Searching in Sparse Spaces

via Random-Projection Pivots

Background

- Sparse vector spaces
  - vectors $v = (v_1, \ldots, v_M)$ of dimensionality M where just "**a few**" dimensions are **non-zero**
  - typically represented by a **list of pairs** *(dim, value)* for the non-zero dimensions

    $v = \{ (d_i, v_i) \}$

- (Dis)similarity measures
  - common feature of majority of commonly used measures *sim(u, v)* :

    They consider only those **dimensions** that are **non-zero in both** vectors *u* and *v*
    - e.g. all measures that are based on **inner product** (doc product)

Example of Spaces: IR

- There are many such spaces widely used in Information Retrieval (IR)

t - a *term* from a dictionary of size M

d - *document* (sequence of terms)

q - *query* (sequence of terms)

$\mathrm{tf}_{t,d}$ - *term frequency* (number of occurrences of term t in document d)

df_t - *document frequency* of term t (number of documents containing term t)

$\mathrm{idf}_t = \log \frac{N}{\mathrm{df}_t}$ - *inverted document frequency*

$\text{tf-idf}_{t,d} = \mathrm{tf}_{t,d} \times \mathrm{idf}_t$ - a way to weight terms t in documents (and query)

IR definitions & notation from: *C. D. Manning, P. Raghavan, and H. Schütze, Introduction to information retrieval. Cambridge University Press, 2008.*

Vector Space Model in IR

Vector space model - **documents** and **queries** represented as **sparse vectors**

- ...of tf-idf scores (and variants, see below)

- Similarity by cosine $$\mathrm{sim}(d_1, d_2) = \frac{\vec{V}(d_1) \cdot \vec{V}(d_2)}{|\vec{V}(d_1)||\vec{V}(d_2)|}$$

    - dot product: $\vec{x} \cdot \vec{y} = \sum_{i=1}^{M} x_i y_i$

    - Euclidean length: $\sqrt{\sum_{i=1}^{M} \vec{V}_i^2(d)}$

$$\vec{V}(d) = \vec{V}_1(d) \ldots \vec{V}_M(d)$$

- For normalized vectors

$$\mathrm{sim}(d_1, d_2) = \vec{v}(d_1) \cdot \vec{v}(d_2)$$

$$\vec{v}(d_1) = \vec{V}(d_1) / |\vec{V}(d_1)|$$

Variants of tf-idf Scoring

Term frequency		Document frequency		Normalization	
n (natural)	$\mathrm{tf}_{t,d}$	n (no)	1	n (none)	1
l (logarithm)	$1 + \log(\mathrm{tf}_{t,d})$	t (idf)	$\log \frac{N}{\mathrm{df}_t}$	c (cosine)	$\frac{1}{\sqrt{w_1^2 + w_2^2 + \ldots + w_M^2}}$
a (augmented)	$0.5 + \frac{0.5 \times \mathrm{tf}_{t,d}}{\max_t(\mathrm{tf}_{t,d})}$	p (prob idf)	$\max\{0, \log \frac{N - \mathrm{df}_t}{\mathrm{df}_t}\}$	u (pivoted unique)	$1/u$ (Section 6.4.4)
b (boolean)	$\begin{cases} 1 & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$			b (byte size)	$1/\mathit{CharLength}^{\alpha}, \alpha < 1$
L (log ave)	$\frac{1 + \log(\mathrm{tf}_{t,d})}{1 + \log(\mathrm{ave}_{t \in d}(\mathrm{tf}_{t,d}))}$				

so called SMART notation: *ddd.qqq* e.g. *lnc.ltc*

but always they are sparse vectors with **(normalized) dot product**

Probabilistic Ranking Models

- Standard in modern IR
- Theory behind: Estimation of the **probability** that a document d_i **is relevant** to a query q (estimation of RSV - *Retrieval Status Value*)

  - Let us begin with a simple scoring
  - Weighting and normalization factors

This is famous **BM25** weighting scheme

$$RSV_d = \sum_{t \in q} \left[\log \frac{N}{\mathrm{df}_t} \right] \cdot \quad \mathrm{tf}_{t,d} \quad \cdot \frac{(k_3 + 1)\mathrm{tf}_{tq}}{k_3 + \mathrm{tf}_{tq}}$$

L_d and L_{ave} are length of doc *d* and average doc length, resp.
k_1, k_3 and *b* are tuning parameters

The core idea is still a "**dot product**" in a sparse space

Objectives

- Efficient and accurate **kNN** retrieval in **sparse spaces** with (dis)similarity functions based on **dot product** (non-zero dimensions)

- If the query is very short (1-3 terms), the best approach is **inverted file**
    - for every term *t*, keep a posting list of (*coef*, *docid*) of all documents containing term *t*
    - given query *q*, scan posting lists of query terms and calculate individual document scores
    - if fact, better evaluation algorithms are used: static/dynamic pruning, block-WAND, etc.

1. For longer queries (query expansion, query-by-document), other solutions might be more efficient
2. There are more complex sparse representations (+ similarities) for which the posting lists for the query terms might not contain all relevant docs
    - bridge the vocabulary mismatch: IBM Model1 coefficients, word embeddings

From IR Scores to Distance Spaces

$u \cdot v = cos(u, v) \cdot |u| \cdot |v|$

$\delta(u,v) = 1 - cos(u, v)$ - not metric (but gives the same ordering as $cos(u,v)$)

$\delta(u,v) = arccos(cos(u, v))$ - angular distance

But, we don't need triangle inequality for approximate search

... in fact, for some indexes, we **don't need** to explicitly express a **distance**

Pivoting Techniques

- Pivoting techniques:
  - pre-**select** some pivots/anchors/reference objects (reference documents)
  - calculate (dis)**similarity** between **objects** (documents/queries) and the **pivots**
  - index & search data based on the the data-pivot similarities
    - PP-Index, M-Index, MI-File, PPP-Codes, NAPP, etc.

