is illustrated. Matching vocabulary terms are lexicographically ordered in the postings.

Example with trigrams

- Suppose the text is ***november***
  - Trigrams are *nov, ove, vem, emb, mbe, ber*.
- The query is ***december***
  - Trigrams are *dec, ece, cem, emb, mbe, ber*.
- So 3 trigrams overlap (of 6 in each term)
- How can we turn this into a normalized measure of overlap?

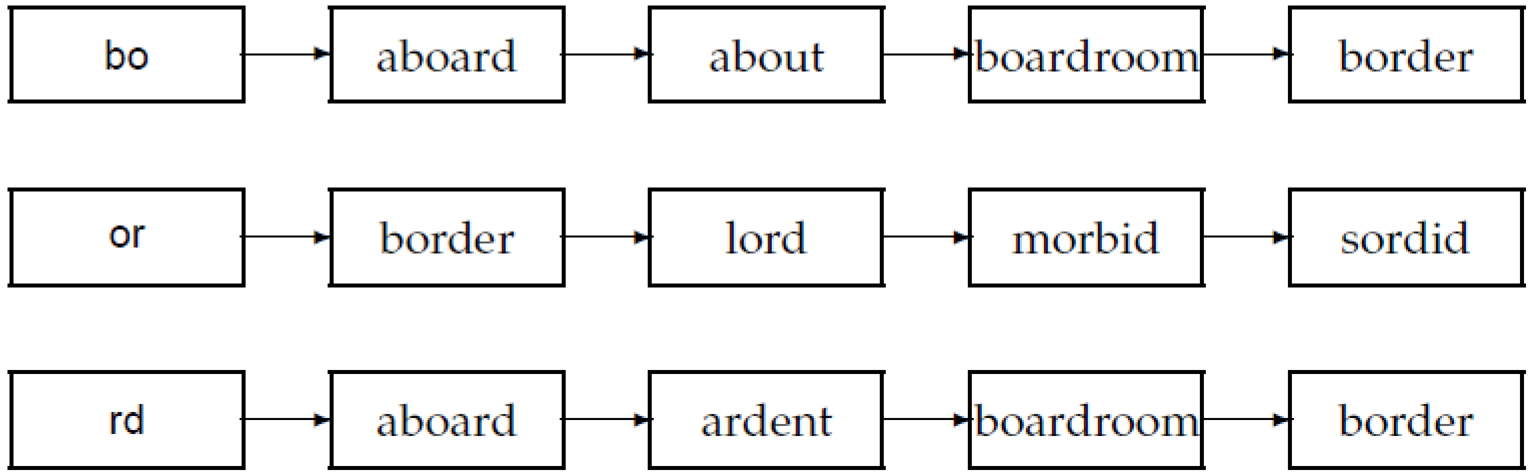
One option – Jaccard coefficient

- A commonly-used measure of overlap
- Let *X* and *Y* be two sets; then the J.C. is

$$|X \cap Y| / |X \cup Y|$$

- Equals 1 when *X* and *Y* have the same elements and zero when they are disjoint
- *X* and *Y* don’t have to be of the same size
- Always assigns a number between 0 and 1
  - Now threshold to decide if you have a match
  - E.g., if J.C. > 0.8, declare a match

Matching Bigrams



▶ **Figure 3.7** Matching at least two of the three 2-grams in the query bord.

Making/Using Inverted Index for 3-Grams

```
# IR4A.py

f = open("names5.txt", "r")
names = [line.rstrip() for line in f]
f.close()
invertedIndex = {}
for i in range(len(names)):
    name = names[i]
    grams3 = len(name) - 2
    for j in range(grams3):
        s = name[j:j+3]
        if invertedIndex.get(s) == None:
            invertedIndex.update({s : {i}})
        else:
            invertedIndex.get(s).add(i)
```

```
query = input("Enter a name with at least five characters: ").lower()
    if len(query) < 3:
        break
    grams = set()
    qlen = len(query) - 2
    for j in range(qlen):
        grams.add(query[j:j+3])
    intersections = {} # can be replaced with a Counter
    for t in grams:
        if invertedIndex.get(t) == None:
            continue
        for d in invertedIndex.get(t):  # same as IR3C.py


    for k, v in intersections.items():
            jc = v /(|A| + |B| - v)  # pseudo code
            if jc > 0.3:
                print(jc, names[k])
```

Near Duplication

- In many cases, the contents of one web page are identical to those of another except for a few characters – say, a notation showing the date and time at which the page was last modified.
- Even in such cases, we want to be able to declare the two pages to be close enough that we only index one copy.
- Short of exhaustively comparing all pairs of web pages, an infeasible task at the scale of billions of pages, how can we detect and filter out such near duplicates?
- Spelling-checking documents?
    - A document as a sequence of words … a word is a sequence of characters?

