AD-hoc IR: Basic Process

1. Information Need
2. Representation
3. Query
4. Retrieval Model
5. Indexed Objects
6. Retrieved Objects
7. Evaluation

Course Review
Prof. Chris Clifton
6 December 2017
Ad-hoc IR: Terminologies

Terminologies:
• Query
  – Representative data of user’s information need: text (default) and other media
• Document
  – Data candidate to satisfy user’s information need: text (default) and other media
• Database|Collection|Corpus
  – A set of documents
• Corpora
  – A set of databases
  – Valuable corpora from TREC (Text Retrieval Evaluation Conference)

Text Representation

• What to index?
  – All words
    • Stopwords, Stemming
  – Controlled Vocabulary
    • Ontologies
  – Phrases, N-Grams
• How to represent?
  – “Bag of Words” (Vector Space Model)
  – Preserve order, distance
Text Preprocessing: extract representative index terms
- Parse query/document for useful structure
  - E.g., title, anchor text, link, tag in xml.....
- Tokenization
  - For most western languages, words separated by spaces; deal with punctuation, capitalization, hyphenation
  - For Chinese, Japanese: more complex word segmentation...
- Remove stopwords: (remove “the”, “is”,..., existing standard list)
- Morphological analysis (e.g., stemming):
  - Stemming: determine stem form of given inflected forms
- Other: extract phrases; decompounding for some European languages

rörelseuppskattningssökningsintervallsinställningar

Text Representation: Indexing

<table>
<thead>
<tr>
<th>Word</th>
<th>Frequency</th>
<th>$r \cdot p_r$</th>
<th>Word</th>
<th>Frequency</th>
<th>$r \cdot p_r$</th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>1130021</td>
<td>0.059</td>
<td>market</td>
<td>52110</td>
<td>0.101</td>
</tr>
<tr>
<td>of</td>
<td>547311</td>
<td>0.058</td>
<td>bank</td>
<td>47940</td>
<td>0.109</td>
</tr>
<tr>
<td>to</td>
<td>516636</td>
<td>0.082</td>
<td>stock</td>
<td>47401</td>
<td>0.110</td>
</tr>
<tr>
<td>a</td>
<td>464736</td>
<td>0.098</td>
<td>trade</td>
<td>47310</td>
<td>0.112</td>
</tr>
<tr>
<td>in</td>
<td>390819</td>
<td>0.103</td>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>and</td>
<td>387703</td>
<td>0.122</td>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Statistics collected from Wall Street Journal (WSJ), 1987
Text Representation: Vector Space Model

- Any text object can be represented by a term vector
  - Documents, queries, passages, sentences
  - A query can be seen as a short document
- Similarity is determined by distance in the vector space
  - Example: cosine of the angle between two vectors

<table>
<thead>
<tr>
<th></th>
<th>D1</th>
<th>D2</th>
<th>D3</th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Oracle</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.2</td>
</tr>
<tr>
<td>Starbucks</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

---

Text Representation: Vector Space Model

- Vector representation
Text Representation: Process of Indexing

- Document → Parser
  - Extract useful fields, useful tokens (lex/yacc)

- Text Preprocess
  - Remove Stopword, Stemming, Phrase Extraction etc

- Indexer
  - Term Dictionary
  - Inverted Lists

- Full Text Indexing
  - Document Attributes

Text Representation: Inverted Lists

<table>
<thead>
<tr>
<th>Doc ID</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>kids question noting in 1980s</td>
</tr>
<tr>
<td>2</td>
<td>young man question everything in 1970s</td>
</tr>
<tr>
<td>3</td>
<td>kids question questions in 1980s</td>
</tr>
<tr>
<td>4</td>
<td>young man question nothing in 2000s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Term ID</th>
<th>Term</th>
<th>Documents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>kids</td>
<td>1, 3</td>
</tr>
<tr>
<td>2</td>
<td>question</td>
<td>1, 2, 3, 4</td>
</tr>
<tr>
<td>3</td>
<td>nothing</td>
<td>1, 4</td>
</tr>
<tr>
<td>4</td>
<td>in</td>
<td>1, 2, 3, 4</td>
</tr>
<tr>
<td>5</td>
<td>19060s</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>young</td>
<td>2, 4</td>
</tr>
<tr>
<td>7</td>
<td>man</td>
<td>2, 4</td>
</tr>
<tr>
<td>8</td>
<td>everything</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>1970s</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>questions</td>
<td>3</td>
</tr>
<tr>
<td>11</td>
<td>1980s</td>
<td>3</td>
</tr>
<tr>
<td>12</td>
<td>2000s</td>
<td>4</td>
</tr>
</tbody>
</table>
Types of Retrieval Models

• Exact Match (Selection) vs. Best Match (Ranking)
  • Best Match is usually more accurate/effective
    – Do not need precise query; representative query generates good results
    – Users have control to explore the rank list: view more if need every piece; view less if need one or two most relevant
  • Exact Match
    – Hard to define the precise query; too strict (terms are too specific) or too coarse (terms are too general)
    – Users have no control over the returned results
    – Still prevalent in some markets (e.g., legal retrieval)

Overview of Retrieval Models

Retrieval Models
• Boolean
• Vector space
  – Basic vector space
  – Extended Boolean
• Probabilistic models
  – Statistical language models
  – Two Possion model
  – Bayesian inference networks
• Citation/Link analysis models
  – Page rank
  – Hub & authorities

<table>
<thead>
<tr>
<th>Retrieval Models</th>
<th>Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>Westlaw</td>
</tr>
<tr>
<td>Vector space</td>
<td>SMART, LUCENE</td>
</tr>
<tr>
<td>Probabilistic models</td>
<td>Lemur Project (Indri, Galago)</td>
</tr>
<tr>
<td>Citation/Link analysis models</td>
<td>Google, Clever</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Retrieval Models: Unranked Boolean

Unranked Boolean: Exact match method
- Selection Model
  - Retrieve a document iff it matches the precise query
  - Often return unranked documents (or with chronological order)
- Operators
  - Logical Operators: AND OR, NOT
  - Proximity operators:
    - #1(white house) (i.e., within one word distance, phrase)
    - #sen(Iraq weapon) (i.e., within a sentence)
  - String matching operators: Wildcard (e.g., ind* for india and indonesia)
  - Field operators: title(information and retrieval)...

