Dealing with Text Databases

- Unstructured data
- Boolean queries
  - Sparse matrix representation
  - Inverted index
- Counts vs. frequencies
- Term frequency
- tf x idf term weights
- Documents as vectors
  - Cosine similarity
  - Dimensionality reduction
- Vectors and Boolean queries
Christopher Manning

Prabhakar Raghavan

Hinrich Schütze

Unstructured data

Which plays of Shakespeare contain the words *Brutus AND Caesar* but *NOT Calpurina*?

- (Calpurinia, third and last wife of Julius Caesar)

One could grep all of Shakespeare’s plays for *Brutus* and *Caesar*, then strip out lines containing *Calpurnia*?

- Slow (for large corpora)
- *NOT Calpurnia* is non-trivial
Term-document incidence

<table>
<thead>
<tr>
<th>Term and Document</th>
<th>Antony and Cleopatra</th>
<th>Julius Caesar</th>
<th>The Tempest</th>
<th>Hamlet</th>
<th>Othello</th>
<th>Macbeth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Antony</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Brutus</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Caesar</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Calpurnia</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Cleopatra</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mercy</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>worse</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Brutus AND Caesar but NOT Calpurnia

Incidence vectors

- So we have a 0/1 vector for each term

- To answer query: take the vectors for Brutus, Caesar and not Calpurnia (complemented) © bitwise AND

- 110100 AND 110111 AND 101111 = 100100
Answers to query
110100 AND 110111 AND 101111 = 100100

- Antony and Cleopatra
- Hamlet

<table>
<thead>
<tr>
<th></th>
<th>Antony and Cleopatra</th>
<th>Julius Caesar</th>
<th>The Tempest</th>
<th>Hamlet</th>
<th>Othello</th>
<th>Macbeth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Antony</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Brutus</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Caesar</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Calpurnia</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Cleopatra</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mercy</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>worse</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Sparse matrix representation

- For real data matrix becomes very big
- Matrix has much, much more zeros than ones
  - Matrix is extremely sparse
  - Why? Not every term (word) in every document present
- What’s a better representation?
  - We only record the 1 positions
Inverted index

- For each term $T$, we must store a list of all documents that contain $T$
- Do we use an array or a list for this?

<table>
<thead>
<tr>
<th>Term</th>
<th>List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brutus</td>
<td>2 4 8 16 32 64 128</td>
</tr>
<tr>
<td>Calpurnia</td>
<td>1 2 3 5 8 13 21 34</td>
</tr>
<tr>
<td>Caesar</td>
<td>13 16</td>
</tr>
</tbody>
</table>

What happens if the word *Caesar* is added to document 14?

Inverted index

- Linked lists generally preferred to arrays
  - Dynamic space allocation
  - Insertion of terms into documents easy
  - Space overhead of pointers
Inverted index construction

Documents to be indexed.

Token stream.

Linguistic modules

Modified tokens.

Indexer

Inverted index

Boolean queries: Exact match

- The Boolean Retrieval model is being able to ask a query that is a Boolean expression:
  - Boolean Queries are queries using AND, OR and NOT to join query terms
  - Views each document as a set of words (terms)
  - Is precise: document matches condition or not
**Exact match**

- Primary commercial retrieval tool for 3 decades

- Professional searchers (e.g., lawyers) still like Boolean queries:
  - You know exactly what you’re getting.

**Scoring**

- Our queries have all been Boolean
- Good for expert users with precise understanding of their needs and the corpus

- Not good for (the majority of) users with poor Boolean formulation of their needs
Scoring

- We wish to return in order the documents most likely to be useful to the searcher
- How can we rank order the docs in the corpus with respect to a query?
- Assign a score – say in [0,1]
  - for each doc on each query

Incidence matrices

- Recall: Document (or a zone in it) is binary vector X in \(\{0,1\}^v\)
- Query is a vector
- Score: Overlap measure: \(|X \cap Y|\)

<table>
<thead>
<tr>
<th></th>
<th>Antony and Cleopatra</th>
<th>Julius Caesar</th>
<th>The Tempest</th>
<th>Hamlet</th>
<th>Othello</th>
<th>Macbeth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Antony</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Brutus</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Caesar</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Calpurnia</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Cleopatra</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mercy</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>worser</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Example

- On the query *ides of march*, Shakespeare’s *Julius Caesar* has a score of 3
- All other Shakespeare plays have a score of 2 (because they contain *march*) or 1
- Thus in a rank order, *Julius Caesar* would come out tops