- NAPP inverted index (Neighborhood APProximation):
  - approximate the position of object x by set $P(x)$ = K **closest pivots** to x (w/o their order)
  - given query q, the **candidate set** are all objects x s.t. $|P(x) \cap P(q)| \geq s$ (share at least s pivots)
  - build an **inverted index**: for each pivot p keep list of objects x for which $p \in P(x)$
  - use well-known IR algorithm(s) to identify the candidate set & then refine

[E. S. Tellez, E. Chavez, and G. Navarro, "Succinct nearest neighbor search," Inf. Syst. 7 (38)]

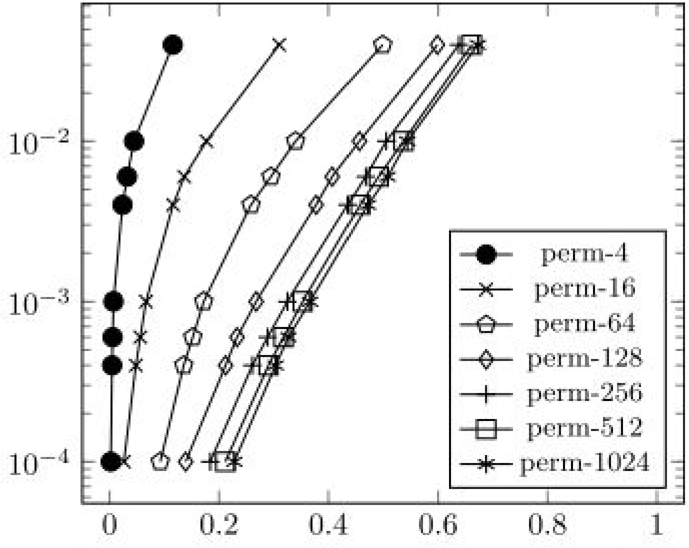
Pivoting Techniques in Sparse Spaces

Naidan, Boytsov & Nyberg tried **several** kNN approaches to **different** (non-metric) **spaces**

All tested techniques **failed** on the dataset with tf-idf scores from 4.2M Wikipedia pages with cosine similarity

[B. Naidan, L. Boytsov, and E. Nyberg, "Permutation Search Methods are Efficient, Yet Faster Search is Possible," in Proceedings of VLDB 2015]

recall of 10-NN vs. fraction of accessed objects using NAPP



(e) Wiki-sparse (cosine similarity)

Basic Insight into the Problem

- In sparse spaces, two random objects do not share many non-zero dims
- ...and if they do, these are “very common dimensions” (terms with low idf)

- Analysis of the wiki-sparse dataset (4.2M documents with tf-idf scores):
  - dictionary size *M* (dimensions): 100.000 Avg. number of non-zero dimensions: 156.0
  - number of overlapping dimensions = # of dimensions (terms) in both vectors: 3.81
  - number of overlapping dimensions after removing 1% of terms with lowest IDF: 0.95

  - analyze how many dimensions are between object and all *K* closest pivots
  - analyze *how much energy* is between an object and its closest pivots
    - “how much of the vector” is actually used for indexing

The Idea

- **Generate** pivots so that they use more information from the data:

1. Select pivots as **long documents**
2. **Merge** (put together) random **documents**
   - the same but filtering out terms with lowest IDF
3. **k-means** and then **merge** documents in the clusters

4. **random** dimensions!
   - each **pivot** = given number of **randomly chosen dimensions** (terms from the dictionary)
     - each of these selected dimensions is set to 1 (and then the vector is normalized)
   - surprisingly good results

Already Done

- **first** analysis and experiments
- proposal of an efficient **structure** to calculate similarity between an object and *all pivots at once*
    - keep a hash map for all dimensions
    - for each dimension, store a list of pivots that contain this dimension
    - given an object vector, iterate over its non-zero dimensions in the map and accumulate the similarity to all pivots
- efficient implementation within Leo & Bileg NMSLIB C++ library
- experiments and results used in

  [1] L. Boytsov, D. Novak, Y. Malkov, and E. Nyberg, "Off the Beaten Path : Let ' s Replace Term-Based Retrieval with k-NN Search," in CIKM, 2016, pp. 1099–1108.
    - without any details about the pivot-selection technique

What's Next?

- Theory:
  - **estimate** the number of dimensions shared between a vector and pivot/pivots/closest pivots
  - **derive** probability-based **properties** why our approach works :-)
  - **confirm** these estimations experimentally
- Representations - find out how much information we loose when
  - we transform sparse vectors to full (our) pivot rankings + Spearman footrule (Kendall)
  - we transform to pivot ranking prefixes + modified Spearman (M-Index style)
  - we forget the order of the pivot ranking + Jaccard coefficient (NAPP style)

What’s Next?

- Effectiveness/efficiency experiments:
    - 3 (or 4) datasets, 2-3 representations (similarity spaces), queries of different lengths
- Baselines to compare with:
    - Lucene-like inverted index (different optimizations like WAND, block WAND, etc.)
    - Falconn: current best LSH technique for cosine distance
      [Andoni, Indyk et al. Practical and Optimal LSH for Angular Distance. NIPS 2015]
    - k-NN graph (proximity graph): namely, SW-graph

Advantage of our approach: “works” for any (dis)similarity

- indexing is based directly on the similarity function

- Write and submit a paper

Thanks for your attention. Questions, comments, please...