Retrieval Models: Ranked Boolean

Ranked Boolean: Exact match
- Similar to unranked Boolean but documents are ordered by some criterion

Retrieve docs from Wall Street Journal Collection
Query: (Thailand AND stock AND market)
Which word is more important?
Many “stock” and “market”, but fewer “Thailand”. Fewer may be more indicative

Term Frequency (TF): Number of occurrence in query/doc; larger number means more important

Inversed Document Frequency (IDF):
Larger means more important

Total number of docs
Number of docs contain a term

There are many variants of TF, IDF: e.g., consider document length
Retrieval Models: Ranked Boolean

- Ranked Boolean: Calculate doc score
- Term evidence: Evidence from term $i$ occurred in doc $j$: $(tf(i,j))$ and $(tf(i,j) * idf(i))$
- AND weight: minimum of argument weights
- OR weight: maximum of argument weights

<table>
<thead>
<tr>
<th>Term evidence</th>
<th>Min=0.2 AND</th>
<th>Max=0.6 OR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0. 0. 0.</td>
<td>2 6 4</td>
<td>0. 0. 0.</td>
</tr>
</tbody>
</table>

Query: (Thailand AND stock AND market)

Retrieval Models: Vector Space Model

- Give two vectors of query and document
- query $\vec{q} = (q_1, q_2, ..., q_n)$
- document $\vec{d}_j = (d_{j1}, d_{j2}, ..., d_{jn})$
- calculate the similarity

Cosine similarity: Angle between vectors

$$\text{sim}(\vec{q}, \vec{d}_j) = \cos(\theta(\vec{q}, \vec{d}_j))$$

$$\cos(\theta(\vec{q}, \vec{d}_j)) = \frac{\vec{q} \cdot \vec{d}_j}{||\vec{q}|| ||\vec{d}_j||} = \frac{q_1d_{j1} + q_2d_{j2} + ... + q_nd_{jn}}{\sqrt{q_1^2 + q_2^2 + ... + q_n^2} \sqrt{d_{j1}^2 + d_{j2}^2 + ... + d_{jn}^2}}$$
Retrieval Models: Vector Space Model

• Vector Coefficients
• The coefficients (vector elements) represent term evidence/term importance
• Derived from several elements
  – Document term weight: Evidence of the term in the document/query
  – Collection term weight: Importance of term from observation of collection
  – Length normalization: Reduce document length bias
• Naming convention for coefficients:

\[ q_k \text{, } d_{j,k} = DCL.DCL \]

First triple represents query term; second for document term

Retrieval Models: Vector Space Model

• Common vector weight components:
• Inc.ltc: widely used term weight
  – “l”: \( \log(tf) + 1 \)
    • 0 if \( tf = 0 \)
  – “n”: no weight/normalization
  – “t”: \( \log(N/df) \)
  – “c”: cosine normalization

\[
\frac{q_k \cdot d_{j,k}}{\sqrt{\sum (\log tf_j(k) + 1) \sum (\log df_j(k) + 1)}} \cdot \sqrt{\sum (\log tf_j(k) + 1) \sum (\log df_j(k) + 1)}
\]
Retrieval Models: Latent Semantic Indexing

- Latent Semantic Indexing (LSI): Explore correlation between terms and documents
  - Two terms are correlated (may share similar semantic concepts) if they often co-occur
  - Two documents are correlated (share similar topics) if they have many common words
- Associate each term and document with a small number of semantic concepts/topics

Use singular value decomposition (SVD) to find a small set of concepts/topics.
Retrieval Models: Latent Semantic Indexing

- Use singular value decomposition (SVD) to find a small set of concepts/topics

\[ m: \text{number of concepts/topics} \]

\[ \text{Representation of document in concept space} \]

\[ \text{Representation of term in concept space} \]

\[ \text{Diagonal matrix: concept space} \]

Properties of Latent Semantic Indexing

- Diagonal elements of \( S \) as \( S_k \) in descending order, the larger the more important

\[ \hat{x}_k = \sum_{i \leq k} u_k s_{ik} v_k' \] is the rank-\( k \) matrix that best approximates \( X \), where \( u_k \) and \( v_k' \) are the column vector of \( U \) and \( V' \)
Other properties of Latent Semantic Indexing

- The columns of $\mathbf{U}$ are eigenvectors of $\mathbf{X}\mathbf{X}^T$
- The columns of $\mathbf{V}$ are eigenvectors of $\mathbf{X}^T\mathbf{X}$
- The singular values on the diagonal of $\mathbf{S}$, are the positive square roots of the nonzero eigenvalues of both $\mathbf{A}\mathbf{A}^T$ and $\mathbf{A}^T\mathbf{A}$

Retrieval Models: Latent Semantic Indexing

- Retrieval with respect to a query
- Map (fold-in) a query into the representation of the concept space

$$q^* = q^T U_k Inv(S_k)$$

- Use the new representation of the query to calculate the similarity between query and all documents
  - Cosine Similarity
Probability Ranking Principle (PRP)

Let \( x \) represent a document in the collection. Let \( R \) represent relevance of a document w.r.t. given (fixed) query and let \( R=1 \) represent relevant and \( R=0 \) not relevant.

Need to find \( p(R=1|x) \) - probability that a document \( x \) is relevant.

\[
p(R=1|x) = \frac{p(x|R=1)p(R=1)}{p(x)} \quad \text{p(R=1), p(R=0) - prior probability of retrieving a relevant or non-relevant document}
\]

\[
p(R=0|x) = \frac{p(x|R=0)p(R=0)}{p(x)} \quad \text{p(x|R=1), p(x|R=0) - probability that if a relevant (not relevant) document is retrieved, it is } x.
\]

\[
p(R=0|x) + p(R=1|x) = 1
\]

• Simple case: no selection costs or other utility concerns that would differentially weight errors
• PRP in action: Rank all documents by \( p(R=1|x) \)
• Theorem: Using the PRP is optimal, in that it minimizes the loss (Bayes risk) under 1/0 loss
  – Provable if all probabilities correct, etc. [e.g., Ripley 1996]
• How do we compute all those probabilities?
  – Do not know exact probabilities, have to use estimates
Binary Independence Model

- Queries: binary term incidence vectors
- Given query \( q \),
  - for each document \( d \) need to compute \( p(R|q,d) \).
  - replace with computing \( p(R|q,x) \) where \( x \) is the binary term incidence vector representing \( d \).
- Interested only in ranking
- Use odds and Bayes’ Rule:

\[
O(R|q,x) = \frac{p(R=1|q)p(\bar{x} R=1,q)}{p(R=0|q)p(\bar{x} R=0,q)}
\]

- Using Independence Assumption:

\[
p(\bar{x} R=1,q) = \prod_{i=1}^{n} \frac{p(x_i R=1,q)}{p(x_i R=0,q)}
\]

\[
O(R|q,\bar{x}) = O(R|q) \cdot \prod_{i=1}^{n} \frac{p(x_i R=1,q)}{p(x_i R=0,q)}
\]
Binary Independence Model

\[ O(R \mid q, \bar{x}) = O(R \mid q) \cdot \prod_{i=1}^{n} \frac{p(x_i \mid R = 1, q)}{p(x_i \mid R = 0, q)} \]