Overlap matching

- What’s wrong with the overlap measure?
- It doesn’t consider:
  - Term frequency in document
  - Term scarcity in collection (document mention frequency)
    - *of* is more common than *ides* or *march*
  - Length of documents
Scoring: density-based

- Obvious next idea: if a document talks about a topic *more*, then it is a better match
- This applies even when we only have a single query term.
- Document relevant if it has a lot of the terms
- This leads to the idea of term weighting

Term-document count matrices

- Consider the number of occurrences of a term in a document:
  - Bag of words model
  - Document is a vector in *N*: a column below

<table>
<thead>
<tr>
<th></th>
<th>Antony and Cleopatra</th>
<th>Julius Caesar</th>
<th>The Tempest</th>
<th>Hamlet</th>
<th>Othello</th>
<th>Macbeth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Antony</td>
<td>157</td>
<td>73</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Brutus</td>
<td>4</td>
<td>157</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Caesar</td>
<td>232</td>
<td>227</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Calpurnia</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Cleopatra</td>
<td>57</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mercy</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>worse</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Counts vs. frequencies

- Consider again the *ides of march* query
  - *Julius Caesar* has 5 occurrences of *ides*
  - No other play has *ides*
  - *march* occurs in over a dozen
  - All the plays contain *of*

- By this scoring measure, the top-scoring play is likely to be the one with the most *ofs*

Digression: terminology

- **WARNING**: In a lot of IR literature, “frequency” is used to mean “count”
  - Thus *term frequency* in IR literature is used to mean *number of occurrences* in a doc
  - Not divided by document length (which would actually make it a frequency)

- We will conform to this misnomer
  - In saying *term frequency* we mean the *number of occurrences* of a term in a document.
Term frequency \( tf \)

- Long docs are favored because they’re more likely to contain query terms.
- Can fix this to some extent by normalizing for document length.
- But is raw \( tf \) the right measure?

Weighting term frequency: \( tf \)

- What is the relative importance of
  - 0 vs. 1 occurrence of a term in a doc
  - 1 vs. 2 occurrences
  - 2 vs. 3 occurrences …
- Unclear: while it seems that more is better, a lot isn’t proportionally better than a few.
  - Can just use raw \( tf \)
  - Another option commonly used in practice:
    - \( t=\text{term}, d=\text{document} \)

\[
wf_{t,d} = 0 \text{ if } tf_{t,d} = 0, \quad 1 + \log tf_{t,d} \text{ otherwise}
\]
Score computation

Score for a query $q = \sum_{t \in q} tf_{t,d}$

- [Note: 0 if no query terms in document]
- This score can be zone-combined
- Can use $wf$ instead of $tf$ in the above
- Still doesn’t consider term scarcity in collection (ides is rarer than of)

Weighting should depend on the term overall

- Which of these tells you more about a doc?
  - 10 occurrences of hernia?
  - 10 occurrences of the?
- Would like to attenuate the weight of a common term
  - But what is “common”?
- Suggest looking at collection frequency ($cf$)
  - The total number of occurrences of the term in the entire collection of $n$ documents
Document frequency

- But document frequency \((df)\) may be better:
- \(df = \) number of docs in the corpus containing the term

<table>
<thead>
<tr>
<th>Word</th>
<th>(cf)</th>
<th>(df)</th>
</tr>
</thead>
<tbody>
<tr>
<td>alfa</td>
<td>10422</td>
<td>17</td>
</tr>
<tr>
<td>insurance</td>
<td>10440</td>
<td>3997</td>
</tr>
</tbody>
</table>

- Document/collection frequency weighting is only possible in known (static) collection
  - The number of documents in the entire collection of \(n\) documents
- So how do we make use of \(df\) ?

\[ idf_i = \log \left( \frac{n}{df_i} \right) \]

tf \(\times\) idf term weights

- tf \(\times\) idf measure combines:
  - term frequency \((tf)\)
    - or \(wf\), some measure of term density in a doc
  - inverse document frequency \((idf)\)
    - measure of informativeness of a term: its rarity across the whole corpus
    - could just be raw count of number of documents the term occurs in \((idf_i = 1/df)\)
    - but by far the most commonly used version is:

- See Kishore Papineni, NAACL 2, 2002 for theoretical justification
Summary: tf x idf (or \textbf{tf.idf})

- Assign a tf.idf weight to each term \(i\) in each document \(d\)

\[ w_{i,d} = tf_{i,d} \times \log\left(\frac{n}{df_i}\right) \]

