Information Retrieval Methods for Software Engineering

Andrian Marcus

Wayne State University

Software Visualization and Evolution REsearch GROUP

with substantial contributions from Giuliano Antoniol

Why use information retrieval in software engineering?
Information in Software

- **Structural information** - the structural aspects of the source code (e.g., control and data flow)

- **Dynamic information** - behavioral aspects of the program (e.g., execution traces)

- **Lexical information** - captures the problem domain and developer intentions (e.g., identifiers, comments, documentation, etc.)

- **Process information** - Evolutionary data, history of changes (e.g., CVS logs, bug reports, etc.)

Why Analyze the Textual Information?

- Software = text, structure, behavior
- Text -> *what is the software doing?*
- Structure + behavior -> *how is the software doing it?*
- We need all three for complete code view and comprehension
- Text is the common form of information representation among various software artifacts at different abstraction levels
How to Analyze the Text in Software?

- Natural Language Processing (NLP)
- WordNet
- Ontologies
- Information/Text Retrieval (IR/TR)

- Combinations of the above

What is information retrieval?
What is Information Retrieval?

- The process of actively seeking out information relevant to a topic of interest (van Rijsbergen)
  - Typically it refers to the automatic (rather than manual) retrieval of documents
  - Document - generic term for an information holder (book, chapter, article, webpage, class body, method, requirement page, etc.)

Information Retrieval System (IRS)

- An Information Retrieval System is capable of storage, retrieval, and maintenance of information (e.g., text, images, audio, video, and other multi-media objects)
- Difference from DBMS
  - used on unstructured information
  - indexing mechanism used to define “keys”
IR in Practice

- Information Retrieval is a research-driven theoretical and experimental discipline
  - The focus is on different aspects of the information-seeking process, depending on the researcher’s background or interest:
    - Computer scientist - fast and accurate search engine
    - Librarian - organization and indexing of information
    - Cognitive scientist - the process in the searcher’s mind
    - Philosopher - is this really relevant?
    - Etc.
  - Progress influenced by advances in Computational Linguistics, Information Visualization, Cognitive Psychology, HCI, ...

What Do We Want From an IRS?

- Systemic approach
  - Goal (for a known information need):
    - Return as many relevant documents as possible and as few non-relevant documents as possible

- Cognitive approach
  - Goal (in an interactive information-seeking environment, with a given IRS):
    - Support the user’s exploration of the problem domain and the task completion.
Disclaimer

• We are IR users and we’ll take a simple view: a document is relevant if it is about the searcher’s topic of interest
• As we deal with software artifacts, mostly source code and other artifact textual representations, we will focus on text documents, not other media
  - Most current tools that search for images, video, or other media rely on text annotations
  - Real content retrieval of other media (based on shape, color, texture, ...) are not mature yet

What is Text Retrieval?

• TR = IR of textual data
  - a.k.a. document retrieval
• Basis for internet search engines
• Search space is a collection of documents
• Search engine creates a cache consisting of indexes of each document - different techniques create different indexes
Advantages of Using TR

- No predefined grammar and vocabulary
- Some techniques able to infer word relationships without a thesaurus or an ontology
- Robust with respect to data distribution and type

Terminology

- Document = unit of text - set of words
- Corpus = collection of documents
- Term vs. word - basic unit of text - not all terms are words
- Query
- Index
- Rank
- Relevance
A Typical TR Application

- Build corpus
- Index corpus
1. Formulate a query (Q)
   - Can be done by the user or automatically
2. Compute similarities between Q and the documents in the corpus
3. Rank the documents based on the similarities
4. Return the top N as the result
5. Inspect the results
6. GO TO 1. if needed or STOP

Document-Document Similarity

- Document representation
  - Select features to characterize document: terms, phrases, citations
  - Select weighting scheme for these features:
    - Binary, raw/relative frequency, ...
    - Title / body / abstract, selected topics, taxonomy
- Similarity / association coefficient or dissimilarity / distance metric
Similarity [Lin 98, Dominich 00]

- Given a set $X$, a similarity on $X$ is a function:
  - Co-domain: for all points $x, y$ in $X$
    \[ 0 \leq \sigma(x, y) \leq 1 \]
  - Symmetry: for all points $x, y$ in $X$
    \[ \sigma(x, y) = \sigma(y, x) \]
  - And for all $x, y$ in $X$ if $x == y$
    \[ \sigma(x, y) = 1 \]

Association Coefficients

- Simple matching
  \[ \frac{|X \cap Y|}{|X| + |Y|} \]
  \[ \sum_i x_i y_i \]

- Dice’s coefficient
  \[ \frac{2 \cdot |X \cap Y|}{|X| + |Y|} \]
  \[ \frac{2 \cdot \sum_i x_i y_i}{\sum_i x_i^2 + \sum_i y_i^2} \]

- Cosine coefficient
  \[ \frac{|X \cap Y|}{|X| \cdot |Y|} \]
  \[ \frac{\sum_i x_i \cdot y_i}{\sqrt{\sum_i x_i^2 \cdot \sum_i y_i^2}} \]

- Jaccard coefficient
  \[ \frac{|X \cap Y|}{|X| + |Y| - |X \cap Y|} \]
  \[ \frac{\sum_i x_i y_i}{\sum_i x_i^2 + \sum_i y_i^2 - \sum_i x_i y_i} \]
Information retrieval techniques?

Classification of IR Models

<table>
<thead>
<tr>
<th>Mathematical Basis</th>
<th>Properties of the Model</th>
<th>without term-interdependencies</th>
<th>with term-interdependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>set-theoretic</td>
<td></td>
<td>Standard Boolean</td>
<td>Fuzzy Set</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Extended Boolean</td>
<td></td>
</tr>
<tr>
<td>algebraic</td>
<td></td>
<td>Vector Space</td>
<td>Generalised Vector Space</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Topic-based Vector Space</td>
</tr>
<tr>
<td>probabilistic</td>
<td></td>
<td>Binary Interdependence</td>
<td>Retrieval by Logical Imaging</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Inference Network</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Belief Network</td>
<td></td>
</tr>
</tbody>
</table>
Most Popular Models Used in SE

- Vector Space Model (VSM)
- Latent Semantic Indexing (LSI)
- Probabilistic Models
- Latent Dirichlet Allocation (LDA)

Document Vectors

- Documents are represented vectors, which represent “bags of words”
  - the ordering of words in a document is ignored:
    “John is quicker than Mary” and “Mary is quicker than John” have the same vectors
- Represented as vectors when used computationally
  - A vector is like an array of floating point
  - Has direction and magnitude
  - Each vector holds a place for every term in the collection
    - most vectors are sparse
Vector Space Model