• Since \( x_i \) is either 0 or 1:

\[ O(R \mid q, \bar{x}) = O(R \mid q) \cdot \prod_{x_i=1}^{n} \frac{p(x_i = 1 \mid R = 1, q)}{p(x_i = 1 \mid R = 0, q)} \times \prod_{x_i=0}^{n} \frac{p(x_i = 0 \mid R = 1, q)}{p(x_i = 0 \mid R = 0, q)} \]

• Let \( p_i = p(x_i = 1 \mid R = 1, q) \); \( r_i = p(x_i = 1 \mid R = 0, q) \); 

• Assume, for all terms not occurring in the query \( (q_i = 0) \) \( p_i = r_i \)

\[ O(R \mid q, \bar{x}) = O(R \mid q) \cdot \prod_{x_i=1}^{n} \frac{p_i}{r_i} \cdot \prod_{x_i=0}^{n} \frac{1 - p_i}{1 - r_i} \]

Binary Independence Model

\[ O(R \mid q, \bar{x}) = O(R \mid q) \cdot \prod_{x_i=1}^{n} \frac{p_i(1 - r_i)}{r_i(1 - p_i)} \cdot \prod_{x_i=0}^{n} \frac{1 - p_i}{1 - r_i} \]

Constant for each query

Only quantity to be estimated for rankings

\[ \text{Retrieval Status Value (RSV)} = \log \prod_{x_i=1}^{n} \frac{p_i(1 - r_i)}{r_i(1 - p_i)} = \sum_{x_i=1}^{n} \log \frac{p_i(1 - r_i)}{r_i(1 - p_i)} \]
Binary Independence Model

All boils down to computing RSV.

\[ RSV = \log \prod_{x_i = 0} p_i (1 - r_i) - \log \sum_{x_i = 0} p_i (1 - r_i) \]

\[ RSV = \sum_{x_i = 0} c_i; \quad c_i = \log \frac{p_i (1 - r_i)}{r_i (1 - p_i)} \]

The \( c_i \) are log odds ratios.
They function as the term weights in this model.

So, how do we compute \( c_i \)'s from our data?

---

Binary Independence Model

- Estimating RSV coefficients in theory
- For each term \( i \) look at this table of document counts:

<table>
<thead>
<tr>
<th>Documents</th>
<th>Relevant</th>
<th>Non-Relevant</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_i = 1 )</td>
<td>( s )</td>
<td>( n - s )</td>
<td>( n )</td>
</tr>
<tr>
<td>( x_i = 0 )</td>
<td>( S - s )</td>
<td>( N - n - S + s )</td>
<td>( N - n )</td>
</tr>
<tr>
<td>Total</td>
<td>( S )</td>
<td>( N - S )</td>
<td>( N )</td>
</tr>
</tbody>
</table>

- Estimates:
  \[ p_i \approx \frac{s}{S} \]
  \[ r_i \approx \frac{(n - s)}{(N - S)} \]

\[ c_i \approx K(N, n, S, s) = \log \frac{s/(S - s)}{(n - s)/(N - n - S + s)} \]

For now, assume no zero terms.
Estimation – key challenge

- \( p_i \) (probability of occurrence in relevant documents) cannot be approximated as easily
- \( p_i \) can be estimated in various ways:
  - from relevant documents if know some
    - Relevance weighting can be used in a feedback loop
  - constant (Croft and Harper combination match) – then just get idf weighting of terms (with \( p_i = 0.5 \))
    \[
    RSV = \sum_{x_i \in q} \log \frac{N}{n_i}
    \]
  - proportional to prob. of occurrence in collection
    - Greiff (SIGIR 1998) argues for \( 1/3 + 2/3 \) df/N

Okapi BM25

- BM25 metric, used in Okapi IR system
  - \( V=\)relevant documents, \( VNR=\)not relevant

- \( RSV_d = \sum_{t \in q} \left[ \log \left( \frac{|VR_{d}| + 0.5}{|VNR_{d}| + 0.5} \right) \times \frac{(k_1 + 1)tf_{td}}{k_1(1-b)\left(\frac{td}{tave}\right) + tf_{td}} \times \frac{(k_2 + 1)tf_{te}}{k_2 + tf_{te}} \right] \)
  - Probabilistic interpretation of IDF
  - Term frequency with normalization by length
  - Query term weighting
PRP and BIM

• Getting reasonable approximations of probabilities is possible.
• Requires restrictive assumptions:
  – Term independence
  – Terms not in query don’t affect the outcome
  – Boolean representation of documents/queries/relevance
  – Document relevance values are independent
• Some of these assumptions can be removed
• Problem: either require partial relevance information or only can derive somewhat inferior term weights

Query Expansion

• Users often start with short queries with ambiguous representations
• Observation: Many people refine their queries by analyzing the results from initial queries, or consulting other resources (thesaurus)
  – By adding and removing terms
  – By reweighting terms
  – By adding other features (e.g., Boolean operators)
• Technique of query expansion:
  Can a better query be created automatically?
Query Expansion

- Add terms to query to improve recall
  - And possibly precision
- Query Expansion via External Resources
  - Thesaurus
    - “Industrial Chemical Thesaurus”, “Medical Subject Headings” (MeSH)
  - Semantic network
    - WordNet
- Relevance Feedback
  - Use user-specified “good documents” to get new terms
  - Blind/Pseudo Relevance Feedback
Relevance Feedback Vector Space Model

- **Goal**: Move new query close to relevant documents and far away from irrelevant documents
- **Approach**: New query is a weighted average of original query, and relevant and non-relevant document vectors

\[
\tilde{q}' = \tilde{q} + \alpha \frac{1}{|R|} \sum_{d_i \in R} \tilde{d}_i - \beta \frac{1}{|NR|} \sum_{d_i \in NR} \tilde{d}_i \quad \text{(Rocchio formula)}
\]

- **Desirable weights for \( \alpha \) and \( \beta \)**

Try find \( \alpha \) and \( \beta \) such that

\[
\tilde{q}(\alpha, \beta) \cdot \tilde{d}_i \geq 1 \quad \text{for} \quad \tilde{d}_i \in R
\]

\[
\tilde{q}(\alpha, \beta) \cdot d_i \leq -1 \quad \text{for} \quad d_i \in NR
\]
Blind (Pseudo) Relevance Feedback

- Pseudo-relevance feedback
  - Assume top $N$ (e.g., 20) documents in initial list are relevant
  - Assume bottom $N'$ (e.g., 200-300) in initial list are irrelevant
  - Calculate weights of term according to some criterion (e.g., Rocchio)
  - Select top $M$ (e.g., 10) terms
- Local context analysis
  - Similar approach to pseudo-relevance feedback
  - But use passages instead of documents for initial retrieval; use different term weight selection algorithms

