Challenges

1. Capacity:
   ▶ Collections too large for manual indexing.
2. Consistency:
   ▶ Output of keyword extraction depends on the indexer.
3. Synonymy:
   ▶ Several words with the same meaning.
4. Polysemy:
   ▶ One word with several meanings.
5. Term weighting:
   ▶ Some words carry more information than others.
   ▶ Information content depends on the context.

Evaluation of information retrieval systems

- Recall: \[
\frac{\text{# retrieved relevant documents}}{\text{# relevant documents}}\]
- Precision: \[
\frac{\text{# retrieved relevant documents}}{\text{# retrieved documents}}\]

- Both ratings should be close to 1.
The vector space model

1. “Bag of words” approach:
   - Rely only on word frequencies.
   - Ignore other language structures (sentences, paragraphs, etc.).
2. Each document represented by a vector \( \mathbf{a} \).
3. Entry \( \mathbf{a}_i \) represents how important term \( i \) is in describing the content of the document.
4. Gather \( d \) documents with \( t \) terms into a \( t \times d \) term-by-document matrix
   \[
   \mathbf{A} = \begin{bmatrix}
   \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_d
   \end{bmatrix}.
   \]
5. The columns of \( \mathbf{A} \) are called document vectors.
6. The rows of \( \mathbf{A} \) are called term vectors.

Vector space representation

Example

- Documents:
  - D1 How to Bake Bread without Recipes
  - D2 The Classic Art of Viennese Pastry
  - D3 Numerical Recipes: The Art of Scientific Computing
  - D4 Breads, Pastries, Pies and Cakes: Quantity Baking Recipes
  - D5 Pastry: A Book of Best French Recipes

- Term-by-document matrix

\[
\mathbf{A} = \begin{bmatrix}
1 & 0 & 0 & 1 & 0 \\
1 & 0 & 1 & 1 & 1 \\
1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 0
\end{bmatrix}
\]

The key idea of the vector space model

Interpret documents as vectors and use geometry to measure similarity.

- Recall the cosine formula:
  \[
  \cos \theta = \frac{\mathbf{x}^T \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|},
  \]
  where \( \theta \) is the angle between \( \mathbf{x} \) and \( \mathbf{y} \).
- Note: In the special case that \( \|\mathbf{x}\| = \|\mathbf{y}\| = 1 \), the formula simplifies to
  \[
  \cos \theta = \mathbf{x}^T \mathbf{y}.
  \]
Weighting of matrix elements

\[ a_{ij} = \ell_{ij} g_i \]

1. \( \ell_{ij} \): local weight of term \( i \) in document \( j \)
   - Frequently occurring terms should receive a large weight...
   - ...but not too large to drown out less frequent words.
2. \( g_i \): global weight of term \( i \) in the collection
   - Information-rich words should receive a large weight...
   - ...but not if they occur in most of the documents.

Example: The popular log-entropy term weighting scheme chooses

\[ \ell_{ij} = \log_2 (1 + f_{ij}), \quad g_i = 1 + \sum_k p_{ik} \log_2 \frac{p_{ik}}{\log_2 d}, \]

where \( f_{ij} \) is the term frequency of term \( i \) in document \( j \) and

\[ p_{ij} = \frac{f_{ij}}{\sum_k f_{ik}} \]

is large if the term occurs mostly in document \( j \).

#### Example

**Documents:** Wikipedia pages for
- D1 Coffee
- D2 Gevalia
- D3 Java (island)
- D4 Java (coffee)
- D5 Java (language)
- D6 Starbucks

**Term frequency of top 10 words**

\[
\begin{bmatrix}
244 & 9 & 2 & 15 & 0 & 78 \\
4 & 0 & 0 & 0 & 85 \\
23 & 0 & 0 & 0 & 41 \\
52 & 0 & 0 & 0 & 0 \\
3 & 3 & 0 & 0 & 144 \\
9 & 0 & 1 & 0 & 34 \\
0 & 0 & 0 & 43 & 0 \\
8 & 0 & 1 & 17 & 9 \\
22 & 0 & 0 & 0 & 12 \\
20 & 0 & 4 & 0 & 5
\end{bmatrix}
\]

