The Use of Text Retrieval and Natural Language Processing in Software Engineering

Tutorial @ SaTToSE 2016

WASHINGTON STATE UNIVERSITY

Dr. Venera Arnaoudova

Assistant Professor

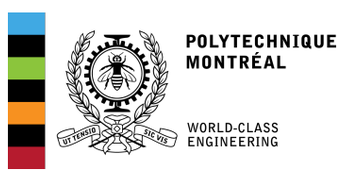
The Use of Text Retrieval and Natural Language Processing in Software Engineering

Sonia Haiduc

Venera Arnaoudova

Andrian Marcus

Giuliano Antoniol



Outline

- Introduction
- Background on IR and NLP
- SE tasks using IR and NLP
  - Task definition
  - Input
  - Output
  - Preprocessing
  - Techniques
  - Evaluation
  - Tools used
- Hands-on

Textual Information in Software

- Captures concepts of the problem domain, developer intentions, developer communication, etc.

- Found in many software artifacts:
  - Requirements
  - Design documents
  - Source code (identifiers, comments)
  - Commit notes
  - Documentation
  - User manuals
  - Q/A websites
  - Developer communication: emails, chat, tweets
  - Etc.

Text Retrieval

- *Information Retrieval (IR)*: the process of actively seeking out information relevant to a topic of interest
  (van Rijsbergen)

- **Text Retrieval (TR):** a branch of IR where the information is stored in text format
  - Search space: collection of documents (*corpus*)
  - *Document* - generic term for an information unit
    - book, chapter, article, webpage, etc.
    - class, method, interface, etc.
    - individual requirement, bug description, test case, e-mail, design diagram, etc.

Natural Language Processing

- Refers to the use and ability of systems to process sentences in a natural language such as English (rather than in a specialized, artificial computer language such as C++)

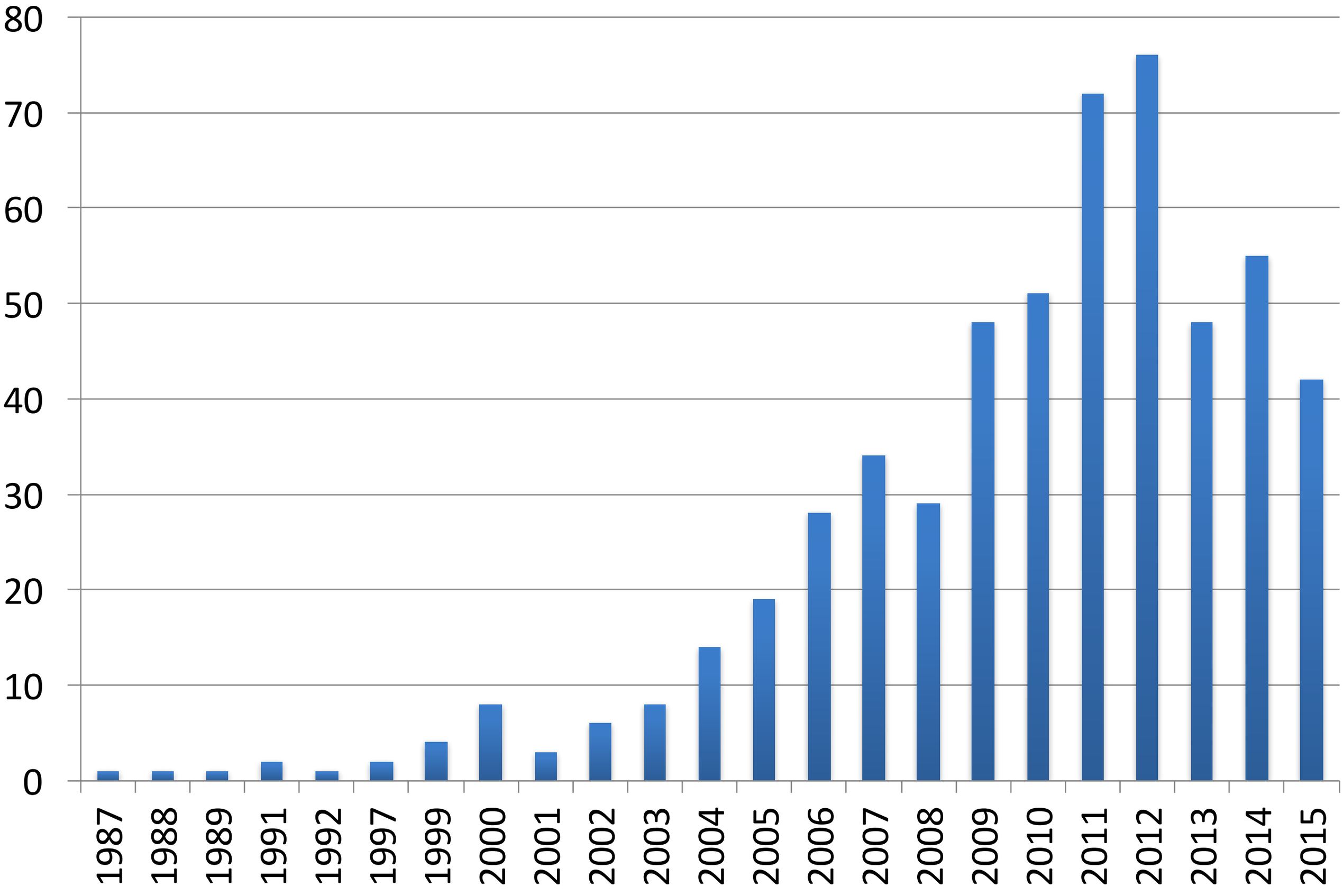
- Combines techniques from computer science, artificial intelligence, computational linguistics, probability and statistics

TR and NLP in Software Engineering

- Applied to over 30 different SE tasks

- Traceability Link Recovery
- Feature/concept/concern/bug location
- Code reuse
- Bug triage
- Program comprehension
- Architecture/design recovery
- Quality assessment and measurement
- Software evolution analysis
- Defect prediction and debugging
- Automatic documentation
- Testing
- Requirements analysis
- Restructuring/refactoring
- Software categorization
- Licensing analysis
- Impact analysis
- Clone detection
- Effort prediction/estimation
- Domain analysis
- Web services discovery
- Use case analysis
- Team management, etc.

Publications per year



Outline

- Introduction
- Background on IR and NLP
- SE tasks using IR and NLP
  - Task definition
  - Input
  - Output
  - Preprocessing
  - Techniques
  - Evaluation
  - Tools used
- Hands-on

A Typical TR Application

1. Build corpus
2. Index corpus – choose the *TR model*
3. Formulate a *query*
   - Manual or automatic
4. Compute *similarities* between the query and the documents in the corpus (i.e., relevance)
5. *Rank* the documents based on the similarities
6. Return the top N documents as the *result list*
7. Inspect the results
8. GO TO 3. if needed or STOP

Creating a Corpus of a Software System

- Parsing software artifacts and extracting documents
  - ***corpus*** – collection of documents (e.g., methods)
- Text normalization (white space and non-textual tokens removal, tokenization)
- Splitting: split_identifiers, SplitIdentifiers, etc.
- Stop words removal
  - common words in English, programming language keywords, project-specific words, etc.
- Abbreviation and acronym expansion
- Stemming

Extracting Documents

- Documents can be at different granularities (e.g., methods, classes, files, emails, bug descriptions, etc.)

```
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
}

  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
  }

    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
    }
```

```
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
}

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
}

  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
  }
```

```
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


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
}
  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
  }
```

Extracting Documents

- Documents can be at different granularities (e.g., methods, classes, files, emails, bug descriptions, etc.)

Transform Source Code to Plain Text

```
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
}
```

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

Text Normalization

- Break up the text in “tokens”

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

Splitting

- Splitting: decomposing identifiers into their compound words
- Identifiers may use division markers (e.g., camelCase and _)

- Examples:
  - `getName` -> ‘get’, ‘Name’
  - `getMAXstring` -> ‘get’, ‘MAX’, ‘string’
  - `ASTNode` -> ‘AST’, ‘Node’
  - `account_number` -> ‘account’, ‘number’
  - `simpletypename` -> ‘simple’, ‘type’, ‘name’

Stop Words

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

- Typically function words, not indicative of content

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

Abbreviation and Acronym Expansion

- Expand abbreviations and acronyms to the corresponding full words

- Examples:
  - `mess` -> ‘message’
  - `src` -> ‘source’
  - `regex` -> ‘regular expression’
  - `ASCII` -> ‘American Standard Code for Information Interchange’
  - `auth` -> ‘authenticate’ OR ‘author’

Stemming

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

- Replace each term by the class representative (root or most common variant)

Most Popular TR Models Used in SE

- Vector Space Model (VSM)
- Latent Semantic Indexing (LSI)
- Latent Dirichlet Allocation (LDA)
- Etc.