CS47300: Web Information Search and Management

Course Review
Prof. Chris Clifton
8 December 2017
Ad-hoc IR Evaluation: Criteria

- Effectiveness
  - Favor returned document ranked lists with more relevant documents at the top
  - Objective measures
    - Recall and Precision
    - Mean-average precision
    - Rank based precision

<table>
<thead>
<tr>
<th>Relevant</th>
<th>Retrieved</th>
<th>Not retrieved</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relevant</td>
<td>Relevant docs retrieved</td>
<td>Relevant docs not retrieved</td>
</tr>
<tr>
<td>Irrelevant</td>
<td>Irrelevant docs retrieved</td>
<td>Irrelevant docs not retrieved</td>
</tr>
</tbody>
</table>

\[
\text{Precision} = \frac{\text{Relevant docs retrieved}}{\text{Retrieved docs}}
\]

\[
\text{Recall} = \frac{\text{Relevant docs retrieved}}{\text{Relevant docs}}
\]

Evaluation: “Ground Truth”

- Manual labeling: Expensive
  - Especially to find “Relevant documents not retrieved”

<table>
<thead>
<tr>
<th>Relevant</th>
<th>Retrieved</th>
<th>Not retrieved</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relevant</td>
<td>Relevant docs retrieved</td>
<td>Relevant docs not retrieved</td>
</tr>
<tr>
<td>Irrelevant</td>
<td>Irrelevant docs retrieved</td>
<td>Irrelevant docs not retrieved</td>
</tr>
</tbody>
</table>

- Pooling Strategy
  - Retrieve documents using multiple methods
  - Judge top \( n \) documents from each method
  - Whole retrieved set is the union of top retrieved documents from all methods
  - Problems: the judged relevant documents may not be complete
  - \textit{It is possible to estimate the total number of relevant documents by random sampling}
Everyone wants a “score” – which system is best

- Mean average precision
  - Calculate precision at each relevant document; average over all precision values
- 11-point interpolated average precision
  - Calculate precision at standard recall points (e.g., 10%, 20%...); smooth the values; estimate 0 % by interpolation
  - Average the results
- Rank based precision
  - Calculate precision at top ranked documents (e.g., 5, 10, 15...)
  - Desirable when users care more for top ranked documents

Evaluation Example

- Evaluate a ranked list
  - Precision at Recall
- Evaluate at every relevant document

<table>
<thead>
<tr>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.1</td>
</tr>
<tr>
<td>1</td>
<td>0.2</td>
</tr>
<tr>
<td>0.667</td>
<td>0.2</td>
</tr>
<tr>
<td>0.714</td>
<td>0.3</td>
</tr>
<tr>
<td>0.8</td>
<td>0.4</td>
</tr>
<tr>
<td>0.667</td>
<td>0.4</td>
</tr>
<tr>
<td>0.714</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Not Retrieved: ++++++
Text Categorization

- **Task**
  - Assign predefined categories to text documents / objects
- **Motivation**
  - Provide an organizational view of the data
- **Procedures**
  - **Training**: Given a set of categories and labeled document examples; learn a method to map a document to correct category (categories)
  - **Testing**: Predict the category (categories) of a new document

Example: 1990 US Census

- Included 22 million responses
- Needed to be classified into industry categories (200+) and occupation categories (500+)
- Estimated $15 million if done by hand
- Two alternative automatic text categorization methods evaluated
  - **Knowledge-Engineering (Expert System)**
    - Human developed rules, 192 person-months, 47% accurate
  - **Machine Learning (k-nearest neighbor method)**
    - K-Nearest Neighbor, 4 person-months, 60% accurate
Techniques Explored in Text Categorization

- Rule-based Expert system (Hayes, 1990)
- Nearest Neighbor methods (Creecy'92; Yang'94)
- Decision symbolic rule induction (Apte'94)
- Naïve Bayes (Language Model) (Lewis'94; McCallum’98)
- Regression method (Furh’92; Yang'92)
- Support Vector Machines (Joachims’98)
- Boosting or Bagging (Schapier’98)
- Neural networks (Wiener’95)
- ……

Text Categorization: Evaluation

Contingency Table Per Category (for all docs)

<table>
<thead>
<tr>
<th></th>
<th>Truth: True</th>
<th>Truth: False</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predicted Positive</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>Predicted Negative</td>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>a+c</td>
<td>b+d</td>
<td>n=a+b+c+d</td>
</tr>
</tbody>
</table>

a: number of truly positive docs  b: number of false-positive docs

c: number of false negative docs  d: number of truly-negative docs

n: total number of test documents
Text Categorization: Evaluation

Recall: \( r = \frac{a}{a+c} \) percentage of positive docs detected

Precision: \( p = \frac{a}{a+b} \) how accurate are the predicted positive docs

Accuracy: \( \frac{a+d}{n} \) how accurate are all the predicted docs

F-measure: \( F_{\beta} = \frac{(\beta^2 + 1)pr}{\beta^2 p + r} \) \( F_1 = \frac{2pr}{p + r} \)

Harmonic average: \( \frac{1}{\frac{1}{\frac{1}{x_1} + \frac{1}{x_2}}} \)

Error: \( \frac{b+c}{n} \) error rate of predicted docs

Accuracy + Error = 1

• Micro F1-Measure
  – Calculate a single contingency table for all categories and calculate F1 measure
  – Treat each prediction with equal weight; better for algorithms that work well on large categories

• Macro F1-Measure
  – Calculate a single contingency table for every category; calculate F1 measure separately and average the values
  – Treat each category with equal weight; better for algorithms that work well on many small categories
Text Categorization: Evaluation

Performance of different algorithms on Reuters-21578 corpus: 90 categories, 7769 Training docs, 3019 test docs, (Yang, JIR 1999)

K-Nearest Neighbor Classifier

- Idea: find your language by what language your neighbors speak
- Use K nearest neighbors to vote
  1-NN: Red; 5-NN: Brown; 10-NN: ?; Weighted 10-NN: Brown
K Nearest Neighbor: Framework

Training data \( D = \{ (x_i, y_i) \} \), \( x_i \in \mathbb{R}^M \), docs, \( y_i \in \{0, 1\} \)

Test data \( x \in \mathbb{R}^M \) The neighborhood is \( D_k \in D \)

Scoring Function \( \hat{y}(x) = \frac{1}{k} \sum_{x_i \in D_k(x)} \text{sim}(x, x_i) y_i \)

Classification: \[
\begin{cases}
1 & \text{if } \hat{y}(x) - t > 0 \\
0 & \text{otherwise}
\end{cases}
\]