- Documents are represented as vectors in the term space
  - Terms are usually stems a.k.a. word root
  - Documents represented by binary vectors of terms
- Queries are represented same as documents
- A vector similarity measure between the query and documents is used to rank retrieved documents
  - Query and Document similarity is based on length and direction of their vectors
  - Vector operations to capture Boolean query conditions
  - Terms in a vector can be “weighted” in many ways

The Vector-Space Model

- Assume $t$ distinct terms remain after preprocessing
  - call them index terms or the vocabulary.
- These “orthogonal” terms form a vector space.
  - Dimension = $t = |\text{vocabulary}|$
- Each term, $i$, in a document or query, $j$, is given a real-valued weight, $w_{ij}$.
- Both documents and queries are expressed as $t$-dimensional vectors:
  $$d_j = (w_{1j}, w_{2j}, ..., w_{tj})$$
Document Vectors

<table>
<thead>
<tr>
<th>DocID</th>
<th>Nova</th>
<th>Galaxy</th>
<th>Film</th>
<th>Role</th>
<th>Diet</th>
<th>Fur</th>
<th>Web</th>
<th>Tax</th>
<th>Fruit</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D2</td>
<td>3</td>
<td>7</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D3</td>
<td></td>
<td>4</td>
<td>11</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D4</td>
<td></td>
<td></td>
<td>9</td>
<td>4</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D5</td>
<td>4</td>
<td>7</td>
<td>9</td>
<td></td>
<td>5</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Document Collection

- A collection of $n$ documents can be represented in the VSM by a term-document matrix.
- An entry in the matrix corresponds to the "weight" of a term in the document; zero means the term has no significance in the document or it simply doesn’t exist in the document.
Graphic Representation

Example:

\[ D_1 = 2T_1 + 3T_2 + 5T_3 \]
\[ D_2 = 3T_1 + 7T_2 + T_3 \]
\[ Q = 0T_1 + 0T_2 + 2T_3 \]

• Is \( D_1 \) or \( D_2 \) more similar to \( Q \)?
• How to measure the degree of similarity? Distance? Angle? Projection?

Term Weights - Local Weights

• The weight of a term in the document-term matrix \( w_{ik} \) is a combination of a local weight \( l_{ik} \) and a global weight \( g_{ik} \): \( w_{ik} = l_{ik} \times g_{ik} \).
• Local weights \( l_{ik} \): used to indicate the importance of a term relative to a particular document. Examples:
  - term frequency \( (tf_{ik}) \): number of times term \( i \) appears in doc \( k \) (the more a term appears in a doc, the more relevant it is to that doc)
  - log-term frequency \( (log \ tf_{ik}) \): mitigates the effect of \( tf \) - relevance does not always increase proportionally with term frequency
  - binary \( (b_{ik}) \): 1 if term \( i \) appears in doc \( k \), 0 otherwise
Term Weights - Global Weights

- Global weight \( g_{ik} \): used to indicate the importance of a term relative to the entire document collection. Used as an indication of a term’s discrimination power. Examples:
  - document frequency (df_{i}): number of documents containing term \( i \); rare terms are more informative than frequent terms; \( df_{i} \) is an inverse measure of the informativeness of \( t \)
  - inverse document frequency (idf_{i}): \( idf_{i} = \log_{2}(N/df_{i}) \)
    - \( N \): total number of documents; log is used to “dampen” the effect of \( tf \)

TF x IDF Calculation

\[ w_{ik} = tf_{ik} \times \log\left(\frac{N}{n_{k}}\right) \]

- \( T_{k} \) = term \( k \) in document \( D_{i} \)
- \( tf_{ik} \) = frequency of term \( T_{k} \) in document \( D_{i} \)
- \( idf_{k} \) = inverse document frequency of term \( T_{k} \) in \( C \)
- \( N \) = total number of documents in the collection \( C \)
- \( n_{k} \) = the number of documents in \( C \) that contain \( T_{k} \)
- \( idf_{k} = \log\left(\frac{N}{n_{k}}\right) \)
Inverse Document Frequency

- IDF provides high values for rare words and low values for common words

For a collection of 10,000 documents:

\[
\begin{align*}
\log \left( \frac{10,000}{10,000} \right) &= 0 \\
\log \left( \frac{10,000}{5,000} \right) &= 0.301 \\
\log \left( \frac{10,000}{20} \right) &= 2.698 \\
\log \left( \frac{10,000}{1} \right) &= 4
\end{align*}
\]

Computing TF-IDF -- An Example

Given a document D1 containing terms with frequencies:

A(3), B(2), C(1)

Assume collection contains 10,000 documents; assume document frequencies of these terms are:

A(50), B(1300), C(250)

Then:

A: \( tf = 3/3; \quad idf = \log(10000/50) = 5.3; \quad tf\text{-idf} = 5.3 \)
B: \( tf = 2/3; \quad idf = \log(10000/1300) = 2.0; \quad tf\text{-idf} = 1.3 \)
C: \( tf = 1/3; \quad idf = \log(10000/250) = 3.7; \quad tf\text{-idf} = 1.2 \)
Vector Space “Relevance” Measure

\[ D_i = w_{d_{i1}}, w_{d_{i2}}, ..., w_{d_{in}} \]
\[ Q = w_{q1}, w_{q2}, ..., w_{qt} \]  \( w = 0 \) if a term is absent

if term weights normalized:  \( \text{sim}(Q, D_i) = \sum_{j=1}^{t} w_{qj} \times w_{dj} \)

otherwise normalize in the similarity comparison:

\[
\text{sim}(Q, D_i) = \frac{\sum_{j=1}^{t} w_{qj} \times w_{dj}}{\sqrt{\sum_{j=1}^{t} (w_{qj})^2 \times \sum_{j=1}^{t} (w_{dj})^2}}
\]

Computing Relevance Scores

Say we have query vector \( Q = (0.4, 0.8) \)
Also, document \( D_2 = (0.2, 0.7) \)

What does their similarity comparison yield?

\[
\text{sim}(Q, D_2) = \frac{(0.4 \times 0.2) + (0.8 \times 0.7)}{\sqrt{[(0.4)^2 + (0.8)^2] \times [(0.2)^2 + (0.7)^2]}}
\]

\[
= \frac{0.64}{\sqrt{0.42}} = 0.98
\]
Vector Space with Term Weights and Cosine Matching

\[ \text{sim}(Q, D) = \frac{\sum_{j=1}^{t} w_{qj} w_{dj}}{\sqrt{\sum_{j=1}^{t} (w_{qj})^2 \sum_{j=1}^{t} (w_{dj})^2}} \]

\[ \text{sim}(Q, D_1) = \frac{(0.4 \cdot 0.8) + (0.8 \cdot 0.3)}{\sqrt{[(0.4)^2 + (0.8)^2] \cdot [(0.2)^2 + (0.7)^2]}} \]
\[ = \frac{0.64}{\sqrt{0.42}} = 0.98 \]

\[ \text{sim}(Q, D_2) = \frac{0.8}{\sqrt{0.58}} = 0.74 \]