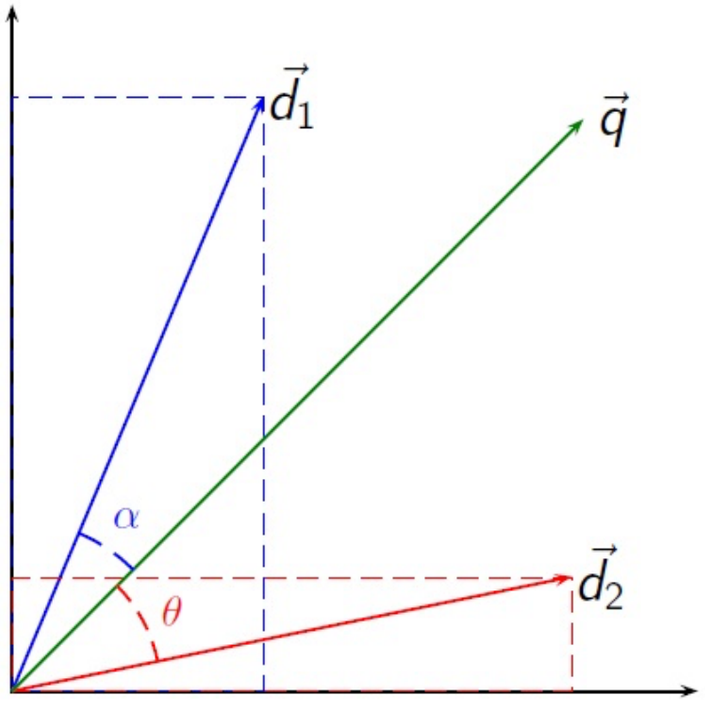
The Vector-Space Model

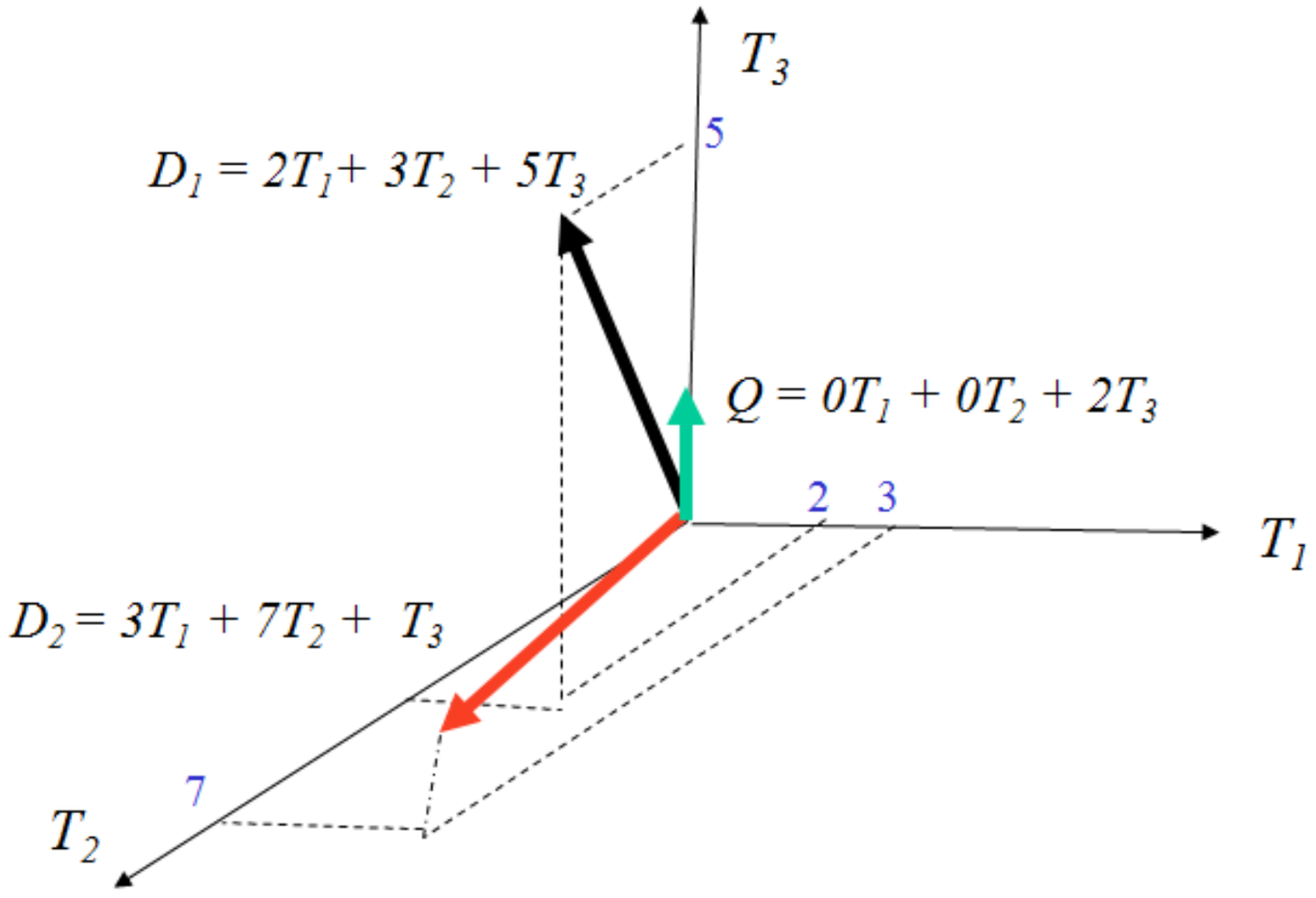
- Assume *t* distinct terms remain after preprocessing
  - call them ***index terms*** or the ***vocabulary***
- These "orthogonal" terms form a vector space.
  - Dimension = *t* = |vocabulary|
- Each term, *i*, in a document or query, *j*, is given a real-valued ***weight***, $w_{ij.}$
- Both *documents* and *queries* are expressed as ***t-dimensional vectors*** (most vectors are sparse):

$$d_j = (w_{1j}, w_{2j}, \ldots, w_{tj})$$

- Each vector holds a place for every term in the collection

Query and Document Vectors





Document-Term Matrix

- A collection of *n* documents can be represented in the VSM by a document-term 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.

$$\begin{array}{c|cccc} & T_1 & T_2 & \ldots & T_t \\ D_1 & w_{11} & w_{21} & \ldots & w_{t1} \\ D_2 & w_{12} & w_{22} & \ldots & w_{t2} \\ : & : & : & & : \\ : & : & : & & : \\ D_n & w_{1n} & w_{2n} & \ldots & w_{tn} \end{array}$$

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}) $w_{ik} = l_{ik} * g_{ik}$
- *Local weights* (l_{ik}) used to indicate the importance of a term relative to a particular document:
  - *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}) is used to indicate the importance of a term relative to the entire document collection. Used as an indication of a term's *discrimination* power.
  - *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; IDF provides high values for rare words and low values for common words

Query-Document Similarity

- A ***vector similarity measure*** between the query and documents is used to rank retrieved documents

Cosine similarity

$$\frac{\sum_i x_i \; y_i}{\sqrt{\sum_i x_i^2 \quad \sum_i y_i^2}}$$

Identical meaning: value of cosine = 1
Unrelated meaning: value of cosine = 0

Two Problems with VSM

- VSM uses *exact word matching*
  - If the exact terms from a query are not found in a relevant document, the document will not be retrieved

- The same concept can be expressed using different sets of terms (*synonyms*)
  - e.g. *bandit*, *brigand*, *thief*

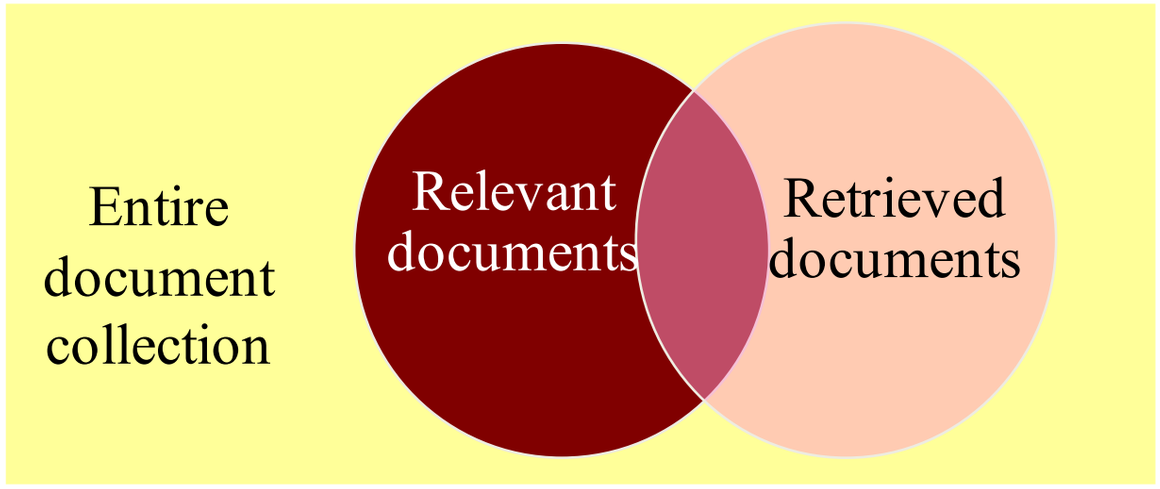
- Identical terms can be used in very different semantic contexts (*homonyms*)
  - e.g. *bank*
    - repository where important material is saved
    - the slope beside a body of water

Latent Semantic Indexing

- Uses linear algebra technique called ***Singular Value Decomposition*** (***SVD***) to simulate human learning of word and passage meaning
  - attempts to estimate the hidden structure
  - discovers the most important associative patterns between words and concepts
  - reduces the document-term matrix to a smaller dimension