**Log-term frequency of top 10 words**

\[
L = \begin{bmatrix}
7.94 & 3.32 & 1.58 & 4 & 0 & 6.30 \\
2.32 & 0 & 0 & 0 & 0 & 6.43 \\
4.58 & 0 & 0 & 5.39 & 1 & \\
5.73 & 0 & 0 & 0 & 0 & \\
2 & 2 & 0 & 0 & 1 & 5.49 \\
3.32 & 0 & 1 & 0 & 2 & 5.13 \\
0 & 0 & 0 & 5.46 & 0 & \\
3.17 & 0 & 1 & 1 & 4.17 & 3.32 \\
4.52 & 0 & 0 & 0 & 3.70 & \\
4.39 & 0 & 2.32 & 0 & 2 & 2.58
\end{bmatrix}
\]

**Document-relative frequency of top 10 words**

\[
P = \begin{bmatrix}
0.70 & 0.03 & 0.01 & 0.04 & 0 & 0.22 \\
0.04 & 0 & 0 & 0 & 0 & 0.96 \\
0.35 & 0 & 0 & 0 & 0.63 & 0.02 \\
1 & 0 & 0 & 0 & 0 & 0 \\
0.06 & 0.06 & 0 & 0 & 0.02 & 0.86 \\
0.19 & 0 & 0.02 & 0 & 0.06 & 0.72 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0.22 & 0 & 0.03 & 0.03 & 0.47 & 0.25 \\
0.65 & 0 & 0 & 0 & 0 & 0.35 \\
0.62 & 0 & 0.12 & 0 & 0.09 & 0.16
\end{bmatrix}
\]

\[ g = \begin{bmatrix}
0.53 \\
0.90 \\
0.60 \\
1 \\
0.70 \\
0.55 \\
1 \\
0.31 \\
0.64 \\
0.41
\end{bmatrix}
\]
Weighting of matrix elements

Example

<table>
<thead>
<tr>
<th>Coffee</th>
<th>Gevalia</th>
<th>Java (island)</th>
<th>Java (coffee)</th>
<th>Java (language)</th>
<th>Starbucks</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.20</td>
<td>1.76</td>
<td>0.84</td>
<td>2.12</td>
<td>0</td>
<td>3.34</td>
</tr>
<tr>
<td>2.08</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5.77</td>
<td></td>
</tr>
<tr>
<td>2.74</td>
<td>0</td>
<td>0</td>
<td>3.22</td>
<td>0.60</td>
<td></td>
</tr>
<tr>
<td>5.73</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1.40</td>
<td>1.40</td>
<td>0</td>
<td>0.70</td>
<td>3.84</td>
<td></td>
</tr>
<tr>
<td>1.82</td>
<td>0.55</td>
<td>0</td>
<td>1.10</td>
<td>2.82</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5.46</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.99</td>
<td>0.31</td>
<td>0.31</td>
<td>1.30</td>
<td>1.03</td>
<td></td>
</tr>
<tr>
<td>2.88</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2.36</td>
<td></td>
</tr>
<tr>
<td>1.78</td>
<td>0.94</td>
<td>0.81</td>
<td>1.05</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$A = \text{diag}(g)L = \begin{bmatrix}
4.20 & 1.76 & 0.84 & 2.12 & 0 & 3.34 \\
2.08 & 0 & 0 & 0 & 5.77 & \\
2.74 & 0 & 0 & 3.22 & 0.60 & \\
5.73 & 0 & 0 & 0 & 0 & \\
1.40 & 1.40 & 0 & 0.70 & 3.84 & \\
1.82 & 0.55 & 0 & 1.10 & 2.82 & \\
0 & 0 & 0 & 5.46 & 0 & \\
0.99 & 0.31 & 0.31 & 1.30 & 1.03 & \\
2.88 & 0 & 0 & 0 & 2.36 & \\
1.78 & 0.94 & 0.81 & 1.05 & 
\end{bmatrix}$

Query matching

- Model a query $q$ as a $t$-vector (document vector).
- Compute for each document $j$

$$\cos \theta_j = \frac{a_j^T q}{\|a_j\| \|q\|}.$$  

- Note that $a_j$ and $q$ are generally very sparse.
- A value of the cosine similarity $\cos \theta_j$ close to 1 means that document $j$ is relevant to the query.
- Set threshold $\epsilon$ to make a binary decision of relevance for each document:

$$(\text{document } j \text{ is relevant}) \leftrightarrow \cos \theta_j \geq \epsilon.$$
Query matching

Example

\[
\cos \theta_j = (\tilde{A}^T \tilde{q})^T = \begin{bmatrix}
0.09 & 0.24 & 0.03 & 0.28 & 0 & 0.08 \\
0.08 & 0.31 & 0.02 & 0.19 & 0.01 & 0.12 \\
0.09 & 0.17 & 0.02 & 0.19 & 0 & 0.15 \\
0.06 & 0 & 0 & 0 & 0.09 & 0.01 \\
0.04 & 0 & 0 & 0 & 0.17 & 0.01 \\
\end{bmatrix}
\]

Low-rank approximation

1. Approximate \( A \) by

\[
\tilde{A} \approx A
\]

such that

\[
k := \text{rank}(\tilde{A}) \ll \text{rank}(A).
\]

2. **Important:** Work with \( \tilde{A} \) in factored form, e.g.

\[
\tilde{A}_{m \times n} = B_{m \times k} C^T_{k \times n}
\]

where \( B \) and \( C \) have only \( k \) columns.