- \(tf_{i,d}\) = frequency of term \(i\) in document \(d\)
- \(n\) = total number of documents
- \(df_i\) = the number of documents that contain term \(i\)

- Increases with the number of occurrences within a doc
- Increases with the rarity of the term across the whole corpus

Document frequency is the number of documents in the collection containing a certain term

\[ df_i, \quad \text{(9.7)} \]

If \(s\) is the number of documents in the collection, then surprise of a term \(i\) is defined as

\[ s_t = \frac{s}{df_i}, \quad \text{(9.8)} \]

The bigger the probability \(p(\text{term}_i)\)

\[ p(\text{term}_i) = \frac{df_i}{s}. \quad \text{(9.9)} \]

Information of a term is defined as

\[ I_i = \log_2(s_t) = -\log_2(p(\text{term}_i)). \quad \text{(9.10)} \]
\[ idf_i := I_t = \log_2(s_t) = -\log_2(p(\text{term}_i)) = \log_2 \left( \frac{s}{\partial f_t} \right) \approx \log \left( \frac{s}{\partial f_t} \right) \quad (9.12) \]

or

\[ w_{td} = tf_{td} \times \log \left( \frac{s}{\partial f_t} \right) = tf_{td} \cdot I_t \quad (9.13) \]

\[ w'_{td} = w_{td} \times \log \left( \frac{s}{\partial f_t} \right) = w_{td} \cdot I_t \quad (9.14) \]

With \( n \) being the number of different terms \( \text{term}_i \) and \( p(\text{term}_i) \) the probability of occurrence of the term, then the theoretical minimum average number of bits of a document is computed using the Shannon's formula of entropy

\[ H = -\sum_{i=1}^{n} p(\text{term}_i) \cdot \log_2 p(\text{term}_i). \quad (9.11) \]
Real-valued term-document matrices

- Function (scaling) of count of a word in a document:
  - Bag of words model
  - Each is a vector in $\mathbb{R}^v$
  - Here log-scaled $tf.idf$

<table>
<thead>
<tr>
<th>Term</th>
<th>Antony and Cleopatra</th>
<th>Julius Caesar</th>
<th>The Tempest</th>
<th>Hamlet</th>
<th>Othello</th>
<th>Macbeth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Antony</td>
<td>13.1</td>
<td>11.4</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Brutus</td>
<td>3.0</td>
<td>8.3</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Caesar</td>
<td>2.3</td>
<td>2.3</td>
<td>0.0</td>
<td>0.5</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>Calpurnia</td>
<td>0.0</td>
<td>11.2</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Cleopatra</td>
<td>17.7</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Meryc</td>
<td>0.5</td>
<td>0.0</td>
<td>0.7</td>
<td>0.9</td>
<td>0.9</td>
<td>0.3</td>
</tr>
<tr>
<td>Worser</td>
<td>1.2</td>
<td>0.0</td>
<td>0.6</td>
<td>0.6</td>
<td>0.6</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Documents as vectors

- Each doc $j$ can now be viewed as a vector of $wf \times idf$ values, one component for each term
- So we have a vector space
  - terms are axes
  - docs live in this space
  - even with stemming, may have 20,000+ dimensions
Why turn docs into vectors?

- First application: **Query-by-example**
  - Given a doc \( d \), find others “like” it.

- Now that \( d \) is a vector, find vectors (docs) “near” it....

**Intuition**

Postulate: Documents that are “close together” in the vector space talk about the same things.
Desiderata for proximity

- If $d_1$ is near $d_2$, then $d_2$ is near $d_1$
- If $d_1$ near $d_2$, and $d_2$ near $d_3$, then $d_1$ is not far from $d_3$
- No doc is closer to $d$ than $d$ itself

First cut

- Idea: Distance between $d_1$ and $d_2$ is the length of the vector $|d_1 - d_2|$.
  - Euclidean distance

- Why is this not a great idea?
- We still haven’t dealt with the issue of length normalization
  - Short documents would be more similar to each other by virtue of length, not topic
- However, we can implicitly normalize by looking at angles instead
Cosine similarity

- Distance between vectors $d_1$ and $d_2$ captured by the cosine of the angle $\theta$ between them.
- Note – this is similarity, not distance
  - No triangle inequality for similarity.