Latent Semantic Indexing

- Why need it?
  - some problems for retrieval methods based on term matching
    - vector-space similarity approach works only if the terms of the query are explicitly present in the relevant documents
  - rich expressive power of natural language
    - often queries contain terms that express concepts related to text to be retrieved
  - With the vector space model, we are assuming independence among terms in a document
    - ... however we know this is not true!!
Two Problems

- The same concept can be expressed using different sets of terms (*synonyms*)
  - e.g. *bandit*, *brigand*, *thief*
- Negatively affects recall
- Identical terms can be used in very different semantic contexts (*homonyms*)
  - e.g. *bank*, *chip*
  - repository where important material is saved
  - the slope beside a body of water
- Negatively affects precision

Idea

- **Idea** (*Deerwester et al.)*:
  “We would like a representation in which a set of terms, which by itself is incomplete and unreliable evidence of the relevance of a given document, is replaced by some other set of entities which are more reliable indicants. We take advantage of the implicit higher-order (or latent) structure in the association of terms and documents to reveal such relationships.”
Using SVD

• LSI uses linear algebra technique called singular value decomposition (SVD)
  - attempts to estimate the hidden structure
  - discovers the most important associative patterns between words and concepts
• In other words...
  - The analysis is moved from the space of terms to the space of concepts/topics
• Data driven

What is SVD?

• Given a term to document matrix $X$ with $n$ terms and $m$ documents
• SVD decomposes a matrix into three matrices
  $$X = U\Sigma V^T$$
  - $\Sigma$ is a $k \times k$ diagonal matrix containing singular values
    - where $k$ is the rank of $X$
• $U$ ($m \times k$) and $V$ ($k \times n$) contains eigenvectors, i.e., linearly independent vectors
Basically...

- Instead of representing documents as a set of correlated factors (terms), we represent documents as set of uncorrelated factors (concepts)
- Some of these factors in the orthonormal matrices $U$ and $V$ are very small
- We can ignore them by setting them to zero

SVD: Dimensionality Reduction

Forced to 0
LSI Example

- A collection of documents:
  d1: Indian government goes for open-source software
  d2: Debian 3.0 Woody released
  d3: Wine 2.0 released with fixes for Gentoo 1.4 and Debian 3.0
  d4: gnuPOD released: iPOD on Linux... with GPLed software
  d5: Gentoo servers running at open-source mySQL database
  d6: Dolly the sheep not totally identical clone
  d7: DNA news: introduced low-cost human genome DNA chip
  d8: Malaria-parasite genome database on the Web
  d9: UK sets up genome bank to protect rare sheep breeds
  d10: Dolly’s DNA damaged

LSI Example:

term-documents matrix

<table>
<thead>
<tr>
<th></th>
<th>d1</th>
<th>d2</th>
<th>d3</th>
<th>d4</th>
<th>d5</th>
<th>d6</th>
<th>d7</th>
<th>d8</th>
<th>d9</th>
<th>d10</th>
</tr>
</thead>
<tbody>
<tr>
<td>open-source</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>software</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Linux</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>released</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Debian</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Gentoo</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>database</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Dolly</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>sheep</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>genome</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DNA</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Some Cosine Similarities

- \( \text{sim}(d_1, d_3) = 0 \)
- \( \text{sim}(d_1, d_4) = 0.3 \)
- \( \text{sim}(d_1, d_5) = 0.7 \)
- \( \text{sim}(d_7, d_{10}) = 0.63 \)
- \( \text{sim}(d_8, d_{10}) = 0 \)

...However \( d_3 \) is about (open source) Linuxes, and \( d_8 \) about (DNA) manipulation

Reconstructed Term-Document Matrix (\( k=2 \))

\[ X' = U' \cdot \Sigma' \cdot V^T \]
How to Choose k?

- Finding optimal dimension for semantic space
  - precision-recall improve as dimension is increased until hits optimal, then slowly decreases until it hits standard vector model
  - run SVD once with big dimension, say $k = 1000$
    - then can test dimensions $\leq k$
  - in many tasks 150-350 works well, still room for research
- A lot depends on the application
- There are also procedures to automatically choose $k$

LSI: Pros and Cons

- LSI:
  + Able to deal with synonymy and homonymy
  + Stemming could be avoided
    + However it works better with stemming!
  + Increases similarity between documents of the same cluster
  + Decreases similarity between documents of different clusters
  - More expensive than traditional Vector Space Models (SVD computation)
  - Difficult to add new documents
  - Determining the optimal $k$ is a crucial issue
  - Often needs a large document corpus
Probabilistic Models

- Rigorous formal model attempts to predict the probability that a given document will be relevant to a given query

- Ranks retrieved documents according to this probability of relevance
  (Probability Ranking Principle)

- Relies on accurate estimates of probabilities

Probabilistic Ranking

- Probabilistic ranking given a document \( d \) and a query \( q \):
  - \( \text{sim}(q,d) = \frac{P(d \text{ relevant-to } q)}{P(d \text{ non-relevant-to } q)} \)
  - This is the odds of the document \( d \) being relevant

- Underline model: documents are bags of words
How to Compute

- Terms that appear in previously retrieved relevant documents (for a query q) should be given higher weight
- Probabilistic indexing is more an iterative process requiring a few (known) relevant documents
  - is closer to relevance feedback
- Strong assumption: terms are independent

LSI Drawback

- LSA/LSI pretend that there are underlining concepts/topics
  - Words are observable
  - Topics/concepts are not
- LSI doesn’t tell us how to automatically estimate topics/concepts
- Topics are somehow a summarization of words conveyed concept(s)
Probabilistic Latent Semantic Analysis (PLSA)

- Suppose we have $K$ concepts/topics

\[ P(\text{term}_i \mid \text{doc}) = \sum_{j=1}^{K} P(\text{term}_i \mid \text{concept}_j)P(\text{concept}_j \mid \text{doc}) \]

\[ P(\text{doc}) = \prod_{i=1}^{T} P(\text{term}_i \mid \text{doc}) = \]

\[ = \prod_{i=1}^{T} \sum_{j=1}^{K} P(\text{term}_i \mid \text{concept}_j)P(\text{concept}_j \mid \text{doc}) \]

PLSA Parameters

- PLSA parameters are:

\[ P(\text{concept}_j \mid \text{doc}) \]

\[ P(\text{term}_i \mid \text{concept}_j) \]

- Estimation via fix-point and Maximum Likelihood
- Input the term-document matrix and the number of topics/concepts $K$
PLSA Problem

- Each document is represented as a list of numbers (the concepts mixing proportions)
  - there is no generative probabilistic model for these proportions
- The parameter number grows linearly with the number of documents
- PLSA suffer of over fitting (sparse documents)
- It is a generative model only on the document collection it was estimated
  - how do we assign a probability to an unseen document?
  - new documents are still a problem as in LSA