- Its success depends on choosing the right number of dimensions to extract

Evaluating TR Systems

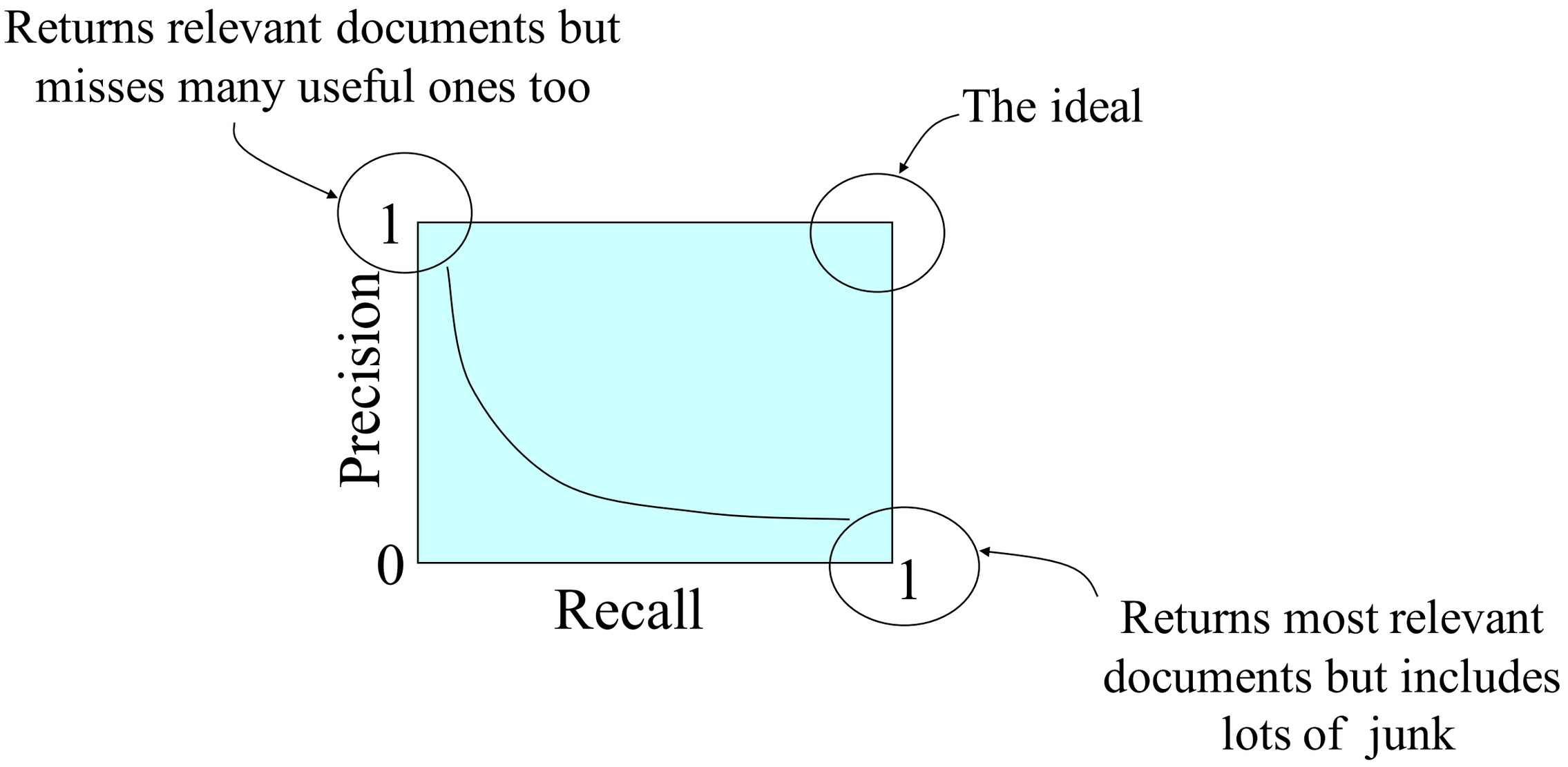


	retrieved	not retrieved
irrelevant	retrieved & irrelevant	not retrieved & irrelevant
relevant	retrieved & relevant	not retrieved but relevant

$$recall = \frac{Number\ of\ relevant\ documents\ retrieved}{Total\ number\ of\ relevant\ documents}$$

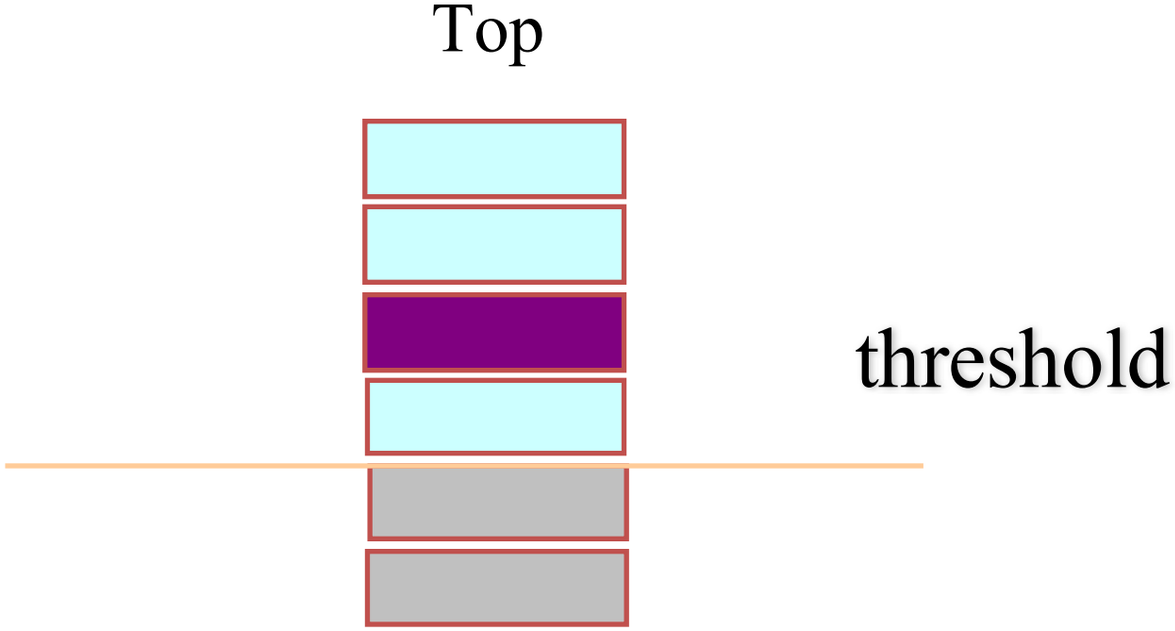
$$precision = \frac{Number\ of\ relevant\ documents\ retrieved}{Total\ number\ of\ documents\ retrieved}$$

Trade-off Between Recall and Precision



Computing Recall/Precision Points

- For a given query, produce the ranked list of documents.
- Any threshold on this ranked list produces different sets of retrieved documents.



- 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

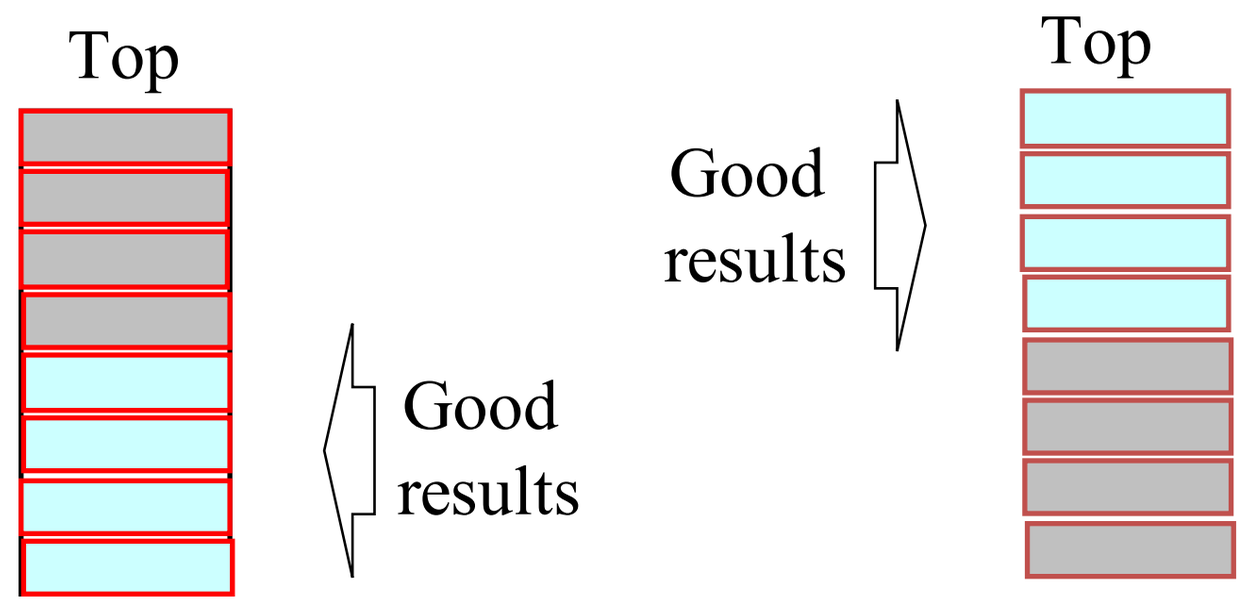
F-Measure

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

$$F = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}.$$

Precision and Recall: the Holy Grail

- Precision and recall do not tell the entire story



- Average precision: $$\text{AveP} = \frac{\sum_{k=1}^{n}(P(k) \times \text{rel}(k))}{\text{number of relevant documents}}$$

Query Formulation

- Usually simple bag of words
  - Ex. *"tutorial software engineering text retrieval"*
- Natural language sentences or paragraphs
  - Ex. "*When is scheduled the tutorial on Text Retrieval Approaches in Software Engineering?*"
- Existing documents

Query Analysis and Expansion

- Spellchecking -> change words
- Compare with vocabulary -> remove words
- Use thesaurus -> suggest alternative words (synonyms – from dictionary, source code, etc.)

Query Modification

- Problem: How can we reformulate the query to help a user?
  - ***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
  - More advanced: automatic based on query properties, mining terms from source code, etc.

Using TR in SE – Option 1

- Formulate the SE problem as a text retrieval problem

- Convert the software artifacts into a text corpus

- Choose the TR model best suited to the problem

SE as TR

- Concept/concern/feature location in software
- Traceability link recovery between software artifacts
- Impact analysis
- Software reuse
- Bug triage
- Requirements analysis
- Etc.

Using TR in SE – Option 2

1. Analysis of the textual information in software

1. Convert the software artifacts into a text corpus

2. Choose the TR model best suited to the problem

3. Compute similarities between documents

4. Perform analysis based on these measures

SE as Text Analysis

- Software categorization
- Refactoring and restructuring
- Reverse engineering
- Bug triage
- Clone detection
- Requirements analysis
- Defect prediction
- Change impact analysis
- Etc.

Advantages of Using TR

- No predefined grammar and vocabulary
- Some techniques able to infer word relationships without a thesaurus or an ontology
- Goes beyond exact string matching
- Offers a ***ranked*** list of results

Limitations of Using IR

- Many TR models are not intuitive for humans
  -> hard to understand the results of TR

- Parameter Configuration

- Ignores document and sentence structure, and relationships between consequent words are ignored (“bag of words”)

Outline

- Introduction
- Background on IR and NLP
- SE tasks using IR and NLP
  - Task definition
  - Input
  - Output
  - Preprocessing
  - Techniques
  - Evaluation
  - Tools used
- Hands-on

Sequences of words

Text is not only a bag of words…

- The order of words matter!