Document Representation: \( X \) uses tf.idf weighting for each dimension

Choices of Similarity Functions

- **Euclidean distance**
  \[
d(x_1, x_2) = \sqrt{\sum_v (x_{1v} - x_{2v})^2}
\]

- **Kullback Leibler distance**
  \[
d(x_1, x_2) = \sum_v x_{1v} \log \frac{x_{1v}}{x_{2v}}
\]

- **Dot product**
  \[
  \vec{x_1} \cdot \vec{x_2} = \sum_v x_{1v} x_{2v}
  \]

- **Cosine Similarity**
  \[
  \cos(x_1, x_2) = \frac{\sum_v x_{1v} x_{2v}}{\sqrt{\sum_v x_{1v}^2} \sqrt{\sum_v x_{2v}^2}}
  \]

- **Kernel functions**
  \[
  k(x_1, x_2) = e^{-d(x_1, x_2)/2\sigma^2} \quad \text{(Gaussian Kernel)}
  \]

Automatic learning of the metrics
Choices of Number of Neighbors (K)

Trade off between small number of neighbors and large number of neighbors

Characteristics of KNN

Pros
- Simple and intuitive, based on local-continuity assumption
- Widely used and provide strong baseline in TC Evaluation
- No training needed, low training cost
- Easy to implement; can use standard IR techniques (e.g., tf.idf)

Cons
- Heuristic approach, no explicit objective function
- Difficult to determine the number of neighbors
- High online cost in testing; find nearest neighbors has high time complexity
Naïve Bayes Classification

• Naïve Bayes (NB) Classification
  – Generative Model: Model both the input data (i.e., document contents) and output data (i.e., class labels)
  – Make strong assumption of the probabilistic modeling approach

• Methodology
  – Similar with the idea of language modeling approaches for information retrieval
  – Train a language model for all the documents in one category

  \[ \text{Category 1: } (\tilde{d}_{1,1}, \tilde{d}_{1,2}, \ldots, \tilde{d}_{1,n_1}) \rightarrow \text{Language model } \theta_1 \]

  \[ \text{Category 2: } (\tilde{d}_{2,1}, \tilde{d}_{2,2}, \ldots, \tilde{d}_{2,n_2}) \rightarrow \text{Language model } \theta_2 \]

  \[ \text{......} \]

  \[ \text{Category C: } (\tilde{d}_{C,1}, \tilde{d}_{C,2}, \ldots, \tilde{d}_{C,n_C}) \rightarrow \text{Language model } \theta_C \]

  – What is the language model? (Multinomial distribution)
  – How to estimate the language model for all the documents in one category?
Naïve Bayes Classification

Maximum Likelihood Estimation:
- Find model parameters for a category that maximizes generation likelihood:

\[ \theta_c^* = \arg \max_{\theta_c} p(\tilde{d}_{c1},...,\tilde{d}_{cn} \mid \theta_c) \]

There are K words in vocabulary, \( w_1...w_K \)

Data: documents \( \tilde{d}_{c1},...,\tilde{d}_{cn} \)

For \( \tilde{d}_{ci} \) with counts \( c_i(w_1), \ldots, c_i(w_k) \), and length \( |\tilde{d}_{ci}| \)

Model: multinomial M with parameters \( \{p(w_k)\} \)

Likelihood:

\[ \Pr(\tilde{d}_{c1},...,\tilde{d}_{cn} \mid \theta) \]

\[ \theta_c^* = \arg \max_{\theta_c} p(\tilde{d}_{c1},...,\tilde{d}_{cn} \mid \theta_c) \]

---

Smoothed Estimator:

- Laplace Smoothing

\[ p(w \mid \theta_c^*) = \frac{1 + \sum_{i=1}^{n_c} c_i(w)}{K + \sum_{i=1}^{n_c} |\tilde{d}_{ci}|} \]

- Hierarchical Smoothing

\[ p(w \mid \theta_c^*) = \lambda_1 P(w \mid \theta_c^*) + \lambda_2 P(w \mid \theta_{c_{mp1}}^*) + \ldots + \lambda_m P(w \mid \theta_{c_{moo}}^*) \]

- Dirichlet Smoothing
Naïve Bayes Classification

- **Prediction:**
  \[
  c^* = \arg \max_c p(c \mid d_i) \\
  = \arg \max_c \left\{ \frac{p(c)p(d_i \mid c)}{p(d_i)} \right\} \\
  = \arg \max_c \left\{ p(c)p(d_i \mid c) \right\} \quad \text{(Bayes Rule)} \\
  = \arg \max_c \left\{ p(c)\prod_k p(w_k \mid c)^{c_i(w_k)} \right\} \quad \text{(Multinomial Dist)} \\
  = \arg \max_c \left\{ \log(p(c)) + \sum_k c_i(w_k) \log(p(w_k \mid c) \right\} \\
  \]

  Plug in the estimator

- **Summary**
  - Utilize multinomial distribution for modeling categories and documents
  - Use posterior distribution (posterior of category given document) to predict optimal category

- **Pros**
  - Solid probabilistic foundation
  - Fast online response, linear classifier for binary classification

- **Cons**
  - Empirical performance not very strong
  - Probabilistic model for each category is estimated to maximize the data likelihood for documents in the category (generative), not for purpose of distinguishing documents in different categories (discriminative)
Support Vector Machine: Large-Margin Decision Criterion

The margin is:

$$m = \frac{2}{||w||}$$

Soft-Margin SVM

- We tolerate some error for specific data points as

$$\frac{1}{2}||w||^2 + C \sum \xi_i$$
Non-linear SVM

Key idea: transform \( X_i \) to a higher dimension space

The Kernel Trick

- Recall the SVM optimization problem

\[
\text{max. } W(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j
\]

subject to \( C \geq \alpha_i \geq 0, \sum_{i=1}^{n} \alpha_i y_i = 0 \)

The data points only appear as inner product

As long as we can calculate the inner product in the feature space, we do not need the mapping explicitly

Many common geometric operations (angles, distances) can be expressed by inner products

Define the kernel function \( K \) by

\[
K(x_i, x_j) = \phi(x_i)^T \phi(x_j)
\]
Clustering: Issues

- Representation for clustering
  - Document representation
    - Vector space? Normalization?
  - Need a notion of similarity/distance
- How many clusters?
  - Fixed a priori?
  - Completely data driven?
    - Avoid “trivial” clusters - too large or small
      - In an application, if a cluster's too large, then for navigation purposes you've wasted an extra user click without whittling down the set of documents much.