Low-rank approximation

Motivations:

1. Data compression.
2. Computational efficiency.
3. Improved performance (recall and precision).
4. Identification of topics and concepts.

Justifications:

1. \( A \) is often rank-deficient in practice
2. \( A \) and \( A + E \) can be considered equally good representations if, e.g.,

\[
\frac{\|E\|}{\|A\|} \leq 20\%
\]

due to, e.g., inconsistent indexing.
3. \( A + E \) might have much lower rank than \( A \).
QR-based low-rank approximation

**Query matching with** $A = Q_AR_A$

Note that

$$a_j^T q = (Ae_j)^T q = (Q_AR_Ae_j)^T q = (Q_Ar_j)^T q = r_j^T (Q_A^T q) = r_j^T q,$$

where

- $r_j := R_Ae_j$ is the $j$-th column of $R_A$.
- $\tilde{q} := Q_A^T q$ has a geometric interpretation (explained next).

**Orthogonal projection**

**Note that**

$$lq = QQ^T q = [Q_A \quad Q_A^\perp] \left[ \begin{array}{c} R_A \\ 0 \end{array} \right] q = Q_A Q_A^T q + Q_A^\perp Q_A^T q = q_A + q_A^\perp.$$

- $q$ may be expressed as a sum of two orthogonal vectors:
  - $q_A$, which is in the column space $C(A)$ of $A$.
  - $q_A^\perp$, which is in $N(A^T)$, which is orthogonal to $C(A)$.
- Interpretation: The projection $Q_A Q_A^T q$ represents the part of the query that is relevant to the document collection.

**QR factorization**

- We can factor $A$ into
  $$A = QR,$$
  where $Q$ is $t \times t$ and orthogonal ($Q^T Q = I$), and $R$ is upper trapezoidal ("rectangular and upper triangular").
- In particular, for "tall-and-narrow" matrices $A$ ($t \geq d$) we have
  $$A = [Q_A \quad Q_A^\perp] \left[ \begin{array}{c} R_A \\ 0 \end{array} \right] = Q_AR_A,$$
  which means that $Q_A$ is an orthonormal basis for $C(A)$. 
Geometric interpretation and similarity

- Cosine similarity with \( A \):
  \[
  \cos \theta_j = \frac{a_j^T q}{\|a_j\| \|q\|}.
  \]

- Cosine similarity with \( A = Q_A R_A \):
  \[
  \cos \theta_j = \frac{r_j^T q}{\|r_j\| \|q\|} < \frac{r_j^T \tilde{q}}{\|r_j\| \|\tilde{q}\|} =: \cos \theta_j',
  \]
  where \( \tilde{q} = Q_A^T q \).

- In words, a price we pay when using the rank-\( k \) approximation for matching, is that a document may look more relevant than it really is.

- This is referred to as an “increase in recall at the risk of reduced precision”.

Query matching with \( A \approx Q_A R_A \)

1. Define \( Q_A := Q_1 \) and \( R_A := [R_{11} \ R_{12}] \ P^T \).
2. Apply the same derivation as before.
3. Also apply the same geometric interpretation.

QR-based low-rank approximation

- Using QR with column pivoting, we can compute
  \[
  AP = QR = [Q_1 \ \ Q_2] \begin{bmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{bmatrix}
  \]
  such that:
  \[
  \begin{align*}
  &P \text{ is a permutation.} \\
  &R_{11} \text{ is square, upper triangular, and invertible.} \\
  &\|R_{22}\| \text{ is “small” compared to } \|R\| \text{ (and } \|A\|). \\
  \end{align*}
  \]

- Define the approximation
  \[
  AP + E = [Q_1 \ Q_2] \begin{bmatrix} R_{11} & R_{12} \\ 0 & 0 \end{bmatrix} = Q_1 [R_{11} \ R_{12}]
  \]
  obtained by setting \( R_{22} \) to zero.

