Today’s topics

• Computing cosine-based ranking
• Speeding up cosine ranking
  – reducing the number of cosine computations
    • Union of term-wise candidates
    • Sampling and pre-grouping
  – reducing the number of dimensions
    • Random projection
    • Latent semantic indexing
Recall doc as vector

• Each doc $j$ is a vector of $tf \times idf$ values, one component for each term.
• Can normalize to unit length.
• So we have a vector space
  – terms are axes
  – docs live in this space
  – even with stemming, may have 10000+ dimensions
Intuition

Postulate: Documents that are “close together” in vector space talk about the same things.
Cosine similarity

Cosine similarity of $D_j, D_k$:

$$sim(D_j, D_k) = \sum_{i=1}^{m} w_{ij} \times w_{ik}$$

Aka normalized inner product

Can also compute cosine similarity from a query (vector of terms, e.g., *truth forever*) to each document.
Exercises

• How would you augment the inverted index built in lectures 1-3 to support cosine ranking computations?
• Walk through the steps of serving a query.
Why use vector spaces?

- **Key**: A user’s query can be viewed as a (very) short document.
- Query becomes a vector in the same space as the docs.
- Can measure each doc’s cosine proximity to query → ranking.
Efficient cosine ranking

• Ranking consists of computing the $k$ docs in the corpus “nearest” to the query $\Rightarrow k$ largest query-doc cosines.

• Efficient ranking:
  – Computing a single cosine efficiently.
  – Choosing the $k$ largest cosine values efficiently.
Computing a single cosine

• For every term $i$, with each doc $j$, store term frequency $tf_{ij}$.
  – Tradeoffs on whether to store term count, freq, or weighted by $idf_i$. (Coding possibilities.)

• Accumulate component-wise sum

$$sim(D_j, D_k) = \sum_{i=1}^{m} w_{ij} \times w_{ik}$$

More on speeding up a single cosine, later in this lecture.
Computing the $k$ largest cosines: selection vs. sorting

- Typically we want to retrieve the top $k$ docs (in the cosine ranking for the query)
  - not totally order all docs in the corpus
  - just pick off docs with $k$ highest cosines.
Use heap for selecting top $k$

- Binary tree in which each node’s value > values of children
- Takes $2n$ operations to construct, then each of $k\log n$ “winners” read off in $2\log n$ steps.
- For $n=1M$, $k=100$, this is about 10% of the cost of sorting.
Bottleneck

- Still need to first compute cosines from query to each of $n$ docs → several seconds for $n=1M$.
- Can select from only non-zero cosines; should be $<< 1M$. 
Can we avoid this?

- Yes, but may occasionally get an answer wrong
  - a doc *not* in the top $k$ may creep into the answer.
Term-wise candidates

• **Preprocess:** Pre-compute, for each term, its $k$ nearest docs.
  – (Treat each term as a 1-term query.)
  – lots of preprocessing.
  – Result: “preferred list” for each term.

• **Search:**
  – For a $t$-term query, take the union of their $t$ preferred lists - call this set $S$.
  – Compute cosines from the query to only the docs in $S$, and choose top $k$. 
Exercises

• Fill in the details of the calculation:
  – Which docs go into the preferred list for a term?

• Devise a small example where this method gives an incorrect ranking.
Sampling and pre-grouping

• First run a pre-processing phase:
  – pick $\sqrt{n}$ docs at random: call these leaders
  – For each other doc, pre-compute nearest leader
    • Docs attached to a leader: its followers;
    • Likely: each leader has $\sim \sqrt{n}$ followers.

• Process a query as follows:
  – Given query $Q$, find its nearest leader $L$.
  – Seek $k$ nearest docs from among $L$’s followers.
Visualization

Leader

Follower

Query
Why use random sampling

• Fast
• Leaders reflect data distribution
General variants

• Have each follower attached to $a=3$ (say) nearest leaders.
• From query, find $b=4$ (say) nearest leaders and their followers.
• Can recur on leader/follower construction.
Exercises

- To find the nearest leader in step 1, how many cosine computations do we do?
- What is the effect of the constants $a, b$ on the previous slide?
- Devise an example where this is likely to fail - we miss one of the $k$ nearest docs.
  - Likely under random sampling.
Dimensionality reduction

• What if we could take our vectors and “pack” them into fewer dimensions (say 10000→100) while preserving distances?
• (Well, almost.)
  – Speeds up cosine computations.
• Two methods:
  – Random projection.
  – “Latent semantic indexing”.
Random projection onto $k \ll m$ axes.