Clustering Algorithms

- Partitioning “flat” algorithms
  - Usually start with a random (partial) partitioning
  - Refine it iteratively
    - k means/medoids clustering
    - Model based clustering
- Hierarchical algorithms
  - Bottom-up, agglomerative
  - Top-down, divisive
Partitioning Algorithms

• Partitioning method: Construct a partition of \( n \) documents into a set of \( k \) clusters
• Given: a set of documents and the number \( k \)
• Find: a partition of \( k \) clusters that optimizes the chosen partitioning criterion
  – Globally optimal: exhaustively enumerate all partitions
  – Effective heuristic methods: k-means and k-medoids algorithms

K-Means

• Assumes documents are real-valued vectors.
• Clusters based on centroids (aka the center of gravity or mean) of points in a cluster, \( c \):
  \[
  \bar{\mu}(c) = \frac{1}{|c|} \sum_{x \in c} \bar{x}
  \]
• Reassignment of instances to clusters is based on distance to the current cluster centroids.
  – (Or one can equivalently phrase it in terms of similarities)
**K Means Example**

**(K=2)**

- Pick seeds
- Reassign clusters
- Compute centroids
- Reassign clusters
- Compute centroids
- Reassign clusters
- Converged!

**Hierarchical Clustering**

- Build a tree-based hierarchical taxonomy (*dendrogram*) from a set of unlabeled examples.

- One option to produce a hierarchical clustering is recursive application of a partitional clustering algorithm to produce a hierarchical clustering.
Hierarchical Clustering algorithms

- **Agglomerative** (bottom-up):
  - Start with each document being a single cluster
  - Join closest two clusters
  - Eventually all documents belong to the same cluster

- **Divisive** (top-down):
  - Start with all documents belong to the same cluster
  - Divide to get “best” two clusters, repeat
  - Eventually each node forms a cluster on its own.

- Does not require the number of clusters $k$ in advance
- Needs a termination/readout condition
  - The final mode in both Agglomerative and Divisive is of no use.

What Is A Good Clustering?

- Internal criterion: A good clustering will produce high quality clusters in which:
  - the intra-class (that is, intra-cluster) similarity is high
  - the inter-class similarity is low
  - The measured quality of a clustering depends on both the document representation and the similarity measure used

- External criterion: The quality of a clustering is also measured by its ability to discover some or all of the hidden patterns or latent classes
  - Assessable with gold standard data
External Evaluation of Cluster Quality

- Assesses clustering with respect to ground truth
- Assume that there are \( C \) gold standard classes, while our clustering algorithms produce \( k \) clusters, \( \pi_1, \pi_2, \ldots, \pi_k \) with \( n_i \) members.
- Simple measure: purity, the ratio between the dominant class in the cluster \( \pi_i \) and the size of cluster \( \pi_i \)

\[
Purity(\pi_i) = \frac{1}{n_i} \max_j (n_{ij}) \quad j \in C
\]

- Others are entropy of classes in clusters (or mutual information between classes and clusters)

Cluster I: Purity = \( \frac{1}{6} \times \max(5, 1, 0) = \frac{5}{6} \)

Cluster II: Purity = \( \frac{1}{6} \times \max(1, 4, 1) = \frac{4}{6} \)

Cluster III: Purity = \( \frac{1}{5} \times \max(2, 0, 3) = \frac{3}{5} \)
Using Link Structure: HITS

- HITS – Hypertext Induced Topic Selection
- For each vertex $v \in V$ in a subgraph of interest:
  - $a(v)$ - the authority of $v$
  - $h(v)$ - the hubness of $v$
- A site is very authoritative if it receives many citations.
  - Citation from important sites weight more than citations from less-important sites
- Hubness shows the importance of a site.
  - A good hub is a site that links to many authoritative sites

Authority and Hub

- Column vector $a$: $a_i$ is the authority score for the $i$-th site
- Column vector $h$: $h_i$ is the hub score for the $i$-th site
- Matrix $M$:
  $$M_{i,j} = \begin{cases} 1 & \text{the } i\text{th site points to the } j\text{th site} \\ 0 & \text{otherwise} \end{cases}$$

$$M = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$
Authority and Hub

- Column vector $a$: $a_i$ is the authority score for the $i$-th site
- Column vector $h$: $h_i$ is the hub score for the $i$-th site
- Matrix $M$:
  $$
  M_{i,j} = \begin{cases} 
  1 & \text{the } i\text{th site points to the } j\text{th site} \\
  0 & \text{otherwise}
  \end{cases}
  $$
- Recursive dependency:
  $$
  a(v) \leftarrow \sum_{w \in \text{pa}[v]} h(w) \\
  h(v) \leftarrow \sum_{w \in \text{ch}[v]} a(w)
  $$
  $$
  a_i = \alpha_i M^T h_i \\
  h_i = \beta_i M a_i
  $$

Page Rank

- Matrix $M$:
  $$
  M_{i,j} = \begin{cases} 
  1 & \text{the } i\text{th site points to the } j\text{th site} \\
  0 & \text{otherwise}
  \end{cases}
  $$
- Matrix $B$:
  $$
  B_{i,j} = \begin{cases} 
  \frac{1}{\sum_j M_{i,j}} & \sum_j M_{i,j} > 0 \\
  0 & \text{otherwise}
  \end{cases}
  $$
- Matrix $M$:
  $$
  M = \begin{pmatrix}
  0 & 1 & 1 & 1 & 1 & 0 & 1 \\
  1 & 0 & 0 & 0 & 0 & 0 & 0 \\
  1 & 1 & 0 & 0 & 0 & 0 & 0 \\
  0 & 1 & 1 & 0 & 1 & 0 & 0 \\
  1 & 0 & 1 & 0 & 1 & 0 & 0 \\
  1 & 0 & 0 & 1 & 0 & 0 & 0 \\
  0 & 0 & 0 & 1 & 0 & 0 & 0
  \end{pmatrix}
  $$
- Matrix $B$:
  $$
  B = \begin{pmatrix}
  0 & 1/5 & 1/5 & 1/5 & 0 & 1/5 \\
  1/2 & 0 & 0 & 0 & 0 & 0 \\
  0 & 1/2 & 0 & 0 & 0 & 0 \\
  0 & 0 & 1/4 & 1/4 & 0 & 1/4 \\
  1/2 & 0 & 0 & 0 & 1/2 & 0 \\
  0 & 0 & 0 & 0 & 1 & 0 \\
  0 & 0 & 0 & 0 & 0 & 0
  \end{pmatrix}
  $$
Matrix Notation

\( r : r_i \) represents the rank score for the i-th web page

\[
    r(v) = \alpha \sum_{w \in pa[v]} \frac{r(w)}{|ch[w]|'}
\]

\( r = \alpha B' r \)

\( \alpha : \) eigenvalue

\( r : \) eigenvector of \( B \)