A vector can be normalized (given a length of 1) by dividing each of its components by its length – here we use the $L_2$ norm

$$\|x\|_2 = \sqrt{\sum_i x_i^2}$$

This maps vectors onto the unit sphere:

- Then, $|\tilde{d}_j| = \sqrt{\sum_{i=1}^n w_{i,j}} = 1$
- Longer documents don’t get more weight
Normalized vectors

- For normalized vectors, the cosine is simply the dot product:

\[ \cos(\vec{d}_j, \vec{d}_k) = \vec{d}_j \cdot \vec{d}_k \]

- Varies from 0 to 1!!!!!
Euclidean distance between vectors:

$$|d_j - d_k| = \sqrt{\sum_{i=1}^{n} (d_{i,j} - d_{i,k})^2}$$

For normalized vectors, Euclidean distance gives the same proximity ordering as the cosine measure.

Queries in the vector space model

Central idea: the query as a vector:

- We regard the query as short document
- We return the documents ranked by the closeness of their vectors to the query, also represented as a vector

$$\text{sim}(d_j, d_q) = \frac{\vec{d}_j \cdot \vec{d}_q}{\|\vec{d}_j\| \|\vec{d}_q\|} = \frac{\sum_{i=1}^{n} w_{i,j} w_{i,q}}{\sqrt{\sum_{i=1}^{n} w_{i,j}^2} \sqrt{\sum_{i=1}^{n} w_{i,q}^2}}$$

- Note that $d_q$ is very sparse!
- Varies from 0 to 1!!!!!
Normalized vectors

For normalized vectors, the cosine is simply the dot product:

\[
\cos(\vec{d}_j, \vec{d}_k) = \vec{d}_j \cdot \vec{d}_k
\]

Varies from 0 to 1!!!!!

Example

Docs: Austen’s *Sense and Sensibility*, *Pride and Prejudice*; Bronte’s *Wuthering Heights*. tf weights

<table>
<thead>
<tr>
<th></th>
<th>SaS</th>
<th>PaP</th>
<th>WH</th>
</tr>
</thead>
<tbody>
<tr>
<td>affection</td>
<td>115</td>
<td>58</td>
<td>20</td>
</tr>
<tr>
<td>jealous</td>
<td>10</td>
<td>7</td>
<td>11</td>
</tr>
<tr>
<td>gossip</td>
<td>2</td>
<td>0</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>SaS</th>
<th>PaP</th>
<th>WH</th>
</tr>
</thead>
<tbody>
<tr>
<td>affection</td>
<td>0.996</td>
<td>0.993</td>
<td>0.847</td>
</tr>
<tr>
<td>jealous</td>
<td>0.087</td>
<td>0.120</td>
<td>0.466</td>
</tr>
<tr>
<td>gossip</td>
<td>0.017</td>
<td>0.000</td>
<td>0.254</td>
</tr>
</tbody>
</table>

\[
\cos(\text{SAS, PAP}) = 0.996 \times 0.993 + 0.087 \times 0.120 + 0.017 \times 0.0 = 0.999
\]

\[
\cos(\text{SAS, WH}) = 0.996 \times 0.847 + 0.087 \times 0.466 + 0.017 \times 0.254 = 0.889
\]
What if we could take our vectors and “pack” them into fewer dimensions (say 50,000→100) while preserving distances?

(Well, almost.)

- Speeds up cosine computations
A random projection from \( n \) to \( m \) with
\[
m \ll n
\]
is given by this simple algorithm:
- choose a random direction \( \mathbf{a}_1 \) in the vector space;
- for \( i = 2 \) to \( m \)
  - choose a random direction \( \mathbf{a}_i \) that is orthogonal to \( \mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_{i-1} \);
  - project each document vector of dimension \( n \) into the subspace spanned by \( \{\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_m\} \).

The subspace spanned by \( \{\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_m\} \) defines a projection matrix of the dimension \( m \times n \)
\[
A = \begin{pmatrix}
    \mathbf{a}_1^T \\
    \mathbf{a}_2^T \\
    \vdots \\
    \mathbf{a}_m^T
\end{pmatrix} = \begin{pmatrix}
    a_{11} & a_{12} & \cdots & a_{1n} \\
    a_{21} & a_{22} & \cdots & a_{2n} \\
    \vdots & \vdots & \ddots & \vdots \\
    a_{m1} & a_{m2} & \cdots & a_{mn}
\end{pmatrix}.
\]
(9.15)

The vectors that represent the document collection and the query vector are projected with the projection matrix \( A \) into \( m \) dimensional space. Relative distances are preserved by projection \( A \) with high probability according to the The Johnson-Lindenstrauss lemma, see section 7.4. An alternative