{‘a’, ’chasing’, ‘cat’, ’ fish’, ’is’, ’the’}

“a **cat** is chasing the **fish**” ≠ “a **fish** is chasing the **cat**”

NLP Techniques

- Language Models (LM)

Language Models (LM)

- Assign probabilities for sequences of words

P(happy|“I am”)?

uni-gram: ~ P(happy)

bi-gram: ~ P(happy|am)

tri-gram: ~ P(happy|I am)

…

n-gram

…. happy

…. am happy

…. I am happy

Language Models (LM)

- In general:

$$P(w_1,w_2,w_3,\ldots,w_n) =$$

$$P(w_1)P(w_2|w_1)P(w_3|w_1,w_2)\ldots P(w_n|w_1,\ldots,w_{n-1})$$

$$P(w_1w_2w_3\ldots w_n)=\prod_i (w_i|w_1w_2w_3\ldots w_{i-1})$$

$$P(\text{I am smiling}) =$$

$$P(\text{I})\times P(\text{am}|\text{I})\times P(\text{smiling}|\text{I am})$$

Data sparsity

- Corpus:
    - “I am smiling”
    - “You are happy”
    - “We are happy”
- P(happy|I am) if we use a tri-gram language model?

Grammar

Text is not only a bag of words…

- The order of words matter!

{‘a’, ’chasing’, ‘cat’, ’ fish’, ’is’, ’the’}

“a **cat** is chasing the **fish**” ≠ “a **fish** is chasing the **cat**”

Noun Noun Noun Noun

Subject **Object** **Subject** **Object**

NLP Techniques

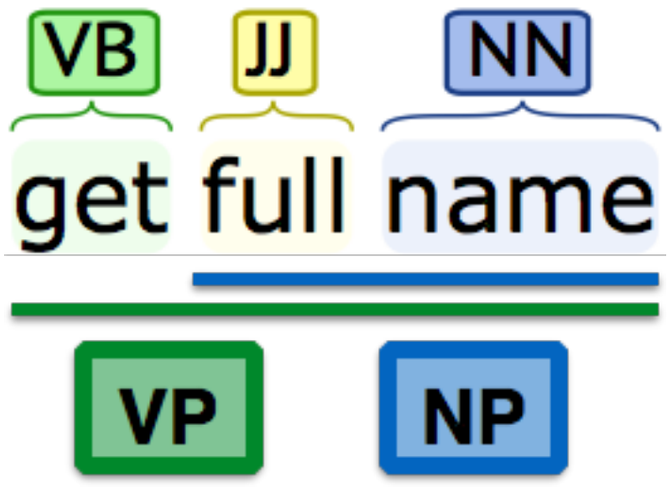
- Language Models (LM)
- Syntactic analysis

Tagging words and phrases

- Tagging words with their respective Part-Of-Speech (POS)

VB: verb
JJ: adjective
NN: noun

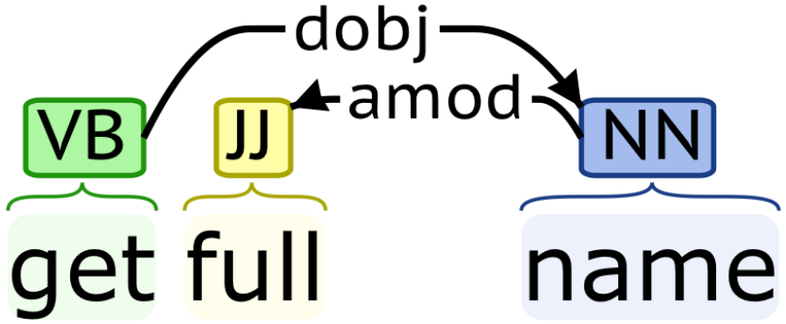
- Chunking



NP: noun phrase
VP: verb phrase

Syntactic Analysis

- Identifying grammatical relations between words



amod: adjectival modifier

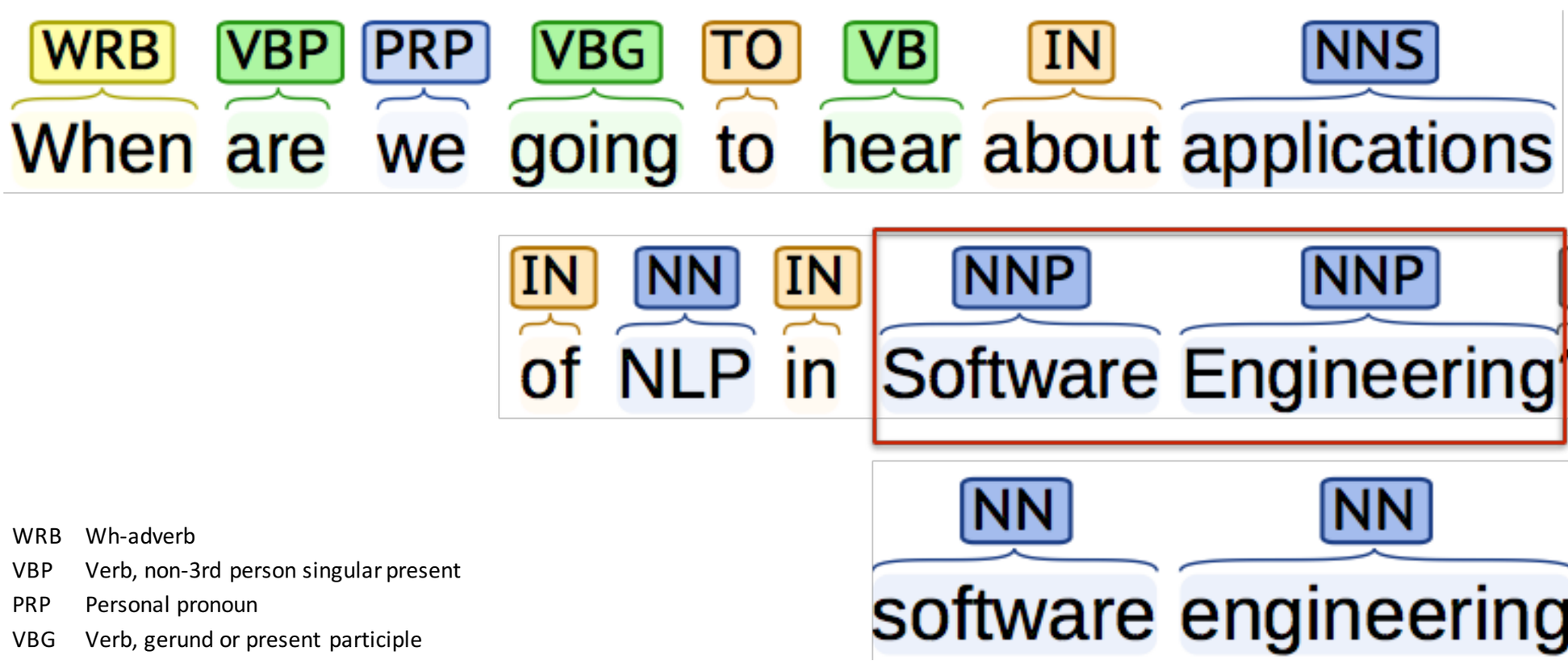
dobj: direct object

Stanford NLP Software

- Stanford CoreNLP
  - Stanford Tokenizer
  - Stanford POS Tagger
  - Stanford Parser
  - …
- …

Stanford POS Tagger

"When are we going to hear about applications of NLP in Software Engineering?"



WRB	Wh-adverb
VBP	Verb, non-3rd person singular present
PRP	Personal pronoun
VBG	Verb, gerund or present participle
VB	Verb, base form
IN	Preposition or subordinating conjunction
NNP	Proper noun, singular

POS Taggers challenge

- What is the POS of the word **back**?
  - Adjective (JJ): “The back door.”
  - Noun (NN): “On my back.”
  - Adverb (RB): “Win the voters back.”
  - Verb (VB): “Promised to back the bill.”

What do POS Taggers rely on?

– Neighbouring words

I play badminton

Verb, non-3rd person singular present (VBP)

– Word probabilities in general

man | man |

noun (pl. **men** | men |)

1 an adult human male.

...

More often used (NN)

verb (**mans**, **manning**, **manned**) [with obj.]

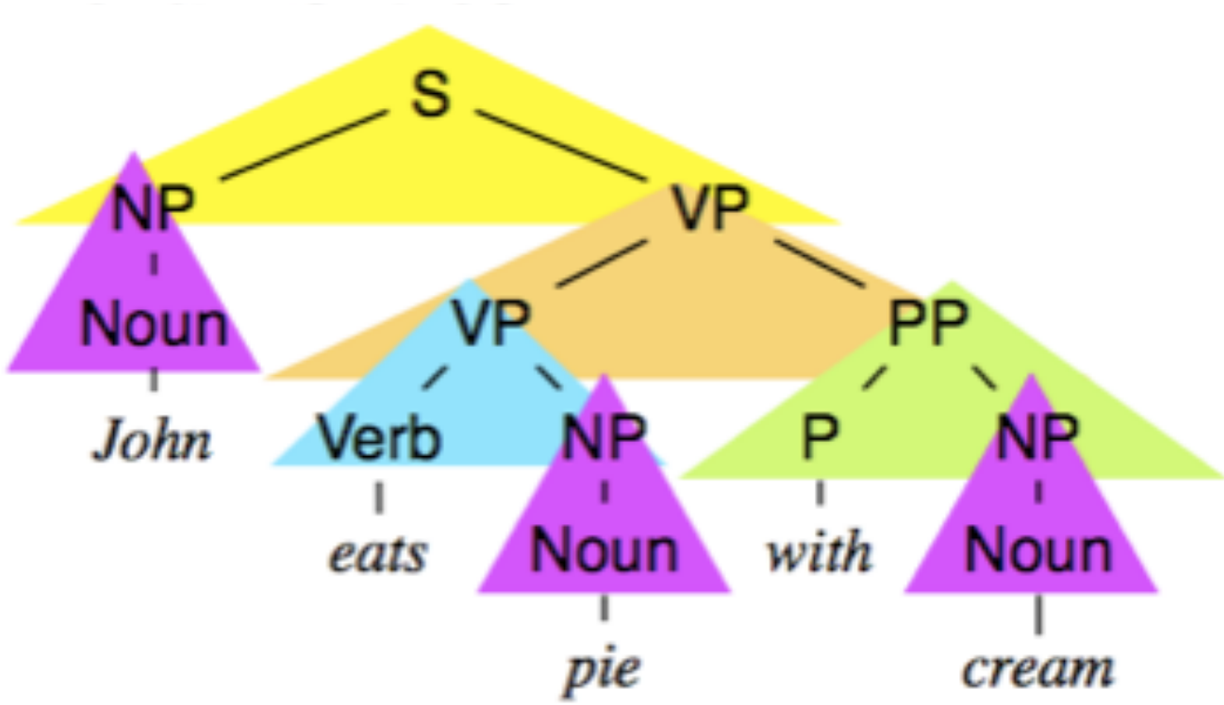
1 (of personnel) work at, run

Parse trees

- “Bell, based in Los Angeles, makes and distributes electronic, computer and building products.”

```
(ROOT
  (S
    (NP
      (NP (NNP Bell))
      (, ,)
    (VP (VBN based)
      (PP (IN in)
        (NP (NNP Los) (NNP Angeles))))
    (, ,))
  (VP (VBZ makes)
    (CC and)
    (VBZ distributes)
    (NP
      (UCP (JJ electronic) (, ,) (NN computer)
        (CC and)
        (NN building))
      (NNS products)))
  (. .)))
```