Finding Pagerank

→ find principle eigenvector of \( B \)
Random Walk Model

• Consider a random walk through the Web graph

\[
B = \begin{pmatrix}
0 & 1/5 & 1/5 & 1/5 & 0 & 1/5 \\
1/2 & 0 & 0 & 0 & 0 & 0 \\
0 & 1/3 & 0 & 1/3 & 0 & 0 \\
1/4 & 0 & 1/4 & 0 & 1/4 & 0 \\
1/2 & 0 & 0 & 1/2 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0
\end{pmatrix}
\]
Random Walk Model

- Consider a random walk through the Web graph

\[ \mathbf{B} = \begin{pmatrix}
0 & 1/5 & 1/5 & 1/5 & 0 & 1/5 \\
1 & 0 & 0 & 0 & 0 & 0 \\
1/2 & 1/2 & 0 & 0 & 0 & 0 \\
0 & 1/3 & 1/3 & 0 & 1/3 & 0 \\
1/4 & 1/4 & 1/4 & 0 & 1/4 & 0 \\
1/2 & 0 & 0 & 0 & 1/2 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
\end{pmatrix} \]

\[ p(k) \text{: percentage of time that the surfer will stay at the } i\text{-th site} \]

\[ p(k) = \sum_{i} p(i)\mathbf{B}_{i,k} \]

\[ \mathbf{p} = \mathbf{B}^T \mathbf{p} \]

Problem

- “Rank Sink” Problem
  - Many Web pages have no inlinks
  - Results in dangling edges in the graph

\[ \mathbf{B} = \begin{pmatrix}
0 & 1/5 & 1/5 & 1/5 & 0 & 1/5 \\
1 & 0 & 0 & 0 & 0 & 0 \\
1/2 & 1/2 & 0 & 0 & 0 & 0 \\
0 & 1/3 & 1/3 & 0 & 1/3 & 0 \\
1/4 & 1/4 & 1/4 & 0 & 1/4 & 0 \\
1/2 & 0 & 0 & 0 & 1/2 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
\end{pmatrix} \]

\[ r\text{(new page)} = 0 \]
Problem

- “Rank Sink” Problem
  - Many Web pages have no outlinks
  - Results in dangling edges in the graph

\[
B = \begin{bmatrix}
0 & 1/5 & 1/5 & 1/5 & 1/5 & 0 & 1/5 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1/2 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1/3 & 1/3 & 0 & 1/3 & 0 & 0 \\
1/4 & 0 & 1/4 & 1/4 & 0 & 1/4 & 0 \\
1/2 & 0 & 0 & 1/2 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\]

\[r(\text{new page}) = 1\]

Distribution of the Mixture Model

\[
H_{i,j} = \frac{1}{n}
\]

\[
B' = \epsilon H + (1 - \epsilon)B
\]

\[
r = B'^T r
\]

Prevents the page ranks from being 0 or 1
Web Crawler

- Starts with a set of seeds, which are a set of URLs given to it as parameters
- Seeds are added to a URL request queue
- Crawler starts fetching pages from the request queue
- Downloaded pages are parsed to find link tags that might contain other useful URLs to fetch
- New URLs added to the crawler’s request queue, or frontier

Basic crawl architecture
URL frontier: two main considerations

- **Politeness**: do not hit a web server too frequently
- **Freshness**: crawl some pages more often than others
  - E.g., pages (such as News sites) whose content changes often

These goals may conflict with each other. (E.g., simple priority queue fails – many links out of a page go to its own site, creating a burst of accesses to that site.)

---

URL frontier: Mercator scheme

- URLs
- Prioritizer
- $K$ front queues
  - Biased front queue selector
  - Back queue router
- $B$ back queues
  - Single host on each
  - Back queue selector
- Crawl thread requesting URL
Front queues

- Prioritizer assigns to URL an integer priority between 1 and $K$
  - Appends URL to corresponding queue
- Heuristics for assigning priority
  - Refresh rate sampled from previous crawls
  - Application-specific (e.g., “crawl news sites more often”)

Biased front queue selector

- When a back queue requests a URL (in a sequence to be described): picks a front queue from which to pull a URL
- This choice can be round robin biased to queues of higher priority, or some more sophisticated variant
  - Can be randomized
Back queue processing

- A crawler thread seeking a URL to crawl:
  - Extracts the root of the heap
  - Fetches URL at head of corresponding back queue \( q \) (look up from table)
  - Checks if queue \( q \) is now empty – if so, pulls a URL \( v \) from front queues
    - If there’s already a back queue for \( v \)’s host, append \( v \) to it and pull another URL from front queues, repeat
    - Else add \( v \) to \( q \)
  - When \( q \) is non-empty, create heap entry for it

Freshness

- Web pages are constantly being added, deleted, and modified
- Web crawler must continually revisit pages it has already crawled to see if they have changed in order to maintain the freshness of the document collection
  - stale copies no longer reflect the real contents of the web pages
Age

- Expected age of a page \( t \) days after it was last crawled:
  
  \[
  \text{Age}(\lambda, t) = \int_{0}^{t} P(\text{page changed at time } x)(t - x)dx
  \]

- Web page updates follow the Poisson distribution on average
  - time until the next update is governed by an exponential distribution
    
    \[
    \text{Age}(\lambda, t) = \int_{0}^{t} \lambda e^{-\lambda x}(t - x)dx
    \]

Detecting Duplicates

- Duplicate and near-duplicate documents occur in many situations
  - Copies, versions, plagiarism, spam, mirror sites
  - 30% of the web pages in a large crawl are exact or near duplicates of pages in the other 70%

- Duplicates consume significant resources during crawling, indexing, and search
  - Little value to most users
Computing Similarity

- Features:
  - Segments of a document (natural or artificial breakpoints)
  - Shingles (Word N-Grams)
  - *a rose is a rose is a rose* \(\rightarrow\) 4-grams are
    - `a_rose_is_a`
    - `rose_is_a_rose`
    - `is_a_rose_is`
    - `a_rose_is_a`
- Similarity Measure between two docs (= sets of shingles)
  - Jaccard coefficient: \(\frac{\text{Size of Intersection}}{\text{Size of Union}}\)

Sketch of a document

- Create a “sketch vector” (of size \(\sim 200\)) for each document
  - Documents that share \(\geq t\) (say 80%) corresponding vector elements are deemed near duplicates
  - For doc \(D\), sketch \(D[i]\) is as follows:
    - Let \(f\) map all shingles in the universe to \(1..2^m\)
      (e.g., \(f =\) fingerprinting)
    - Let \(\pi_i\) be a *random permutation* on \(1..2^m\)
    - Pick \(\text{MIN}\{\pi_i(f(s))\}\) over all shingles \(s\) in \(D\)
Computing Sketch[i] for Doc1

Document 1

- Start with $2^{64}$-bit $f$(shingles)
- Permute on the number line with $\pi_i$
- Pick the min value

Test if Doc1.Sketch[i] = Doc2.Sketch[i]

Document 1

- Are these equal?