Latent Dirichlet Allocation (LDA)

- Documents are represented as random mixtures over latent concepts/topics
- Each concept/topic is characterized by a distribution over words
- Each word is attributable to one of the concepts/topics of the document
- The topic distribution is assumed to have a Dirichlet prior
  - a continuous multivariate probability distribution depending from a parameter vector
Latent Dirichlet Allocation (2)

- Given a document-word matrix
  - Probabilistically determine X most likely topics
  - For each topic determine Y most likely words
  - Do it without human intervention
- Humans do not supply hints for topic list
- Humans do not tune algorithm on the fly
- No need for iterative refinement
- Output
  - Document-Topic Matrix
  - Topic-Word Matrix

LDA Document Generation

- Suppose we have two concepts:
  - corrective maintenance and enhancement
- Both concepts will generate the words “defect” and “improvement”
  - the probability of “defect” will be higher in the concept corrective maintenance
  - In corrective maintenance the probability of the term “defect” will be higher than the probability of the “term improvement”
LDA Document Generation (2)

• Chose a distribution over concepts/topics:
  - Mostly corrective maintenance, mostly enhancement a mix of the two ....

• Parameters:
  - K - the number of topics
  - The Dirichlet prior

• Output Dirichlet parameter vector and thus topics distribution over words

Is This All?

• No ... there is more
  - Jensen-Shannon divergence
    • model documents as probability distributions
    • Jensen-Shannon divergence measures the distance between probabilities distribution

  - and more ...
Evaluation of information retrieval applications?

Evaluating TR Systems

\[
\text{recall} = \frac{\text{Number of relevant documents retrieved}}{\text{Total number of relevant documents}}
\]

\[
\text{precision} = \frac{\text{Number of relevant documents retrieved}}{\text{Total number of documents retrieved}}
\]
Computing Recall/Precision Points

- For a given query, produce the ranked list of retrievals.
- Adjusting a threshold on this ranked list produces different sets of retrieved documents, and therefore different recall/precision measures.
- Mark each document in the ranked list that is relevant according to the gold standard.
- Compute a recall/precision pair for each position in the ranked list that contains a relevant document.

Trade-off Between Recall and Precision

Returns relevant documents but misses many useful ones too

The ideal

Returns most relevant documents but includes lots of junk
Interpolating a Recall/Precision Curve: An Example

Compare Two or More Systems

- The curve closest to the upper right-hand corner of the graph indicates the best performance
F-Measure

- The traditional F-measure or balanced F-score (F1 score) is the harmonic mean of precision and recall

\[ F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}. \]

- The general formula for non-negative real \( \beta \) is

\[ F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{\beta^2 \cdot \text{precision} + \text{recall}}. \]

F-Measure Comparison
Computing Recall/Precision Points

- For a given query, produce the ranked list of retrievals.
- Any threshold on this ranked list produces different sets of retrieved documents. How about recall/precision?
- Mark each document in the ranked list that is relevant according to the gold standard.
- Compute a recall/precision pair for each position in the ranked list that contains a relevant document.

Ranked List Threshold(s)

- Fixed one -- take the first 10 results
- Variable such as score threshold: keep element with score in the top 5%
- Gap threshold:
  - traverse the ranked list (from highest to lowest score)
  - find the widest gap between adjacent scores
  - the score immediately prior to the gap becomes the threshold
Precision and Recall: the Holy Grail

- Precision and recall force to accept a trade-off
- Gold standard ... where is it?
- Precision and recall do not tell the entire story

Other Evaluation Methods

- The rank of the first relevant element in the ranked list
- The average of the rank of the first relevant element in the ranked list in a set of experiments - standard deviation
- Other possible measures of dispersion of relevant elements
  - Compare to the best possible answer
  - Position of the last relevant element
The “Accuracy” Measure

• Highly dependant on the SE task:
  - concept location
    • find the first relevant item in the ranked list
      - position of the first item
  - traceability verification in a mission critical application
    • find the last relevant item in the ranked list
      - position of the last item

Subjective Measures

• Novelty Ratio: The proportion of elements retrieved and judged relevant by the user and of which they were previously unaware
  - Ability to find new information on a concept

• Coverage Ratio: The proportion of relevant items retrieved out of the total relevant documents known to a user prior to the search
  - Relevant when the user wants to locate documents which they have seen before (e.g., the code region changed to fix a known bug).
Other Factors

- *User effort*: effort required to formulate queries, conduct the search, and screen the output.
- *Response time*: time interval between a user query and the presentation results.
- *Form of presentation*: impact the search the user’s ability to utilize the retrieved items.

Query formulation
Query Formulation

• Usually simple bag of words
  - Ex. “tutorial software engineering text retrieval antoniol marcus PASED”
• Boolean operators: and, or, not
• Natural language sentences or paragraphs
  - Ex. “How much longer will this tutorial last? I am getting sleepy.”
• Existing documents

Query Modification

• Problem: How can we reformulate the query to help a user who is trying several searches to get at the same information?
  - Thesaurus expansion:
    • Suggest terms similar to query terms
  - Relevance feedback:
    • Suggest terms (and documents) similar to retrieved documents that have been judged to be relevant
Query Analysis and Expansion

- Spellchecking -> change words
- Compare with vocabulary -> remove words
- Use thesaurus -> suggest alternative words (synonyms)

Relevance Feedback

- Main Idea:
  - Modify existing query based on relevance judgements
    - Extract terms from relevant documents and add them to the query
    - AND/OR re-weight the terms already in the query
- There are many variations:
  - Usually positive weights for terms from relevant docs
  - Sometimes negative weights for terms from non-relevant docs
- Users, or the system, guide this process by selecting terms from an automatically-generated list.
Let’s talk about software engineering

How to Use TR with Software?

- Building the corpus
- Document granularity
- Formulating SE tasks as TR problems
- Querying
Creating a Corpus of a Software System

- Parsing source code and extracting documents
  - corpus - collection of documents (e.g., methods)
- Removing non-literals and stop words
  - common words in English, standard function library names, programming language keywords
- Preprocessing: split_identifiers and SplitIdentifiers
- NLP methods can be applied such as stemming

Parsing Source Code and Extracting Documents

- Documents can be at different granularities (e.g., methods, classes, files)
Parsing Source Code and Extracting Documents

• Documents can be at different granularities (e.g., methods, classes, files)

Source Code is Text Too

```java
public void run(IProgressMonitor monitor) throws InvocationTargetException, InterruptedException {
    if (m_iFlag == 0) {
        processCorpus(monitor, checkUpdate());
    } else if (m_iFlag == 2) {
        processCorpus(monitor, UD_UPDATECORPUS);
    } else {
        processQueryString(monitor);
        if (monitor.isCanceled())
            throw new InterruptedException("The long running
```
Lexical Analysis

• Break up the text in words or “tokens”
• Question: “what is a word”?