Probabilistic Context-Free Grammar (PCFG) Parsers

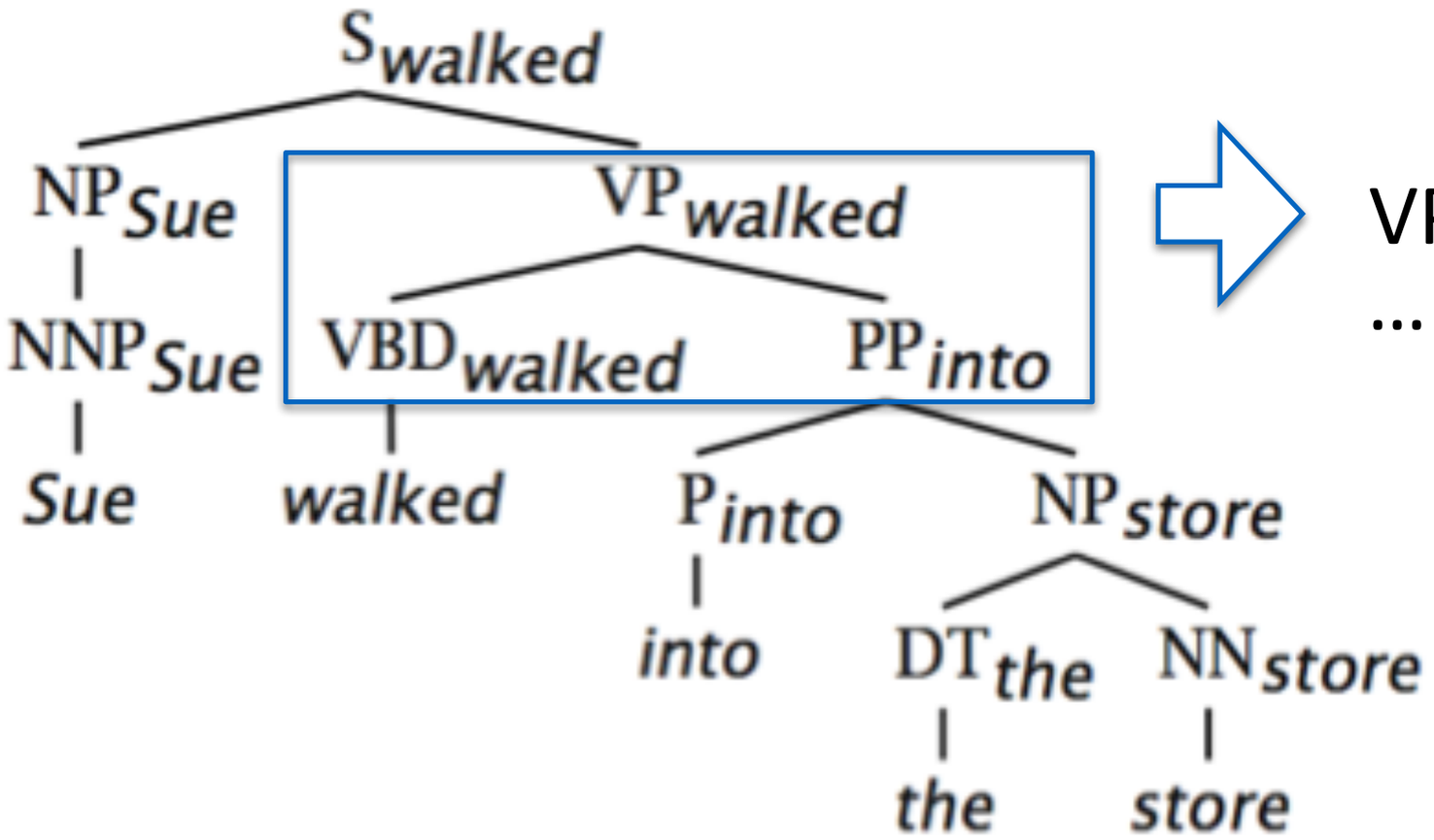


Rule			P
S	→	NP VP	0.8
S	→	S conj S	0.2
NP	→	Noun	0.2
NP	→	Det Noun	0.4
NP	→	NP PP	0.2
NP	→	NP conj NP	0.2
VP	→	Verb	0.4
VP	→	Verb NP	0.3
VP	→	Verb NP NP	0.1
VP	→	VP PP	0.2
PP	→	P NP	1.0

$$P(\tau) = 0.8 \times 0.3 \times 0.2 \times 1.0 \times 0.2^3$$

$$= \mathbf{0.00384}$$

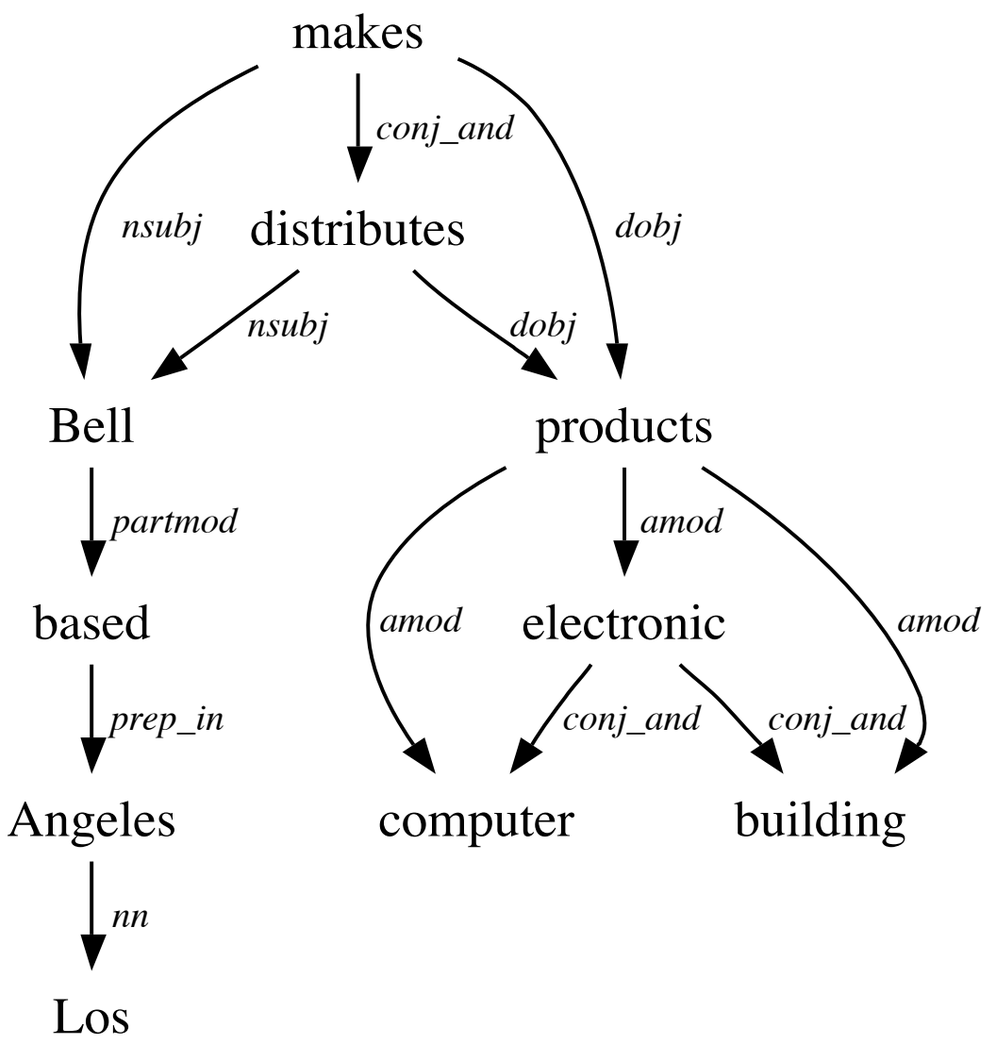
Lexicalized PCFG Parsers



Rule	P
$VP_{walked} \rightarrow VBD_{walked}\ PP_{int\ o}$	0.5
…	

Dependencies

- "Bell, based in Los Angeles, makes and distributes electronic, computer and building products."



nsubj(makes-8, Bell-1)
nsubj(distributes-10, Bell-1)
acl(Bell-1, based-3)
case(Angeles-6, in-4)
compound(Angeles-6, Los-5)
nmod:in(based-3, Angeles-6)
root(ROOT-0, makes-8)
cc(makes-8, and-9)
conj:and(makes-8, distributes-10)
amod(products-16, electronic-11)
conj:and(electronic-11, computer-13)
amod(products-16, computer-13)
cc(electronic-11, and-14)
conj:and(electronic-11, building-15)
amod(products-16, building-15)
dobj(makes-8, products-16)

Semantics

Text is not only a bag of words…

- The order of words matter!