- Note that
  \[
  \|E\|_F = \| \begin{bmatrix} 0 & 0 \\ 0 & -R_{22} \end{bmatrix} \|_F = \| R_{22} \|_F.
  \]

SVD-based low-rank approximation
SVD-based low-rank approximation

- Using the SVD decomposition, we can write
  \[ A = U \Sigma V^T = \sum_{i} U_i \sigma_i V_i^T \approx \sum_{i} U_i \sigma_i V_i^T = U_k \Sigma_k V_k^T, \]
  where \( U_k \Sigma_k V_k^T \) is the truncated SVD.
- The approximation error is given by
  \[ \|A - A_k\|_F^2 = \sum_{i=k+1}^{\min\{t,d\}} \sigma_i^2. \]

Computing an SVD-based approximation

- Note that
  \[ \|A\|_F^2 = \sum_{i=1}^{\min\{t,d\}} \sigma_i^2 \]
  \[ = \sum_{i=1}^{k} \sigma_i^2 + \sum_{i=k+1}^{\min\{t,d\}} \sigma_i^2 \]
  \[ = \sum_{i=1}^{k} \sigma_i^2 + \|A - A_k\|_F^2. \]
- Thus, we can compute the relative approximation error
  \[ \frac{\|A - A_k\|_F}{\|A\|_F} \]
  by computing \( \|A\|_F \) and the first \( k \) singular values.

Query matching with \( A \approx U_k \Sigma_k V_k^T \)

1. Define \( Q_A := U_k \) and \( R_A := \Sigma_k V_k^T \).
2. Apply the same derivation as before.
3. Also apply the same geometric interpretation.

Summary

- We define similarity via the cosine formula
  \[ \cos \theta_j := \frac{a_j^T q}{\|a_j\|_2 \|q\|_2}. \]
- When working with a rank-\( k \) approximation of the form
  \[ A \approx Q_A R_A, \]
  where \( Q_A^T Q_A = I_k \), then we use the modified formula
  \[ \cos \theta'_j := \frac{r_j^T \bar{q}}{\|r_j\|_2 \|\bar{q}\|_2}, \]
  where
  \[ r_j := R_A e_j = R_A(:,j), \quad \bar{q} := Q_A^T q. \]
Relevance feedback

1. To enhance a query, we can combine it with a relevant document.
2. Suppose document $j$ is relevant for query $q$.
3. Then we define

$$q_{\text{new}} := QA^T q + a_j$$

$$\approx QA^T q + QA R e_j$$

$$= QA(Q T q + R e_j)$$

$$= QA(q + r_j).$$

Adding new documents

- When new documents are added to the collection, we can
  2. Fold the new documents into the rank-reduced space (cheap but possibly inaccurate).
  3. Update the truncated SVD (compromise w.r.t. cost and accuracy).

1. Recompute SVD approximation

- Add the new document $\hat{p}$ as a column of $\hat{A}$

$$\hat{A} = \begin{bmatrix} \hat{A} & \hat{p} \end{bmatrix}.$$ 

- Compute new SVD approximation $U_k \Sigma_k V_k^T$ of $\hat{A}$.
2. Folding-in

- Consider the projection of the new document \( \hat{p} \) on the approximated subspace
  \[ p = U_k U_k^T \hat{p}. \]

- Append \( d = U_k^T \hat{p} \) as a new column of \( \Sigma_k V_k^T \)
  \[ V'_k = [V_k \quad \hat{p}^T U_k \Sigma_k^{-1}] . \]

- Note: \( V'_k \) is no longer orthogonal.

3. Approximate SVD updating

Adding new terms:
  \[ B := \begin{bmatrix} A_k \\ T \end{bmatrix} . \]

1st step:
- Rewrite \( B \):
  \[ B = \begin{bmatrix} U_k & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} \Sigma_k & 0 \\ TV_k & I \end{bmatrix} \begin{bmatrix} V_k \quad (I - V_k V_k^T) T^T \end{bmatrix}^T. \]

Approximate SVD updating

2nd step:
- Compute pivoted QR factorization of \( (I - V_k V_k^T) T \):
  \[ (I - V_k V_k^T) T \Pi_V = \hat{V}_k R_r. \]

3rd step:
- Rewrite \( B \):
  \[ B = \begin{bmatrix} U_k & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} \Sigma_k & 0 \\ TV_k & I \end{bmatrix} \begin{bmatrix} V_k \quad \hat{V}_k \end{bmatrix}^T. \]