• Problem cases
  - Numbers: “M16”, “2001”
  - Hyphenation: “MS-DOS”, “OS/2”
  - Punctuation: “John’s”, “command.com”
  - Case: “us”, “US”
  - Phrases: “venetian blind”

Splitting Identifiers

```java
public void run IProgressMonitor monitor throws InvocationTargetException
InterruptedException if m_iFlag the processCorpus monitor checkUpdate else if
m_iFlag processCorpus monitor UD_UPDATECORPUS else a processQueryString
monitor if monitor isCancelled throw new InterruptedException the long
running
```

• IProgressMonitor = i progress monitor
• InvocationTargetException = invocation target exception
• m_iFlag = m i flag
• UD_UPDATECORPUS = ud updatecorpus
Stop Words

- Very frequent words, with no power of discrimination (e.g., language keywords)

- Typically function words, not indicative of content

- The stopwords set depends on the document collection and on the application (e.g., language keywords)

Removing Stop Words

- Common words in English, programming language keywords

```java
public void run IProgressMonitor monitor throws InvocationTargetException InterruptedException if m_iFlag the processCorpus monitor checkUpdate else if m_iFlag processCorpus monitor UD_UPDATECORPUS else a processQueryString monitor if monitor isCancelled throw new InterruptedException the long running
```
Stemming

- Identify morphological variants, creating “classes”
  - system, systems
  - forget, forgetting, forgetful
  - analyse, analysis, analytical, analysing

- Use in an IR system
  - Replace each term by the class representative (root or most common variant)
  - Replace each word by all the variants in its class

Stemming Errors

- Too aggressive
  - organization / organ
  - police / policy
  - army / arm
  - executive / execute

- Too timid
  - european / europe
  - creation / create
  - searcher / search
  - cylindrical / cylinder
Document Granularity

• What is a document in source code?
  - Depends on the problem and programming language
  - Class, method, function, interface, procedure, etc.

• What is a document in other artifacts?
  - Depends on the artifact and problem
  - Individual requirements, bug descriptions, test cases, e-mails, design diagrams, etc.

SE Tasks as TR Problems

• Need to define the followings:
  - What is the document space?
    • source code, other artifacts, combinations, etc.
  - What are the queries?
    • user generated, other documents, etc.
  - How to evaluate the results?
    • precision, recall, accuracy, f-measure, etc.
  - How to index the documents?
    • IR model
Applications

- Concept location
- Traceability link recovery
- Coupling
- Cohesion
- Bug triage
- Comprehension
- Etc.

Concept location is software as a text retrieval problem
**Concept Assignment Problem**

- “... *discovering human oriented concepts and assigning them to their implementation instances within a program* ...” [Biggerstaff’93]
- Need a well defined context (i.e., developer task)
- When does one concept stop and another one starts?
- Composite concepts

---

**Instances of the Concept Assignment Problem**

- Concept location / bug location
- Traceability link recovery between artifacts
- Concern/aspect mining
- Similar problems in other fields, e.g., in bioinformatics - gene expression
Instantiation Requires

- Context (i.e., problem)
- Input
- Output
- Methodology/process

Concept Location = Find the Point of Change

- Change request
- **Concept location**
- Impact analysis
- Implementation
- Change Propagation
- Testing
Concept Location

- Concept location is needed whenever a change is to be made
- Change requests are most often formulated in terms of domain concepts
  - example: “Correct error that arises when trying to paste a text”
  - the programmer must find in the code the locations where concept “paste” is located
  - this is the start of the change

Concept Location in Practice and Research

- Static
  - Dependency based search
  - Text search (e.g., grep, TR-based)

- Dynamic
  - Execution traces (e.g., Reconnaissance)

- Combined
TR-based Concept Location

1. Creating a corpus from the source code
2. Indexing the corpus with the TR method
   (we used LSI, Lucene, GDS, LDA)
3. User formulates a query
4. Ranking methods
5. User examines the results
6. Go to 3 if needed
Improvements

- Clustering the results
  - Adds structure to the results
- Query reformulation via relevance feedback
  - Developers prefer code than queries
  - They know what they are looking for but can’t describe it
- Combination with static and dynamic analysis

Example: Clustering the Results

Clustered results into labeled categories

- Table
  - createTable
    - Widget.setData
    - FilteredList.TableUpdater
    - ...
    - Table.createWidget
  - tableViewer
  - getTable
  - tableValue, keyTable
- Header
  - setHeaderVisible
  - setLineVisible
  - ...

Ranked List

1. WidgetTable.put
2. TableTree.getTable
3. EditorsView.getTable
4. SimpleLookupTable.rehash
5. WidgetTable.shells
   • ...
39. TableTreeEditor.resize
   • ...
71. Widgets.Table.createWidget
**Example: Relevance Feedback**

JFace Text Editor Leaves a Black Rectangle on Content Assist text insertion. Inserting a selected completion proposal from the context information popup causes a black rectangle to appear on top of the display.

1. `createContextInfoPopup()` in [✔️ org.eclipse.jface.text.contentassist.ContextInformationPopup](#)
2. `configure()` in [❌ org.eclipse.jdt.internal.debug.ui.JDIContentAssistPreference](#)
3. `showContextProposals()` in [✔️ org.eclipse.jface.text.contentassist.ContextInformationPopup](#)

![New Query](#)

**TR and Static Code Analysis**

- Add dependency information in the list of results
- Search results are ranked via IR and explored based on program dependencies
- Programmers switch between dependency navigation and IR based search as needed
- Instance of information seeking activity - searching and browsing
- Dependencies can be ranked
- Cluster the software using the dependencies
Dynamic Feature Location

Software Reconnaissance*

Feature Invoked

Feature Not Invoked

Scenario-based Probabilistic Ranking (SPR)**

Challenges in Dynamic Analysis

- Execution traces of a scenario includes all methods relevant to that scenario
- Precision is a problem as execution traces are very large
- Selecting multiple scenarios is difficult
- Filtering the traces is equally problematic - best filtering methods still return hundreds of methods
**TR + Execution Traces**

- Use a single (partial) scenario
- Use IR to rank the execution trace
- Less sensitive to user query quality
- Improves accuracy over its constituent techniques
**Precision and Recall in CL**

- By definition, recall always = 1 because CL stops when the first relevant document is found (i.e., location of change)
- Precision can be translated into the number of retrieved documents (i.e., examined by the user) - called effectiveness - not perfect!
- This is not the same in related applications since output > 1

**When is a CL Technique Good?**

- CL is a human driven, tool assisted process
- The goal of the tools is to reduce human effort
- Approximate human effort with amount of code has to inspect
- Effectiveness = 1/precision = human effort = number of documents inspected/retrieved
- eff < 10 - excellent; 10 < eff < ~20 - good; 20 < eff < ~50 - acceptable; eff > 50 - poor
Evaluation Methodology