A Set of k-Shingles for a Document

- Given a positive integer k and a sequence of terms in a document d, define the k-shingles of d to be the set of all consecutive sequences of k terms in d.
- As an example, consider the following text: a rose is a rose is a rose.
- The 4-shingles for this text are a rose is a, rose is a rose and is a rose is.
    - k = 4 is a typical value used in the detection of near-duplicate web pages.
    - The first two of these shingles each occur twice in the text.
- Intuitively, two documents are near duplicates if the sets of shingles generated from them are nearly the same.

Jaccard Coefficient > 0.9 → Duplication

- The Jaccard coefficient measures the degree of overlap between two sets
- Our test for near duplication between *d*1 and *d*2 is to compute the Jaccard coefficient between their sets of shingles.
- If it exceeds a preset threshold (say, 0.9), we declare them near duplicates and eliminate one from indexing.
- Single-linkage clustering/minimum spanning tree can be used to find sets of duplicated documents and eliminate all but one of them in each set.
    - Need to find pairwise Jaccard coefficients.

Hashing and Minhash

- Let $S(dj)$ denote the set of shingles of document *dj.*
- We map every shingle into a hash value over a large space.
- let $H(dj)$ be the corresponding set of hash values derived from $S(dj)$.
- Let π be a random permutation of the hash space.
- Denote by $\Pi(dj)$ the set of permuted hash values in $H(dj)$.
  - thus for each $h \in H(dj)$, there is a corresponding value $\pi(h) \in \Pi(dj)$.
- Let x^{π}_{j} be the smallest integer in $\Pi(dj)$.
  - Called minhash.

Theorem 19.1.

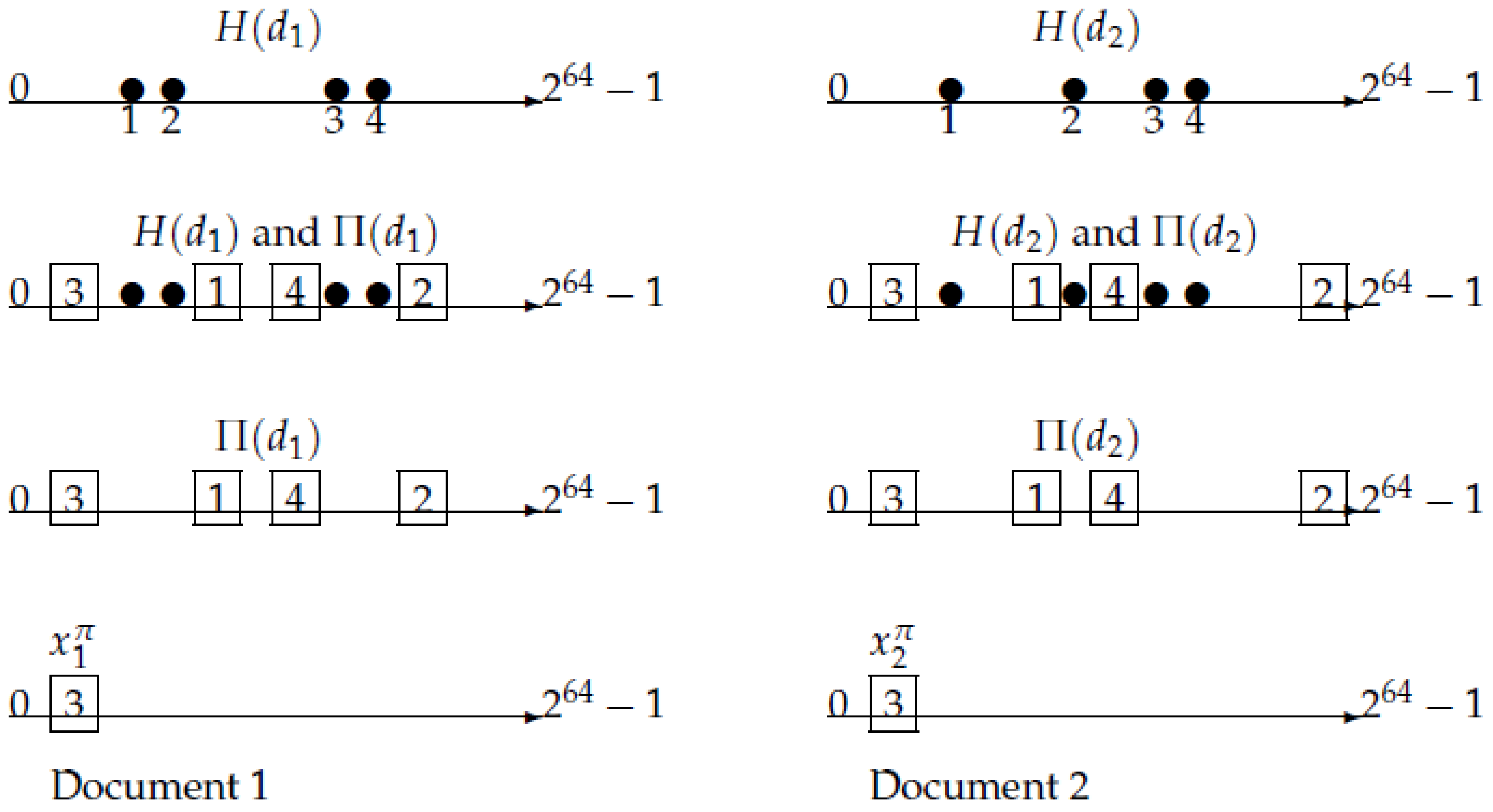
$$J(S(d_1), S(d_2)) = P(x_1^\pi = x_2^\pi).$$