{‘a’, ’chasing’, ‘cat’, ’ fish’, ’is’, ’the’}

“a **cat** is chasing the **fish**” ≠ “a **fish** is chasing the **cat**”

cat [1] | kat |

noun

1 a small domesticated carnivorous mammal with soft fur, a short snout, and retractile claws. It is widely kept as a pet or for catching mice, and many breeds have been developed.

fish [1] | fiSH |

noun (pl. **same** or **fishes**)

a limbless cold-blooded vertebrate animal with gills and fins and living wholly in water: *the sea is thick with fish.*

Semantics

Text is not only a bag of words…

- The order of words matter!

{‘a’, ’chasing’, ‘cat’, ’ fish’, ’is’, ’the’}

“a **cat** is chasing the **fish**” ≠ “a **fish** is chasing the **cat**”

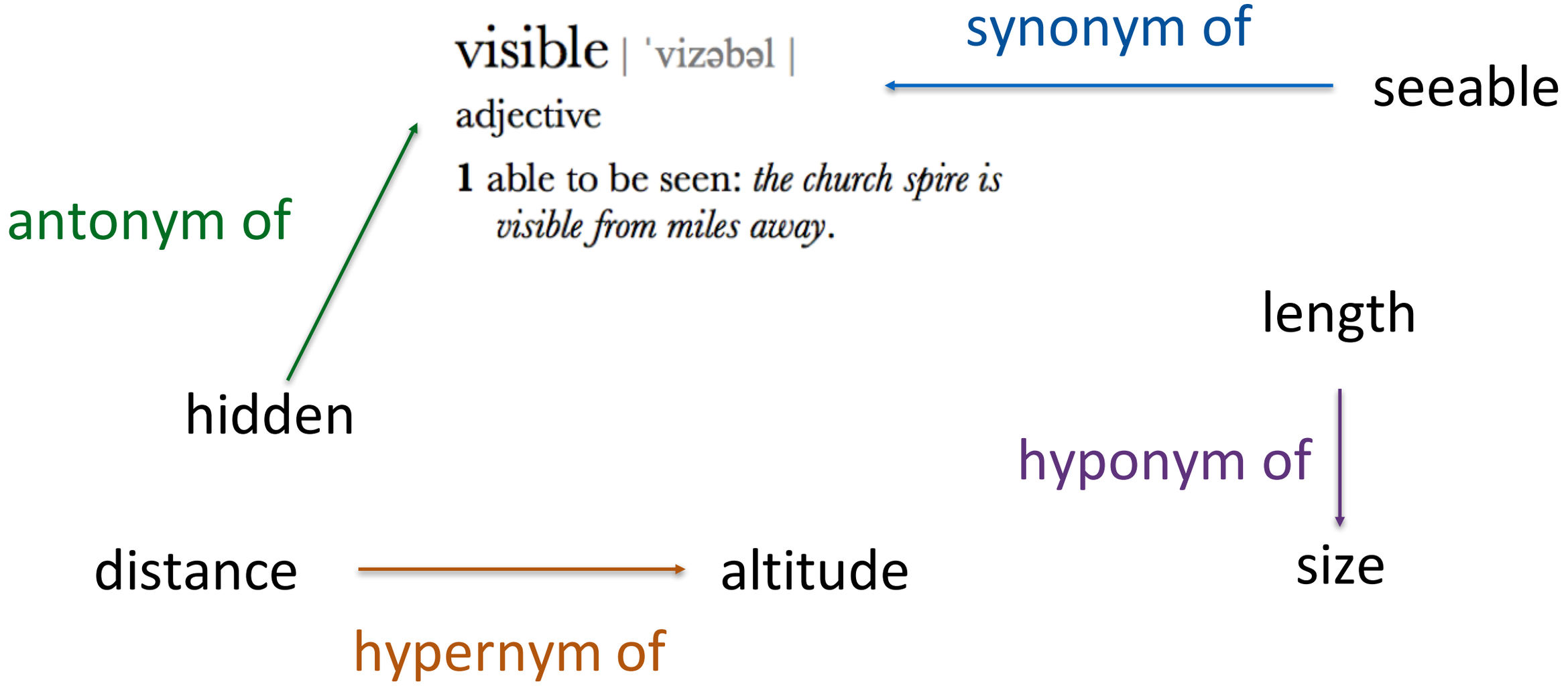
Animal

NLP Techniques

- Language Models (LM)
- Syntactic analysis
- Semantic analysis

Semantic Analysis

- The meaning of words and relations between words



WordNet

- Hierarchically organized lexical database
- **Synset**: a set of synonyms that can be used interchangeably in a particular context
- Synsets are related through pointers
- Pointer may represent a **lexical** or a **semantic relation**

WordNet - meaning of words

- Fish

Noun

- S: (n) **fish** (any of various mostly cold-blooded aquatic vertebrates usually having scales and breathing through gills) *"the shark is a large fish"; "in the living room there was a tank of colorful fish"*
- S: (n) **fish** (the flesh of fish used as food) *"in Japan most fish is eaten raw"; "after the scare about foot-and-mouth disease a lot of people started eating fish instead of meat"; "they have a chef who specializes in fish"*
- S: (n) Pisces, **Fish** ((astrology) a person who is born while the sun is in Pisces)
- S: (n) Pisces, Pisces the Fishes, **Fish** (the twelfth sign of the zodiac; the sun is in this sign from about February 19 to March 20)

Verb

- S: (v) **fish**, angle (seek indirectly) *"fish for compliments"*
- S: (v) **fish** (catch or try to catch fish or shellfish) *"I like to go fishing on weekends"*

WordNet - relations between words

- Fish

- S: (n) **fish** (any of various mostly cold-blooded aquatic vertebrates usually having scales and breathing through gills) *"the shark is a large fish"; "in the living room there was a tank of colorful fish"*
  - *direct hyponym* / *full hyponym*
  - *part meronym*
  - *member holonym*
  - *direct hypernym* / ***inherited hypernym*** / *sister term*
    - S: (n) aquatic vertebrate (animal living wholly or chiefly in or on water)
      - S: (n) vertebrate, craniate (animals having a bony or cartilaginous skeleton with a segmented spinal column and a large brain enclosed in a skull or cranium)
        - S: (n) chordate (any animal of the phylum Chordata having a notochord or spinal column)
          - S: (n) animal, animate being, beast, brute, creature, fauna (a living organism characterized by voluntary movement)

NLP Techniques

- Language Models (LM)
- Syntactic analysis
- Semantic analysis
- Sentiment analysis

Sentiment Analysis

- Classify the polarity of a text

+3 -1 -4

"I **love** this movie but I **really** **hate** the main actor."

positive booster negative

Positive sentiment strength: 3

Negative sentiment strength: -5

NLP Techniques

- Language Models (LM)
- Syntactic analysis
- Semantic analysis
- Sentiment analysis
- Emotion analysis

Emotion Analysis

- Joy: “That’s great work guys!”
- Anger: “ I will come over to your work and slap you!”
- Sadness: “Sorry for the late response.”
- …

Outline

- Introduction
- Background on IR and NLP
- SE tasks using IR and NLP
  - Task definition
  - Input
  - Output
  - Preprocessing
  - Techniques
  - Evaluation
  - Tools used
- Hands-on

Improving the Quality of the Code Lexicon

- ✓ Identifying poor quality identifiers
- ✓ Identifying naming inconsistencies

TASK

SUBTASK

Identifying Poor Quality Identifiers

- Task: Identifying identifiers that are difficult to understand, unclear, meaningless, etc.

- Examples:
  - aSz
  - foo

Identifying Poor Quality Identifiers

- Source code
- Mapping between program identifiers and domain concepts
- Standard lexicon dictionary (a dictionary of allowed terms)
- Synonym/abbreviation dictionary

OUTPUT

Identifying Poor Quality Identifiers

- Identifiers with poor quality
- Suggestions to improve the identifiers

PREPROCESSING

Identifying Poor Quality Identifiers

– Splitting

Identifying Poor Quality Identifiers

– Identifying non-standard lexicon using dictionaries

meaningless: `foo`

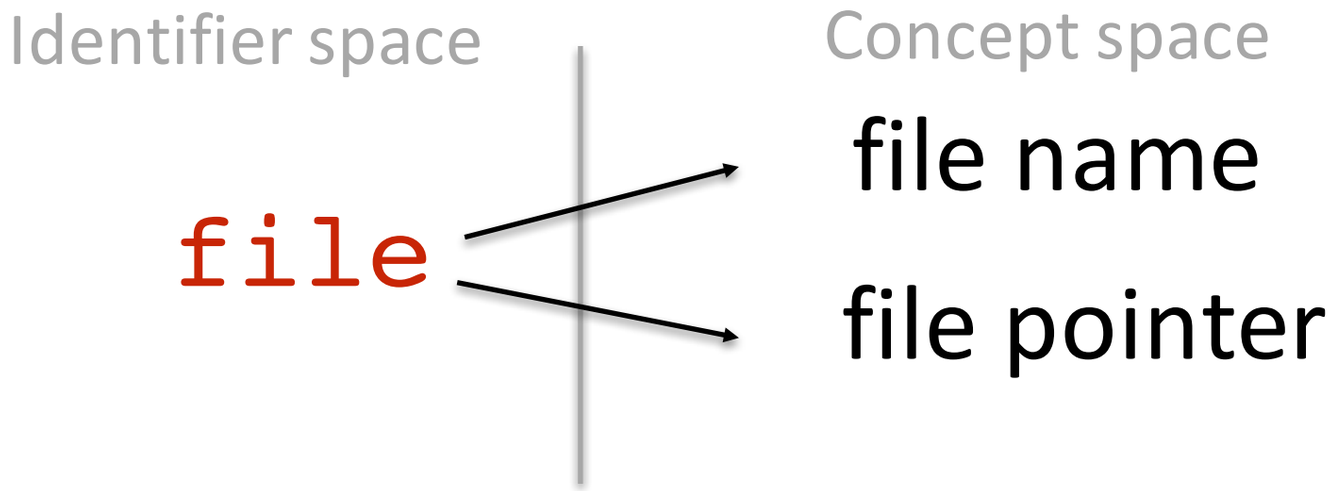
synonyms: `a`**`Copy`** `and print`**`Replica`**