- Choose a random direction $x_1$ in the vector space.
- For $i = 2$ to $k$,
  - Choose a random direction $x_i$ that is orthogonal to $x_1, x_2, \ldots, x_{i-1}$.
- Project each doc vector into the subspace $x_1, x_2, \ldots, x_k$. 
E.g., from 3 to 2 dimensions

$x_1$ is a random direction in $(t_1,t_2,t_3)$ space.
$x_2$ is chosen randomly but orthogonal to $x_1$. 
Guarantee

- With high probability, relative distances are (approximately) preserved by projection.
- Pointer to precise theorem in Resources.
Computing the random projection

- Projecting $n$ vectors from $m$ dimensions down to $k$ dimensions:
  - Start with $m \times n$ matrix of terms $\times$ docs, $A$.
  - Find random $k \times m$ orthogonal projection matrix $R$.
  - Compute matrix product $W = R \times A$.

- $j$th column of $W$ is the vector corresponding to doc $j$, but now in $k \ll m$ dimensions.
Cost of computation

• This takes a total of $kmn$ multiplications.
• Expensive - see Resources for ways to do essentially the same thing, quicker.
• Exercise: by projecting from 10000 dimensions down to 100, are we really going to make each cosine computation faster?
Latent semantic indexing (LSI)

- Another technique for dimension reduction
- Random projection was data-independent
- LSI on the other hand is data-dependent
  - Eliminate redundant axes
  - Pull together “related” axes
    - car and automobile
Notions from linear algebra

- Matrix, vector
- Matrix transpose and product
- Rank
- Eigenvalues and eigenvectors.
Overview of LSI

• Pre-process docs using a technique from linear algebra called **Singular Value Decomposition**.

• Have control over the granularity of this process:
  – create new vector space, details to follow.

• Queries handled in this new vector space.
Singular-Value Decomposition

• Recall $m \times n$ matrix of terms $\times$ docs, $A$.
  – $A$ has rank $r \leq m,n$.

• Define term-term correlation matrix $T=AA^t$
  – $A^t$ denotes the matrix transpose of $A$.
  – $T$ is a square, symmetric $m \times m$ matrix.

• Doc-doc correlation matrix $D=A^tA$.
  – $D$ is a square, symmetric $n \times n$ matrix.
Eigenvectors

- Denote by $P$ the $m \times r$ matrix of eigenvectors of $T$.
- Denote by $R$ the $n \times r$ matrix of eigenvectors of $D$.
- It turns out $A$ can be expressed (decomposed) as $A = PQR^t$
  - $Q$ is a diagonal matrix with the eigenvalues of $AA^t$ in sorted order.
Visualization

\[
\begin{align*}
A & = \begin{array}{c}
\text{\(m\times n\)} \\
\text{\(m\times r\)} & \text{\(r\times r\)} & \text{\(r\times n\)}
\end{array} \\
\end{align*}
\]
Dimension reduction

• For some $s \ll r$, zero out all but the $s$ biggest eigenvalues in $Q$.
  – Denote by $Q_s$ this new version of $Q$.
  – Typically $s$ in the hundreds while $r$ could be in the (tens of) thousands.

• Let $A_s = P Q_s R^t$

• Turns out $A_s$ is a pretty good approximation to $A$.

We’ll explain what this means.
The columns of $A_s$ represent the docs, but in $s << m$ dimensions.
Guarantee

• Relative distances are (approximately) preserved by projection:
  – Of all $m \times n$ rank $s$ matrices, $A_s$ is the best approximation to $A$.

• Pointer to precise theorem in Resources.
Doc-doc similarities

• $A_s A_s^t$ is a matrix of doc-doc similarities:
  – the $(j,k)$ entry is a measure of the similarity of doc $j$ to doc $k$. 
Semi-precise intuition

• We accomplish more than dimension reduction here:
  – Docs with lots of overlapping terms stay together
  – Terms from these docs also get pulled together.

• Thus *car* and *automobile* get pulled together because both co-occur in docs with *tires, radiator, cylinder*, etc.
Query processing

• View a query as a (short) doc:
  – call it row 0 of $A_s$.

• Now the entries in row 0 of $A_s A_s^t$ give the similarities of the query with each doc.

• Entry $(0,j)$ is the score of doc $j$ on the query.

• Exercise: fill in the details of scoring/ranking.
Resources

• Random projection theorem:  
  [http://citeseer.nj.nec.com/dasgupta99elementary.html](http://citeseer.nj.nec.com/dasgupta99elementary.html)

• Faster random projection:  
  [http://citeseer.nj.nec.com/frieze98fast.html](http://citeseer.nj.nec.com/frieze98fast.html)

• Latent semantic indexing:  
  [http://citeseer.nj.nec.com/deerwester90indexing.html](http://citeseer.nj.nec.com/deerwester90indexing.html)

• Books: MG 4.6, MIR 2.7.2.