Big Data L11: Beyond Set Similarity, Spatial Search, and Graph Algorithms

1 Beyond Simple Sets: Multi-sets and Ruzicka Similarity

In many cases, it is not sufficient to know whether an item appears in a set; we must know how often it appears. These are bags or multi-sets.

1.1 Ruzicka Similarity

An extension of Jaccard similarity for bags (multi-sets). Instead of treating each element as binary (present/absent), Ruzicka compares counts.

Formula:

$$R(A, B) = \frac{\sum_i \min(A[i], B[i])}{\sum_j \max(A[j], B[j])} \tag{1}$$

where:

- $A[i]$, $B[i]$ are the counts of item i in bags A and B.

Example: Given

$$X = [1, 3, 5, 7], Y = [2, 3, 1, 6] \tag{2}$$

Compute:

$$\sum \min(X, Y) = 1 + 3 + 1 + 6 = 11 \tag{3}$$

$$\sum \max(X, Y) = 2 + 3 + 5 + 7 = 17 \tag{4}$$

Thus:

$$R(X, Y) = \frac{11}{17} \approx 0.65 \tag{5}$$

2 Beyond Sets: Spatial Similarity Search (Vector Databases)

In modern applications (e.g., document embeddings), objects are represented as high-dimensional vectors. Cosine similarity is often used to measure closeness.

Cosine Similarity Formula:

$$\text{cosine similarity}(a, b) = \frac{a^T b}{\|a\|\|b\|} \tag{6}$$

Important Cases:

- $\cos(0°) = 1$ (same direction, highly similar)
- $\cos(90°) = 0$ (orthogonal, unrelated)
- $\cos(180°) = -1$ (opposite directions)

3 Locality Sensitive Hashing (LSH) for Cosine Similarity

Motivation: Computing cosine similarity between billions of vectors ($O(Nd)$) is expensive.
LSH reduces the search space by hashing similar vectors into the same bucket.

3.1 Charikar's LSH (2002):

Idea:

- Pick a random hyperplane (random vector w).
- Define hash function:

$$h_w(x) = \begin{cases} 1, & \text{if } w^T x \geq 0 \\ -1, & \text{otherwise} \end{cases} \tag{7}$$

Collision Probability:

- Probability that two vectors u, v hash to the same value:

$$\Pr[h(u) = h(v)] = 1 - \frac{\theta}{\pi} \tag{8}$$

where θ is the angle between u and v.
More aligned vectors (small θ) are more likely to collide.
Multiple Projections:

- Use m random hyperplanes to improve precision.
- Probability of at least one collision:

$$1 - \left(\frac{\theta}{\pi}\right)^m \tag{9}$$

4 Relevance from Graph Structure: PageRank

Early search engines ranked documents based on text match alone, vulnerable to spam attacks.
PageRank introduced network structure into search relevance:

- Trusted pages are those with many incoming links from other trusted pages.

5 PageRank Model: The Random Surfer

Treat the web as a directed graph:

- Nodes = web pages
- Edges = hyperlinks

Model user behavior as a random walk:

$$P[\text{next page } v \mid \text{current page } u] = \frac{1}{\text{out-degree}(u)} \tag{10}$$

Goal: Find the steady-state distribution over pages.

6 Mathematical Foundation: Markov Chains

Matrix M:

$$M[v, u] = P[\text{go to } v \mid \text{at } u] \tag{11}$$

M is stochastic: non-negative, columns sum to 1.
Steady-state vector p:

$$p = Mp \tag{12}$$

p is the eigenvector of M with eigenvalue 1.

7 Computing PageRank: Power Iteration

For large graphs:

- Initialize $p^{(0)}$ to uniform.
- Iterate:

$$p^{(k)} = Mp^{(k-1)} \tag{13}$$

until convergence.
This is efficient and parallelizable.

8 Handling Problems in Graphs

8.1 Disconnected Graphs

- No unique steady-state.
- Solution: Ensure graph is strongly connected.

8.2 Sinks (no outgoing links)

- Sink nodes cause probability "leaks."
- Fix: Add self-loops to sinks.

8.3 Spider Traps

- A group of pages that only link to each other traps the surfer.
- Solution: Teleportation.

9 Teleportation / Random Restart (Google's Trick)

Adjust transition matrix:

$$M' = aM + (1 - a)\frac{1}{N}ee^T \tag{14}$$

where:

- $a \approx 0.85$ (follow links with 85% probability),
- With probability $1 - a$, jump randomly to any page.

Prevents sinks and spider traps from breaking PageRank.

10 Personalized PageRank

Instead of jumping uniformly:

- Jump according to a user preference distribution q.
- Modified iteration:

$$p = aMp + (1 - a)q \tag{15}$$

Applications:

- Personalizing search results.
- Recommender systems tailored to user interests.

11 Distributed PageRank with Spark

PageRank computations at web scale use Spark:

- Leverage matrix multiplication via MapReduce.
- GraphX and GraphFrames simplify distributed graph operations.

12 Summary

Concept	**Key Idea**
Ruzicka Similarity	Extend Jaccard to bags by comparing counts
Cosine Similarity	Measure angle between vectors
LSH	Approximate nearest neighbors via random hyperplanes
PageRank	Importance from network links, not just content
Power Iteration	Efficiently find steady-state distribution
Teleportation	Fix sinks and spider traps
Personalized PageRank	Incorporate user preferences into ranking