The Johnson-Lindenstrauss lemma [W. Johnson (1984)] states that if \( s \) points in vector space of dimension \( n \) are projected onto a randomly selected subspace of suitably high dimensions \( m \), then the Euclidean distance between the points are approximately preserved. For
\[
0 < \epsilon < 1
\]
and a set of \( s \) vectors of the dimension \( n \) and dimension \( m \) with
\[
m > 8 \cdot \frac{\log(s)}{\epsilon^2}
\]
and a linear mapping
\[
f : R^n \rightarrow R^m
\]
exists such that that for the Euclidean metric
\[
(1 - \epsilon) \cdot \|\mathbf{x} - \mathbf{y}\|^2 \leq \|f(\mathbf{x}) - f(\mathbf{y})\|^2 \leq (1 + \epsilon) \cdot \|\mathbf{x} - \mathbf{y}\|^2.
\]
\[
\vdots
\]
(7.29)
Measures for Results

- All of the preceding criteria are measurable: we can quantify speed/size; we can make expressiveness precise
- The key measure: user happiness
  - What is this?
  - Speed of response/size of index are factors
  - But blindingly fast, useless answers won’t make a user happy
- Need a way of quantifying user happiness

Measuring user happiness

- Issue: who is the user we are trying to make happy?
  - Depends on the setting
- Web engine: user finds what they want and return to the engine
  - Can measure rate of return users
- eCommerce site: user finds what they want and make a purchase
  - Is it the end-user, or the eCommerce site, whose happiness we measure?
  - Measure time to purchase, or fraction of searchers who become buyers?
Measuring user happiness

- **Enterprise (company/govt/academic):** Care about “user productivity”
  - How much time do my users save when looking for information?
  - Many other criteria having to do with breadth of access, secure access, etc.

Happiness: elusive to measure

- Commonest proxy: *relevance* of search results
- But how do you measure relevance?
- We will detail a methodology here, then examine its issues
- Relevant measurement requires 3 elements:
  1. A benchmark document collection
  2. A benchmark suite of queries
  3. A binary assessment of either Relevant or Irrelevant for each query-doc pair
     - Some work on more-than-binary, but not the standard
Evaluating an IR system

- Note: the information need is translated into a query.
- Relevance is assessed relative to the information need, not the query.
- E.g., Information need: I'm looking for information on whether drinking red wine is more effective at reducing your risk of heart attacks than white wine.
- Query: wine red white heart attack effective
- You evaluate whether the doc addresses the information need, not whether it has those words.

Unranked retrieval evaluation: Precision and Recall

- **Precision**: fraction of retrieved docs that are relevant = \( P(\text{relevant}\mid\text{retrieved}) \)
- **Recall**: fraction of relevant docs that are retrieved = \( P(\text{retrieved}\mid\text{relevant}) \)

<table>
<thead>
<tr>
<th></th>
<th>Relevant</th>
<th>Not Relevant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retrieved</td>
<td>( tp )</td>
<td>( fp )</td>
</tr>
<tr>
<td>Not Retrieved</td>
<td>( fn )</td>
<td>( tn )</td>
</tr>
</tbody>
</table>

- Precision \( P = \frac{tp}{tp + fp} \)
- Recall \( R = \frac{tp}{tp + fn} \)
Accuracy

- Given a query an engine classifies each doc as “Relevant” or “Irrelevant”.
- Accuracy of an engine: the fraction of these classifications that is correct.
- Why is this not a very useful evaluation measure in IR?
- No result, 100% accuracy .....

Precision/Recall

- You can get high recall (but low precision) by retrieving all docs for all queries!
- Recall is a non-decreasing function of the number of docs retrieved
- In a good system, precision decreases as either number of docs retrieved or recall increases
  - A fact with strong empirical confirmation
Difficulties in using precision/recall

- Should average over large corpus/query ensembles
- Need human relevance assessments
  - People aren’t reliable assessors
- Assessments have to be binary
  - Nuanced assessments?
- Heavily skewed by corpus/authorship
  - Results may not translate from one domain to another

A combined measure: $F$

- Combined measure that assesses this tradeoff is $F$ measure (weighted harmonic mean):

$$F = \frac{1}{\alpha \frac{1}{P} + (1 - \alpha) \frac{1}{R}} = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}$$

- People usually use balanced $F_1$ measure
  - i.e., with $\beta = 1$ or $\alpha = \frac{1}{2}$
Evaluating ranked results

- Evaluation of ranked results:
  - The system can return any number of results
  - By taking various numbers of the top returned documents (levels of recall), the evaluator can produce a precision-recall curve