• Case studies with developers
  - Developers receive a change request and perform concept location, assisted by a particular tool we want to evaluate
  - Compare results (i.e., number of inspected documents) with CL without the tool or with other tools

• How do we know CL is successful?
  - Implement and test the change - impractical

Reenactment

• Reenactment of change = perform concept location for existing changes

• Success is achieved when one item in the change set is located

• Allows for automated verification of results -> automation of evaluation
Automated Evaluation

- Mine repositories for past changes
- Match a change request (i.e., bug report or feature request) with patches and find the change set (i.e., methods or classes that changed)
- Use the change request as the starting query
- *Query reformulation not available*

Existing Feature/Concept Location Work

![Diagram showing various tools and techniques for feature/concept location]
Traceability link recovery as a text retrieval problem

CL vs. Traceability Link Recovery

- Both are instances of the concept assignment problem, however
- Different input and output -> different evaluation (recall important)
- Variety of software artifacts
- Code structure and behavior less important than in CL -> dynamic and static analysis not used heavily
- No user (re)formulated query (typically)
- Similar user role: validation and relevance feedback
The Problem

What are the documents associated to a given source code component?

Requirements

Source Code Components

Traceability Definitions - IEEE SE Glossary

- The degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another;
  - the degree to which the requirements and design of a given software component match;

- The degree to which each element in a software development product establishes its reason for existing;
  - the degree to which each element in a graphical environment references the requirement that it satisfies.
Gotel and Finkelstein 1994

- The ability to describe and follow the life of a requirement, in both a forward and backward direction:
  - from its origins, through its development and specification, to its subsequent deployment and use, and through periods of ongoing refinement and iteration in any of these phases.

Why Traceability?

- It is required or suggested by many standards:
  - MIL-STD-498, IEEE/EIA 12207 (Military)
  - ISO/IEC 12207
  - DO178B, DO254 (Avionic)
  - EN50128 (Railways)
- Bottom-up and top-down program comprehension
- Impact analysis
- Forward/Backward requirements traceability and contractual agreements
  - all required functionalities are there
  - there is no EXTRA functionality
- Identification of reusable software components
Traceability Between

- Requirement and code
- Design and code
- Requirement and design
- Requirement and test cases
- Design and test cases
- Bug report and maintainer
- Manual page to code
- ....

The Missing Link

- The basic assumption - the wise developer:
  - Developers use consistent naming conventions to create identifiers, write comments, name artifacts, write manual pages or e-mails. They use domain concepts and knowledge in a uniform and consistent way.
  - If two artifacts are related to the same domain concept, requirement, functionality or knowledge, if one is the refinement of the other, then they will share a set of terms.
- We replace the degree of similarity of two documents with the likelihood of existence of a traceability link between them
Challenges

- High level documents mostly in natural language
  - source code ... well ... acronyms, abbreviations, ....
- We need to process semi-formal documents -- OCL annotations
- Automatic-generated code
- CORBA or other middleware
- COTS
- Reused code

Challenges (2)

- Conceptual distance between different artefacts
  - High level requirement vs code or test cases
- Vocabulary inconsistency
  - fault, defect, bug, issue, ...
- Text sparseness
  - there is no better data than more data
Approaches

• Text retrieval
  - Use as query the source artifact(s) and identify the target artifact(s)

• Improvements
  - Relevance feedback
  - Clustering
  - Document processing
  - Etc.

• Horizontal vs. vertical links

Not all software engineering tasks are text retrieval problems
Relationships in Software

• Structural relationships
  - Coupling
  - Cohesion

• Evolutionary relationships
  - Coupling - co-change

• Semantic relationships?

Conceptual Coupling between Classes

• Method - Class conceptual similarity

• Class - Class conceptual similarity

Conceptual coupling between A and B = 0.4
Maximal Conceptual Coupling

- Conceptual coupling based on the strongest conceptual coupling link

Conceptual coupling between A and B = 0.56

Class A
- method1: 0.7
- method2: 0.6
- method3: 0.4

Class B
- method1: 0.5
- method2: 0.6
- method3: 0.3

Are We Measuring Anything New?

- Compare with other coupling measures:
  - Coupling between classes (CBO) [Chidamber’04]
  - Response for class (RFC) [Chidamber’04]
  - Message passing coupling (MPC) [Li’93]
  - Data abstraction coupling (DAC) [Li’93]
  - Information-flow based coupling (IPC) [Lee’95]
  - A suite of coupling measures by Briand et al: ACAIC, OCAIC, ACMIC and OCMIC
Principal Component Analysis

- Identifying groups of metrics (variables) which measure the same underlying mechanism that defines coupling (dimension)
- PCA procedure:
  - collect data
  - identify outliers
  - perform PCA

PCA Results: Rotated Components

- CoCC and CoCCm define new dimensions ($PC_2$ and $PC_6$)

<table>
<thead>
<tr>
<th>Metric</th>
<th>PC1</th>
<th>PC2</th>
<th>PC3</th>
<th>PC4</th>
<th>PC5</th>
<th>PC6</th>
<th>PC7</th>
<th>PC8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proportion Cumulative</td>
<td>0.19%</td>
<td>9.49%</td>
<td>9.41%</td>
<td>12.19%</td>
<td>19.53%</td>
<td>9.45%</td>
<td>9.05%</td>
<td>9.14%</td>
</tr>
<tr>
<td>PC1</td>
<td>19.00%</td>
<td>28.80%</td>
<td>38.22%</td>
<td>50.41%</td>
<td>69.95%</td>
<td>79.40%</td>
<td>88.46%</td>
<td>97.61%</td>
</tr>
<tr>
<td>CoCC</td>
<td>-0.046</td>
<td>0.941</td>
<td>0.042</td>
<td>0.000</td>
<td>-0.031</td>
<td>0.279</td>
<td>0.129</td>
<td>-0.033</td>
</tr>
<tr>
<td>CoCCm</td>
<td>0.064</td>
<td>0.343</td>
<td>-0.101</td>
<td>0.024</td>
<td>0.115</td>
<td>0.904</td>
<td>0.041</td>
<td>0.074</td>
</tr>
<tr>
<td>CBO</td>
<td>0.260</td>
<td>-0.147</td>
<td>0.185</td>
<td>0.558</td>
<td>0.309</td>
<td>0.341</td>
<td>0.017</td>
<td>0.473</td>
</tr>
<tr>
<td>RFC</td>
<td>0.264</td>
<td>-0.019</td>
<td>0.046</td>
<td>0.266</td>
<td>0.422</td>
<td>0.067</td>
<td>0.075</td>
<td>0.803</td>
</tr>
<tr>
<td>MPC</td>
<td>0.233</td>
<td>-0.017</td>
<td>-0.029</td>
<td>0.154</td>
<td>0.929</td>
<td>0.081</td>
<td>0.024</td>
<td>0.202</td>
</tr>
<tr>
<td>DAC</td>
<td><strong>0.931</strong></td>
<td>-0.027</td>
<td>0.074</td>
<td>0.161</td>
<td>0.268</td>
<td>0.043</td>
<td>0.084</td>
<td>0.136</td>
</tr>
<tr>
<td>ICP</td>
<td>0.346</td>
<td>-0.014</td>
<td>-0.024</td>
<td>0.139</td>
<td><strong>0.903</strong></td>
<td>0.074</td>
<td>0.028</td>
<td>0.162</td>
</tr>
<tr>
<td>ACAIC</td>
<td>0.052</td>
<td>0.035</td>
<td><strong>0.950</strong></td>
<td>0.022</td>
<td>-0.040</td>
<td>-0.081</td>
<td>0.272</td>
<td>0.046</td>
</tr>
<tr>
<td>OCAIC</td>
<td><strong>0.935</strong></td>
<td>-0.026</td>
<td>0.006</td>
<td>0.162</td>
<td>0.274</td>
<td>0.046</td>
<td>0.067</td>
<td>0.127</td>
</tr>
<tr>
<td>ACMIC</td>
<td>0.113</td>
<td>0.129</td>
<td>0.281</td>
<td>0.050</td>
<td>0.040</td>
<td>0.041</td>
<td><strong>0.939</strong></td>
<td>0.049</td>
</tr>
<tr>
<td>OCMIC</td>
<td>0.222</td>
<td>0.029</td>
<td>-0.007</td>
<td><strong>0.928</strong></td>
<td>0.181</td>
<td>-0.018</td>
<td>0.052</td>
<td>0.157</td>
</tr>
</tbody>
</table>
Conceptual Coupling Support for Impact Analysis