abbreviations: `aSz` // `a`: array, `Sz`: size

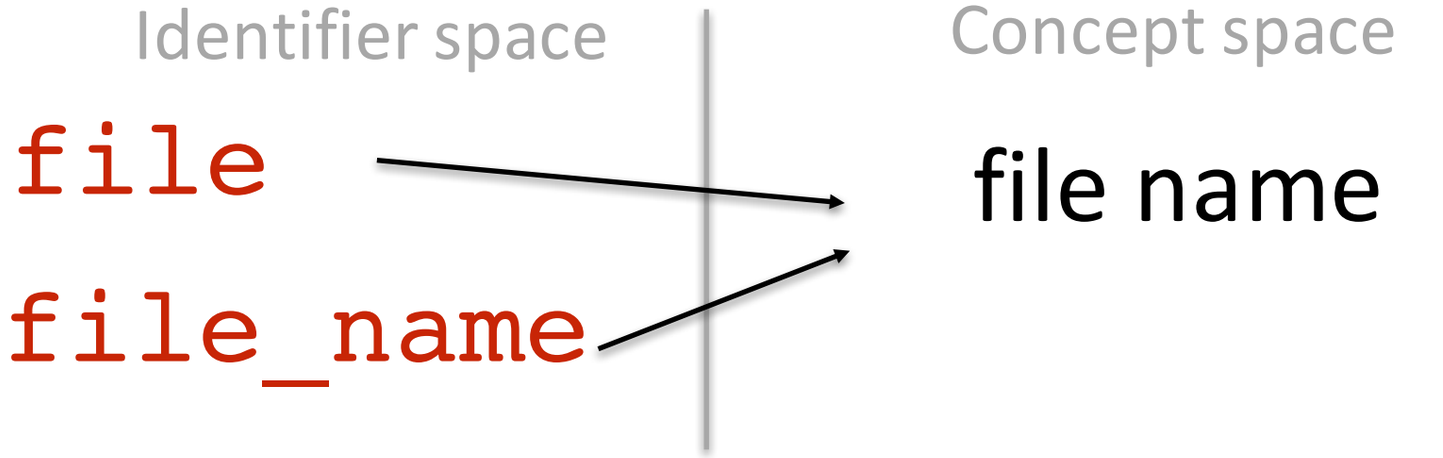
Identifying Poor Quality Identifiers

– Identifying inconsistencies using mappings between identifiers and concepts

Homonym:



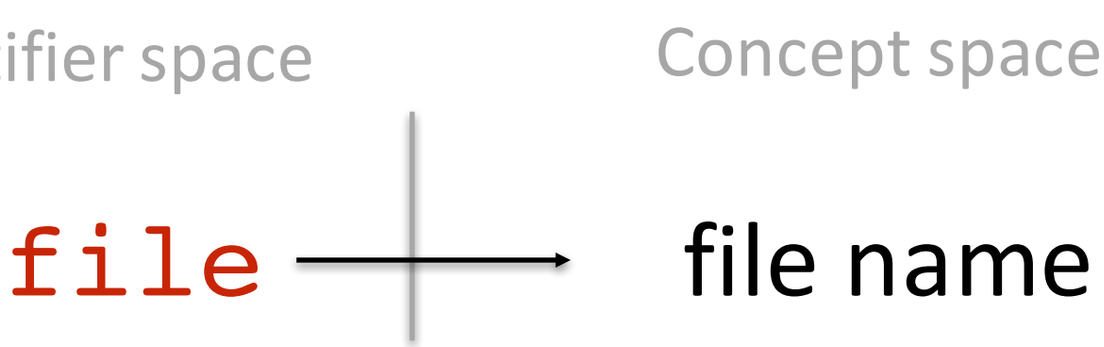
Synonym:



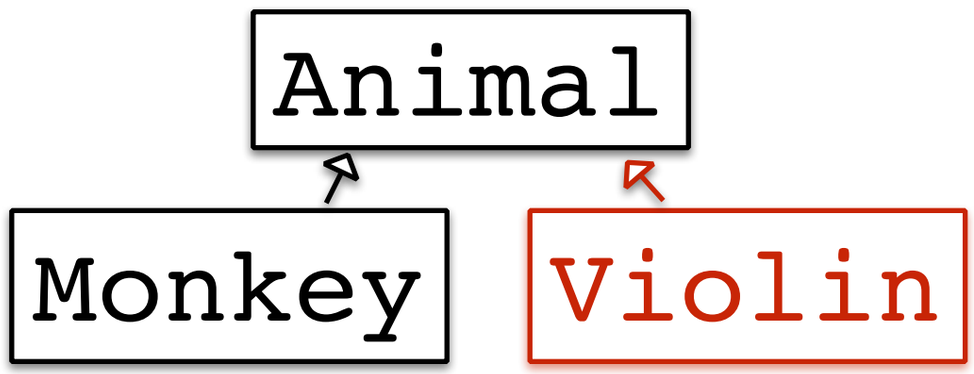
Identifying Poor Quality Identifiers

– Identifying inconsistencies using mappings between identifiers and concepts (cont.)

Conciseness violation:



No hyponymy in a class hierarchy:



Identifying Poor Quality Identifiers

- Inconsistencies based on the concepts (cont.)
  - Identified using:
    - identifiers to concept mapping
    - identifier inclusion (syntactic conciseness and consistency)
    - ontology
    - number of characters
    - string similarity

Identifying Poor Quality Identifiers

- Syntactical standardization

```
class:     Compute      // must be a noun
method:    addition     // must be a verb
```

FunctionId	::=	[Context] (Action | PropertyCheck | Transformation)	
Context	::=	Qualifier <noun>	
Qualifier	::=	(<adjective> | <noun>)*	
Action	::=	SimpleAction | ComplexAction	
SimpleAction	::=	DirectAction | IndirectAction	
ComplexAction	::=	ActionOnObject | DoubleAction	
IndirectAction	::=	Qualifier <noun> ActionSpecifier	{Head word = <noun>}
DirectAction	::=	<verb> ActionSpecifier	{Head word = <verb>}
ActionSpecifier	::=	(<adjective> | <adverb> | <preposition> Qualifier <noun>)*	
...			

Identifying Poor Quality Identifiers

– Other types of measures

overloaded identifiers: `saveAndPrint`

spelling errors: `Examlpe`

useless type: `String name`**`String`**

- Identified using POS analysis, grammatical relations, spell checker, identifier containment

EVALUATION

Identifying Poor Quality Identifiers

- Case study with quantitative and qualitative analyses
- Precision of detected poor quality identifiers

TOOLS USED

Identifying Poor Quality Identifiers

- Semantic relations: WordNet
- POS tagging:
  - Minipar
  - WordNet
- Spell checker: Jazzy

Identifying Naming Inconsistencies

- Task: Identify entities where the name is inconsistent with the type, functionality, or documentation.
- Examples:
  - method named `isNavigateForwardEnabled` documented as backward navigation
  - method named `iterator` whose implementation is only creating and returning an object

Identifying Naming Inconsistencies

- Project bytecode
- Source code

INPUT

Identifying Naming Inconsistencies

- Inconsistencies
- Suggested solution

OUTPUT

PREPROCESSING

Identifying Naming Inconsistencies

- Splitting
- Tokenization

Identifying Naming Inconsistencies

– Contrast the name and type of an entity

opposite name and type:

```
EnterTransport
  exitTransport(..)
```

says many, contains one:

```
boolean statistics
```

Identifying Naming Inconsistencies

– Contrast the name and comment of an entity

opposite name and comment:

// … default exclude …

```
String INCLUDE_NAME_DEFAULT
```

- Defined through a grounded theory approach
- Identified using POS analysis, general ontology, grammatical relations

Identifying Naming Inconsistencies

– Contrasting the name and implementation of an entity

Semantic profile of an “iterator” method:

These methods often **call** other methods with the same name and create objects. They never **return** void, **write** parameter values to fields or call themselves recursively, and very rarely write to fields or return parameter values, and rarely have parameters, contain loops, use local variables, do runtime type-checking or casting, return field values, have branches or have multiple return points.

```
public Iterator iterator() throws
        DomainRegistryException{…}
```

EVALUATION

Identifying Naming Inconsistencies

- Detection precision
- Developers' perception

TOOLS USED

Identifying Naming Inconsistencies

- Semantic relations:
  - WordNet
- POS tagging:
  - WordNet
  - Stanford's POS Tagger