Approximate SVD updating

4th step:
- Compute the truncated SVD:
  \[ \begin{bmatrix} \Sigma_k & 0 \\ TV_k & \Pi_V R_r^T \end{bmatrix} = \begin{bmatrix} P_k & P_{\perp k} \\ 0 & \hat{\Sigma}_r \end{bmatrix} \begin{bmatrix} 0 & Q_k^\perp \\ \hat{\Sigma}_k & Q_k \end{bmatrix}^T \approx P_k \hat{\Sigma}_k Q_k^T. \]

5th step:
- Identify the truncated SVD of \( B \):
  \[ B = \left( \begin{bmatrix} U_k & 0 \\ 0 & I \end{bmatrix} P_k \right) \hat{\Sigma}_k \left( \begin{bmatrix} V_k \quad \hat{V}_k \end{bmatrix} Q_k \right)^T. \]
Approximate SVD updating

Adding new documents:

\[ B := [A_k \quad D] . \]

1st step:
- Compute pivoted QR factorization of \((I - U_k U_k^T)D\):

\[ (I - U_k U_k^T)D \Pi_U = \hat{U}_k R_s . \]

2nd step:
- Form the matrix

\[ \begin{bmatrix} \Sigma_k & U_k^T D \\ 0 & R_s \Pi_U^T \end{bmatrix} \]

3rd step:
- Compute the truncated SVD:

\[ \begin{bmatrix} \Sigma_k & U_k^T D \\ 0 & R_s \Pi_U^T \end{bmatrix} \approx P_k \hat{\Sigma}_k Q_k^T . \]

Approximate SVD updating

4th step:
- Identify the truncated SVD of \(B\):

\[ B_k = ([U_k \quad \hat{U}_k] P_k) \hat{\Sigma}_k \begin{bmatrix} V_k & 0 \\ 0 & I_s \end{bmatrix} Q_k^T . \]

Final step:
- Compute the orthonormal factors

\[ U_B := [U_k \quad \hat{U}_k] P_k, \quad V_B := \begin{bmatrix} V_k & 0 \\ 0 & I_s \end{bmatrix} Q_k . \]

Approximate SVD updating

Now rewrite \(B\) as

\[ B = [A_k \quad D] = [U_k \quad \hat{U}_k] \begin{bmatrix} \Sigma_k & U_k^T D \\ 0 & R_s \Pi_U^T \end{bmatrix} \begin{bmatrix} V_k^T & 0 \\ 0 & I_s \end{bmatrix} . \]

Non-negative Matrix Factorization (NMF)
What is NMF?

1. Given a non-negative $m \times n$ matrix $A$, find non-negative matrices $W$ (of size $m \times k$) and $H$ (of size $k \times n$) such that

   $$A \approx WH.$$ 

2. More precisely, minimize the cost function

   $$f(W, H) = \frac{1}{2} \|A - WH\|_F^2$$

   over the non-negative matrices $W$ and $H$.

3. This is a non-convex constrained optimization problem and hence very difficult.

4. Many optimization methods have been proposed. We will look at the original multiplicative update formulas and a basic alternating least squares method.

Why NMF?

1. Suppose $A$ is a non-negative data matrix.

2. Then

   $$Ae_j \approx WHe_j = Wh_j$$

   implies that column $j$ of $A$ is a linear combination of the columns of $W$, with the weights given by the $j$-th column of $H$.

3. Since $W$ and $H$ are non-negative, no cancellation occurs and hence $A \approx WH$ decomposes the columns of $A$ as sums of parts.

4. The parts are given by the columns of $W$.

5. In information retrieval, the parts can be interpreted as topics.

Multiplicative update rule for NMF

Algorithm due to Lee and Seung:

1. Initialize $W$ and $H$ to random non-negative matrices
2. while not converged do
3. Update the elements of $W$ with the formula

   $$(W)_{ij} = (W)_{ij} \frac{(AH^T)_{ij}}{(WHH^T)_{ij} + \epsilon}$$

4. Normalize the columns of $W$
5. Update the elements of $H$ with the formula

   $$(H)_{ij} = (H)_{ij} \frac{(W^T A)_{ij}}{(W^T WH)_{ij} + \epsilon}$$

6. end while

Note: $\epsilon > 0$ is a small number chosen to avoid overflow.

Alternating least squares for NMF

Algorithm due to Berry et al.:

1. Initialize $W$ and $H$ to random non-negative matrices
2. while not converged do
3. Solve the linear least squares problem

   $$\min_W \|A - WH\|_F$$

4. Set all negative entries in $W$ to 0
5. Solve the linear least squares problem

   $$\min_H \|A - WH\|_F$$

6. Set all negative entries in $H$ to 0
7. end while

Note: No convergence theory.