- http://www.cs.wayne.edu/~severe/CoCC/Mozilla_coupling_metrics.zip
- 11 coupling metrics including CCBC and CCBCm
- Historical change data on Mozilla - 391 bug reports
- Precision, recall, and F-measure at various cut points
- CCBCm turns out to be the best change indicator

Cohesion in Software

- Cohesion is the degree to which the elements in a design unit (class, package) are logically related or “belong together” [Briand’00]
- A cohesive class represents a crisp abstraction from a problem domain
- Different views of cohesion
- No accepted standard in the community
From Coupling to Cohesion

- Coupling - inter module
- Coupling - system level
- Cohesion = intra module coupling
- Cohesion - module level

- Desirable decomposition properties:
  - High module cohesion
  - Low coupling between modules

From Coherence to Cohesion

- Remember that we are measuring textual similarities
- Coherence in linguistics is what makes a text semantically meaningful.
- Cohesion is the grammatical and lexical relationship within a text or sentence. Cohesion can be defined as the links that hold a text together and give it meaning. It is related to the broader concept of coherence
Measuring Class Cohesion

• Structural metrics:
  - LCOM1, LCOM2 [Chidamber 94]; LCOM3, LCOM4 [Hitz 94]
  - LCOM5 [Henderson 96]
  - Connectivity [Hitz 94]; Coh [Briand 97, 98]
  - ICH² [Lee 95]; TCC³, LCC⁴ [Bielean 95, 98]

• Semantic metrics
  - LORM⁵ [Etzkorn 00]; SCDE⁶ [Etzkorn 02]; SCF⁷ [Maletic 01]

• Information entropy-based metrics; Metrics based on data mining; Slice-based metrics; etc.

1. Lack of cohesion in methods
2. Information-flow based cohesion
3. Tight class cohesion
4. Loose class cohesion
5. Logical relatedness of methods
6. Semantic class definition entropy
7. Semantic cohesion of files

Types of Cohesion

• Functional
• Informational
• Communicational
• Procedural
• Temporal
• Logical
• Coincidental
The Conceptual Cohesion of Classes

- Average conceptual similarity of the methods in a class (ACSM) \( c \in C \)
  \[
  \text{ACSM}(c) = \frac{1}{N} \times \sum_{i=1}^{N} \text{CSM}(m_i, m_j)
  \]
- Conceptual cohesion of a class (C3) \( c \in C \)
  
  \[
  \text{C3}(c) = \begin{cases} \text{ACSM}(c) & \text{if} \quad \text{ACSM}(c) > 0 \\ \text{else} & 0 \end{cases}
  \]

Shortcomings of C3

- Are two classes with the same C3 value equally cohesive? (SD of the CSM values)
- Measure the influence of highly related methods in a class with a low C3 cohesion
- Define a new measure based on the counting mechanism utilized in LCOM2
  - Do not take into account intersections of methods based on common attribute usage
  - Count intersections of method pairs based on the CSM value between them and the ACSM
Lack of Conceptual Similarity between Methods (LCSM)

- Let $M_i = \{m_j \mid (m_i, m_j) \in E, m_i \neq m_j\}$ be the set of neighbor methods of $m_i$ (with which $m_i$ has a higher CSM value than the average).
- Let $P = \{(M_i, M_j) \mid M_i \cap M_j = \emptyset\}$
- Let $Q = \{(M_i, M_j) \mid M_i \cap M_j \neq \emptyset\}$
- Lack of conceptual similarity is

$$\text{LCSM}(c) = \begin{cases} |P| - |Q| & \text{if } |P| > |Q| \\ \text{else} & 0 \end{cases}$$

Limitations

- C3 and LCSM do not take into account polymorphism and inheritance
- Method invocation, parameters, attribute references, and types are of interest only at identifier level
- C3 and LCSM do not make distinction between constructors, accessors, and other method stereotypes. Some of these methods can artificially increase or decrease cohesion
Are We Measuring Something New?

• Compare C3, C3’, LCSM, and LCSM’ with \([\text{LCOM}_1 - \text{LCOM}_5]\), Coh, C, ICH, TCC, and LCC

• WinMerge with 51KLOC and 11K comments
• Metrics computed for 34 classes with 522 methods
• Structural metrics computed with Columbus [Ferenc’04], C3 and LCSM - our tool

• Analysis of correlations between metrics

Results

• C3 and C3’ very close values (WinMerge has 20% of code as comments)
• LCSM and LCSM’ are less conclusive in this respect, but the differences are still not major
• C3 and LCSM do not correlate - interesting!
• Significant correlations between C3 and ICH, and C3 and LCOM5 - not major surprise
• No significant correlation between any structural metric and LCSM - somewhat surprising! - expected LCOM2 to correlate
Metrics are Complementary

- Structural metrics tell us if a class is built cohesively
- Semantic/conceptual metrics tell us if a class is written cohesively
- We desire both -> increase maintainability

So What?