Test for 200 random permutations: $\pi_1, \pi_2, \ldots, \pi_{200}$
Collaborative Filtering

What we have:
• Assume there are some ratings by training users
• Test user provides some amount of additional training data

What we do:
• Predict test user’s rating based training information

Objects: \( O_m \)

<table>
<thead>
<tr>
<th>( O_1 )</th>
<th>( O_2 )</th>
<th>( O_3 )</th>
<th>\ldots</th>
<th>( O_j )</th>
<th>\ldots</th>
<th>( O_M )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( U_1 )</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( U_2 )</td>
<td></td>
<td>4</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \ldots )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( U_i )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \ldots )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( U_N )</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Test User \( U_t \)

\( R_{nt}(O_j) = ? \)

Memory-Based Approaches

• Memory-Based Approaches
  – Given a specific user \( u \), find a set of similar users
  – Predict \( u \)'s rating based on ratings of similar users

• Issues
  – How to determine the similarity between users?
  – How to combine the ratings from similar users to make the predictions (how to weight different users)?
Memory-Based Approaches

• How to determine the similarity between users?
  – Measure the similarity in rating patterns between different users

Pearson Correlation Coefficient Similarity

\[ W_{u,u'} = \frac{\sum (R_{u'}(o) - \bar{R}_{u'})(R_u(o) - \bar{R}_u)}{\sqrt{\sum (R_{u'}(o) - \bar{R}_{u'})^2 \sum (R_u(o) - \bar{R}_u)^2}} \]

Vector Space Similarity

\[ W_{u,u'} = \frac{\sum R_{u'}(o)R_u(o)}{\sqrt{\sum R_{u'}(o)^2 \sum R_u(o)^2}} \]

Average Ratings

Prediction:

\[ \hat{R}_{u'}(o) = \bar{R}_{u'} + \frac{\sum_{u} W_{u,u'} (R_u(o) - \bar{R}_u)}{\sum_{u} W_{u,u'}} \]
### Memory-Based Approaches

<table>
<thead>
<tr>
<th></th>
<th>Sub Mean (Train1)</th>
<th>Sub Mean (Train2)</th>
<th>Sub Mean (Test)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train_User 1</td>
<td>-2.2</td>
<td>1</td>
<td>0.333</td>
</tr>
<tr>
<td>Train_User 2</td>
<td>1</td>
<td>-2</td>
<td>1.33</td>
</tr>
<tr>
<td>Test User</td>
<td>-1.667</td>
<td>0.333</td>
<td>1.33</td>
</tr>
</tbody>
</table>

Normalize Rating

Calculate Similarity: \( W_{trn1\_test}=0.92; W_{trn2\_test}=-0.44; \)
## Memory-Based Approaches

<table>
<thead>
<tr>
<th></th>
<th>Movie 1</th>
<th>Movie 2</th>
<th>Movie 3</th>
<th>Movie 4</th>
<th>Movie 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train_User 1</td>
<td>1</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Sub Mean (Train1)</td>
<td>-2.2</td>
<td>1.8</td>
<td>-0.2</td>
<td>-0.2</td>
<td>0.8</td>
</tr>
<tr>
<td>Train_User 2</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Sub Mean (Train2)</td>
<td>1</td>
<td>-2</td>
<td>2</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>Test User</td>
<td>1</td>
<td>?</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Sub Mean (Test)</td>
<td>-1.667</td>
<td>0.333</td>
<td>1.33</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Make Prediction:** \[ 2.67 + (1.8 \times 0.92 + (-2) \times (-0.44)) / (0.92 + 0.44) = 4.54 \]
Collaborative Filtering

- Flexible Mixture Model (FMM):
  Cluster users and objects separately AND allow them to belong to different classes
  \[ P(o_{(i)}, u_{(i)}, r_{(i)}) = \sum_{Z_u, Z_o} P(Z_u) P(Z_o) P(o_{(i)} | Z_u) P(u_{(i)} | Z_o) P(r_{(i)} | Z_u, Z_o) \]

- Training Procedure:
  Annealed Expectation Maximization (AEM) algorithm
  E-Step: Calculate Posterior Probabilities
  \[ P(z_{(i)}, z_u | o_{(i)}, u_{(i)}, r_{(i)}) = \frac{(P(Z_u) P(Z_o) P(o_{(i)} | Z_u) P(u_{(i)} | Z_o) P(r_{(i)} | Z_u, Z_o))^\theta}{\sum_{z_u, z_o} (P(Z_u) P(Z_o) P(o_{(i)} | Z_u) P(u_{(i)} | Z_o) P(r_{(i)} | Z_u, Z_o))^\theta} \]

- Prediction Procedure:
  Fold-in process to calculate joint probabilities
  \[ P(o, u', r_{(i)}) = \sum_{Z_u, Z_o} P(Z_o) P(Z_u) P(o | Z_o) P(u' | Z_u) P(r | Z_o, Z_u) \]
  Fold-in process by EM algorithm
  Calculate expectation for prediction
  \[ R_{u'}^{(o)} = \sum_r r \frac{P(o, u', r)}{\sum_{r'} P(o, u', r')} \]

“Flexible Mixture Model for Collaborative Filtering”, ICML’03
Decoupled Model (DM)

- **Decoupled Model (DM):**
  Separate preference value

  \[ Z_{pref} \in \{1, \ldots, k\} \quad (1 \text{ disfavor, } k \text{ favor}) \]

  from rating \( r \in \{1,2,3,4,5\} \)

- **Joint Probability:**

  \[
  P(o_{(i)}, u_{(i)}, r_{(i)}) = \sum_{z_{o}, z_{u}, z_{r}} P(z_{o}) P(z_{u}) P(o_{(i)} | z_{o}) P(u_{(i)} | z_{u}) \sum_{r_{(i)}} \prod_{z_{o}, z_{u}, z_{r}} P(r_{(i)} | z_{o}, z_{u}, z_{r})
  \]

  "Preference-Based Graphical Model for Collaborative Filtering", UAI’03

  "A study of Mixture Model for Collaborative Filtering", Journal of IR

---

And too recent to “review”:
(you should still remember these…)

- **Web Crawling**
  - Crawler architecture
  - Duplicate detection
  - Freshness, politeness

- **Collaborative Filtering**
  - Memory-based, model-based
  - Content-based filtering

- **Natural Language Processing**
  - Question Answering
  - Sentiment analysis

- **Federated Search**
  - Source representation, selection, result merging
  - Crawling the Deep Web

- **Cross-Lingual IR**

- **Map-Reduce**

- **Fake News and other Bad Actors**
  - Search Engine Optimization
  - Fake news discovery
  - Click fraud