S_{j_1}	S_{j_2}
0	1
1	0
1	1
0	0
1	1
0	1

▶ **Figure 19.9** Two sets S_{j_1} and S_{j_2}; their Jaccard coefficient is 2/5.

Denote by C_{00} the number of rows with 0's in both columns, C_{01} the second, C_{10} the third and C_{11} the fourth. Then,

$$J(S_{j_1}, S_{j_2}) = \frac{C_{11}}{C_{01} + C_{10} + C_{11}}. \qquad (19.2)$$


$H(d_1)$
0
1 2 3 4
$2^{64}-1$
$H(d_1)$ and $\Pi(d_1)$
0 3 1 4 2
$2^{64}-1$
$\Pi(d_1)$
0 3 1 4 2
$2^{64}-1$
x_1^π
0 3
$2^{64}-1$
Document 1
$H(d_2)$
0
1 2 3 4
$2^{64}-1$
$H(d_2)$ and $\Pi(d_2)$
0 3 1 4 2
$2^{64}-1$
$\Pi(d_2)$
0 3 1 4 2
$2^{64}-1$
x_2^π
0 3
$2^{64}-1$
Document 2

Most Popular Surnames from USA Census 2010

- Occurring more than 1000
- names5.txt
  - Length > 4
  - 22039
- names8.txt
  - Length > 7
  - 6309

smith
johnson
williams
brown
jones
garcia
miller
davis
rodriguez
martinez
hernandez
lopez
gonzalez
wilson
anderson
thomas
taylor
moore
jackson
martin
perez
thompson
white
harris

williams
rodriguez
martinez
hernandez
gonzalez
anderson
thompson
robinson
campbell
mitchell
phillips
gutierrez
peterson
richardson
castillo
sullivan
henderson
gonzales
patterson
alexander
hamilton

Near Duplicate Names?

```
def shingles(name):
    grams = set()
    for j in range(len(name) - 2):
        grams.add(name[j:j+3])
    return grams
```

```
N = len(names)
for i in range(N):
    A = shingles(names[i])
    Alen = len(A)
    for j in range(i + 1, N):
        B = shingles(names[j])
        intersection = A & B
        c = len(intersection)
        if c == 0:
            continue
        jc = c / (|A| + |B|- c)  # pseudo code
        if jc > 0.88:
            print(names[i], names[j], jc)
```