- The metrics are different, but are they better?
- Are they better fault predictors?
- Performed a case study on Mozilla
- C3+LCOM3, C3+LCOM1, and C3+Coh turn out to be the best predictors, better than any single metric
Now What?

- Many possible applications
  - Refactoring
  - Remodularization
  - ADT identification
  - Clone detection
  - Predictor models
  - Etc.

Other software engineering tasks commonly solved using text retrieval
Bug Triage

• Incoming bug reports need to be verified, assigned a severity, a developer, etc.
-> bug triage

• The triage often starts by analyzing the natural text contained in the bug report title and description
  - opportunity to automate some of these tasks by using text mining techniques

Tasks and Solutions

• Sub-problems:
  - duplicate bug detection
  - developer recommendation
  - automatic assignment of severity
  - automatic detection of security bugs

• Text mining algorithms used:
  - classification, clustering, text matching, TR, etc.
Duplicate Bug Detection

New bug report
m+1
Query
similarity betw query
and indexed bug
List of potential duplicates
1. i
2. j
...

Developer Recommendation

New bug report
Firefox crashes when I open...
Query
similarity betw query
and source code units
List of textually similar code units
1. i modified by developer Y
2. j modified by developer X, Y
...
Assign new bug report to Y
Assigning Bug Severity

- Using TR, build an index from a set of bugs which have the level of severity assigned
- Extract for each bug the most representative terms
- Build a model which associates a severity level with the set of most representative terms found in bug descriptions having that level of severity
- For incoming bugs, use the model and the terms in the bug description to automatically assign a severity level to the bug

Detection of Security Bugs

- Using TR, build an index from a set of bugs, each labeled as a security bug (SB) or non-security bug (NSB)
- Train a model which refines the index until the recall and precision of the classification of bugs in SB or NSB is satisfactory
- The model assigns to each bug report a probability of belonging to SB and NSB
- The probability of a bug report to be a SB increases if the description of the bug contains keywords like “buffer overflow”, “crash”, “buffer overrun”, etc.
- For incoming bugs, determine the most probable category for the incoming bug and assign the new bug to that category
**Topic Maps in Code**

- TR is used to compute the linguistic similarity between source artifacts (e.g., packages, classes or methods).
- The artifacts are clustered according to their similarity, which partitions the system into linguistic topics that represent groups of documents using similar vocabulary.
- TR is used again to automatically label the clusters with their most relevant terms (determined using the TR technique itself).

**Obtaining the Topic Clusters**

Diagram showing the process of obtaining topic clusters.
Example: Topic Maps in JEdit


Conclusions

• Many successful applications of TR in SE

• The field matures, but there are many open questions

• There is a need for benchmarks and open data
About the Lab

- Clustering of bug descriptions using VSM
- Clustering of bug descriptions using LSI
- Compare clusters
- Searching the corpus
- Term-term, document-document similarities, most relevant terms
- Topics in bug descriptions using LDA
- Compare clusters with topics

Software clustering
(if needed)
Software Clustering

- Used to group software entities in clusters such that:
  - the entities in one cluster are similar to each other
  - entities in different clusters are dissimilar
- Goal: Determine the intrinsic grouping in a set of unlabeled data

Software Clustering - Uses

- Software architecture recovery
- Identifying the topics implemented
- Determining the scattering and tangling of aspects in code
- Detect software clones
- Software remodularization
- Program comprehension
- Traceability link recovery
Taxonomy of Clustering Approaches

Hierarchical Clustering

Agglomerative clustering treats each data point as a singleton cluster, and then successively merges clusters until all points have been merged into a single remaining cluster. Divisive clustering works the other way around.
Agglomerative Clustering

In single-link hierarchical clustering, we merge in each step the two clusters whose two closest members have the smallest distance.

Agglomerative Clustering

In complete-link hierarchical clustering, we merge in each step the two clusters whose merging has the smallest diameter.
K-Means

- Step 0: Start with a random partition into $K$ clusters
- Step 1: Generate a new partition by assigning each pattern to its closest cluster center
- Step 2: Compute new cluster centers as the centroids of the clusters.
- Step 3: Steps 1 and 2 are repeated until there is no change in the membership (also cluster centers remain the same)

Comparing Clusterings - Rand Index

- The *Rand index* is a measure of the similarity between two data clusterings.
- Given a set of $n$ elements $S = \{O_1, \ldots, O_n\}$ and two partitions of $S$ to compare, $X = \{x_1, \ldots, x_r\}$ and $Y = \{y_1, \ldots, y_s\}$, the following is defined:
  - $a$, the number of pairs of elements in $S$ that are in the same set in $X$ and in the same set in $Y$
  - $b$, the number of pairs of elements in $S$ that are in different sets in $X$ and in different sets in $Y$
  - $c$, the number of pairs of elements in $S$ that are in the same set in $X$ and in different sets in $Y$
  - $d$, the number of pairs of elements in $S$ that are in different sets in $X$ and in the same set in $Y$
- The Rand index, $R$, is:
  $$R = \frac{a + b}{a + b + c + d} = \frac{a + b}{\binom{n}{2}}$$
Comparing Clusterings - Rand Index (2)

- $a + b$ can be considered as the number of agreements between $X$ and $Y$.
- $c + d$ can be considered as the number of disagreements between $X$ and $Y$.
- The Rand index has a value between 0 and 1, with 0 indicating that the two data clusters do not agree on any pair of points and 1 indicating that the data clusters are exactly the same.
- The adjusted Rand index is the corrected-for-chance version of the Rand index.

Authoritativeness (Auth)

- Regards the resemblance between the software clusters identified by a clustering approach and an authoritative partition given by an expert.
- The clusters produced by the approach should resemble as much as possible the groups of entities within an authoritative partition.
- MoJo distance-based measures can be used to compute Auth.
- Let be $A$ a source partition and $B$ an authoritative partition, $\text{MoJo}(A, B)$ is defined as the minimum number of join and move operations to turn $A$ into $B$.
- The lower is the MoJo distance, the more the identified clusters resemble groups of entities within the authoritative partition.
Acknowledgments

• People that I am grateful to have worked with on topics mentioned in the tutorial (30):

- Denys Poshyvanyk, Sonia Haiduc, Scott Ohlemacher, Andrey Sergeyev, Jairo Aponte, Laura Moreno, Xinrong Xie, Yubo Dong, Jonathan Maletic, Vaclav Rajlich, Giuliano Antoniol, Massimiliano Di Penta, Yann Gael-Gueheneuc, Rocco Oliveto, Andrea De Lucia, Gabriele Bavota, Tim Menzies, Greg Gay, Paolo Tonella, Surafel Abebe, Giuseppe Scanniello, Tibor Gyimothy, Rudolf Ferenc, Sergio Di Martino, Anna Corazza, Valerio Maggio, Dapeng Liu, Maksym Petrenko, Malcom Gethers, Bogdan Dit