TASK

Building Software Ontologies

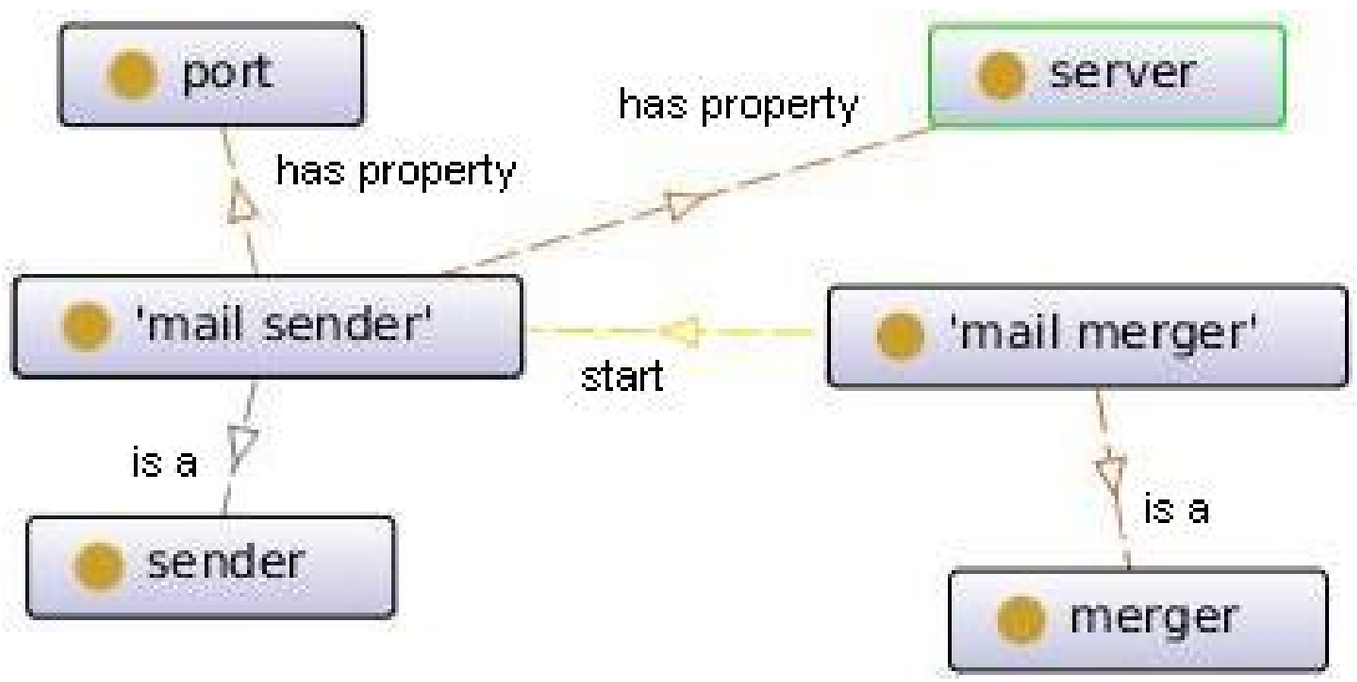
✓ Domain ontology

✓ Identifying semantically related words

TASK

Extracting Domain Concepts

- Task: automatically extracting domain concepts and relations from source code
- Examples:



Extracting Domain Concepts

- Source code
- Documentation (e.g., user manuals, web sites)

OUTPUT

Extracting Domain Concepts

- Domain concepts
- Ontological relations

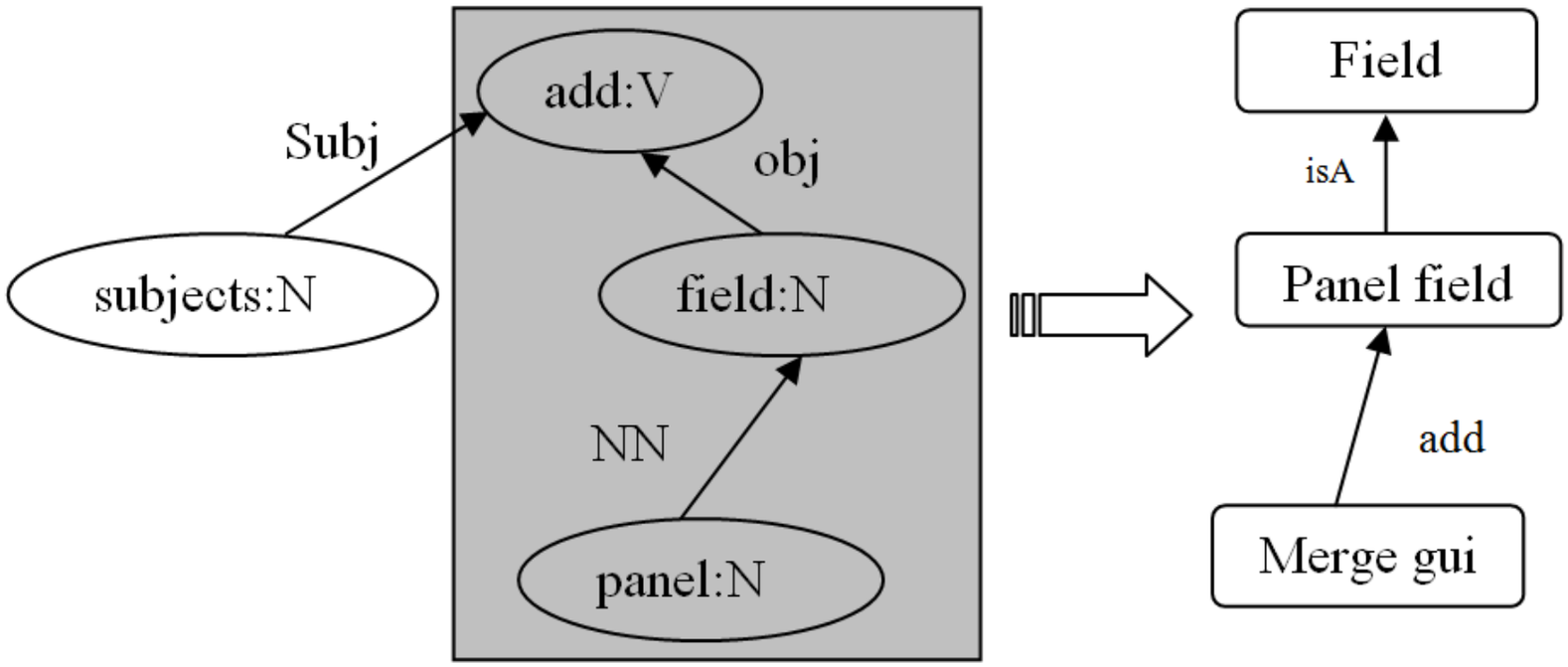
Extracting Domain Concepts

- Splitting
- Tokenization
- Abbreviation expansion
- Stop words removal
- Stemming

Extracting Domain Concepts

- Sentence templates based on constraints for different types of entities
- Example: method `addPanelField` defined in class `MergeGui` generates sentence:

"Subjects add panel field"



TECHNIQUE

Extracting Domain Concepts

– Filter the ontology using terms based on:
  - Keywords
  - LDA

Extracting Domain Concepts

- Precision of the POS tagging
- Number of connected components
- Case study: navigating the concepts for query reformulation in the context of bug location
- Precision and recall of the extracted domain concepts compared to a gold set
- Qualitative analysis

Extracting Domain Concepts

- POS tagging:
  - Minipar
  - WordNet
- Grammatical relations:
  - Minipar
  - TreeTagger
- Topic modeling: Dragon Toolkit

SUBTASK

Identifying Semantically Related Words

- Task: Identifying pairs of words that are semantically related, e.g., same or opposite meaning
- Examples:
  - call - invoke
  - size - capacity
  - serialize - deserialize
  - header - trailer
  - `makeFullMap` - `makeEmptyMap`

INPUT

Identifying Semantically Related Words

- Project description and tags extracted from a hosting site
- Source code

OUTPUT

Identifying Semantically Related Words

- Similar words
- Ranked list of similar tags

Identifying Semantically Related Words

- Splitting
- Tokenization
- Stop words removal
- Stemming

Identifying Semantically Related Words

– Similarity between terms (VSM with tf-idf)

$$sim(t_1, t_2) = w_1 \times dsim(t_1, t_2) + w_2 \times tsim(t_1, t_2)$$

– Cluster tags using the similarity between terms to build a hierarchical taxonomy

Identifying Semantically Related Words

– High similarity between pairs of sentences containing at least one common word

"None **mounted** file for this track."

"None **accessible** file for this track."

"If you do not have **apr_pool_clear** in a wrapper"

"If you do not have **apr_pool_destroy** in a wrapper".

Identifying Semantically Related Words

– Frequency of comment-code word pairs of main action verbs for methods

```
/** Searches an attribute.*/
XMLAttribute findAttribute(…){…}

/** Cancels the current HTTP request.*/
void jsxFunction abort(){…}
```

EVALUATION

Identifying Semantically Related Words

- Precision of the identified pairs of words
- User study evaluating a subset of the identified pairs on a Likert scale.
- Sensitivity evaluation for thresholds (precision and recall)

TOOLS USED

Identifying Semantically Related Words

- Stanford's POS Tagger for comments
- Custom POS Tagger for method names
- WordNet

TASK

Generating Documentation Automatically

✓ Extracting a set of important keywords

✓ Generating natural language sentences

SUBTASK

Extracting a Set of Important Keywords

- Task: Identify the keywords that best represent a software artifact
- Example: {“match”, “text”, “ignorecase"}

```
public static boolean regionMatches(boolean ignoreCase,
    Segment text, int offset, char[] match) {
    int length = offset + match.length;
    if(length > text.offset + text.count)
        return false;
    char[] textArray = text.array;
    for(int i = offset, j = 0; i < length; i++, j++)
    {
        char c1 = textArray[i];
        char c2 = match[j];
        if(ignoreCase)
        {
            c1 = Character.toUpperCase(c1);
            c2 = Character.toUpperCase(c2);
        }
        if(c1 != c2)
            return false;
    }
    return true;
}
```

INPUT

Extracting a Set of Important Keywords

- Source code
- Execution traces

OUTPUT

Extracting a Set of Important Keywords

- Sets of keywords that best represent each
  - Class
  - Method
  - Execution trace segment

PREPROCESSING

Extracting a Set of Important Keywords

- Splitting
- Tokenization
- Stop words removal
- Stemming

TECHNIQUE

Extracting a Set of Important Keywords

- Compare IR-techniques
- Eye-tracking experiment to decide on the importance of terms
- IR-techniques: VSM, LSI, LDA
- Weighting schemes: tf, tf-idf, log, and binary

EVALUATION

Extracting a Set of Important Keywords

- Developers assessing the quality of the summaries
- Comparison with manually summarized artifacts

SUBTASK

Generating Natural Language Sentences

- Task: Generating natural language sentences summarizing a software artifact.
- Examples
  - Method summary: “Export plan component to svg.”
  - Class summary: “An AbstractPlayer extension for m player handlers. This entity class consists mostly of mutators to the m player handler's state. …”
  - Release note: “New class SearcherLifetimeManager implementing Closeable. …”

INPUT

Generating Natural Language Sentences

- Project source code
- Set of releases
- Issue tracker
- Version control repository

OUTPUT

Generating Natural Language Sentences

- Natural language sentences representing
  - method comments
  - class comments
  - release notes
  - commit notes

Generating Natural Language Sentences

- Splitting
- Tokenization
- Abbreviation expansion

Generating Natural Language Sentences

- Method summaries
  - Statement selection
    - Ending statements
    - Statement with a method call with the same action
    - Conditional expressions
    - …

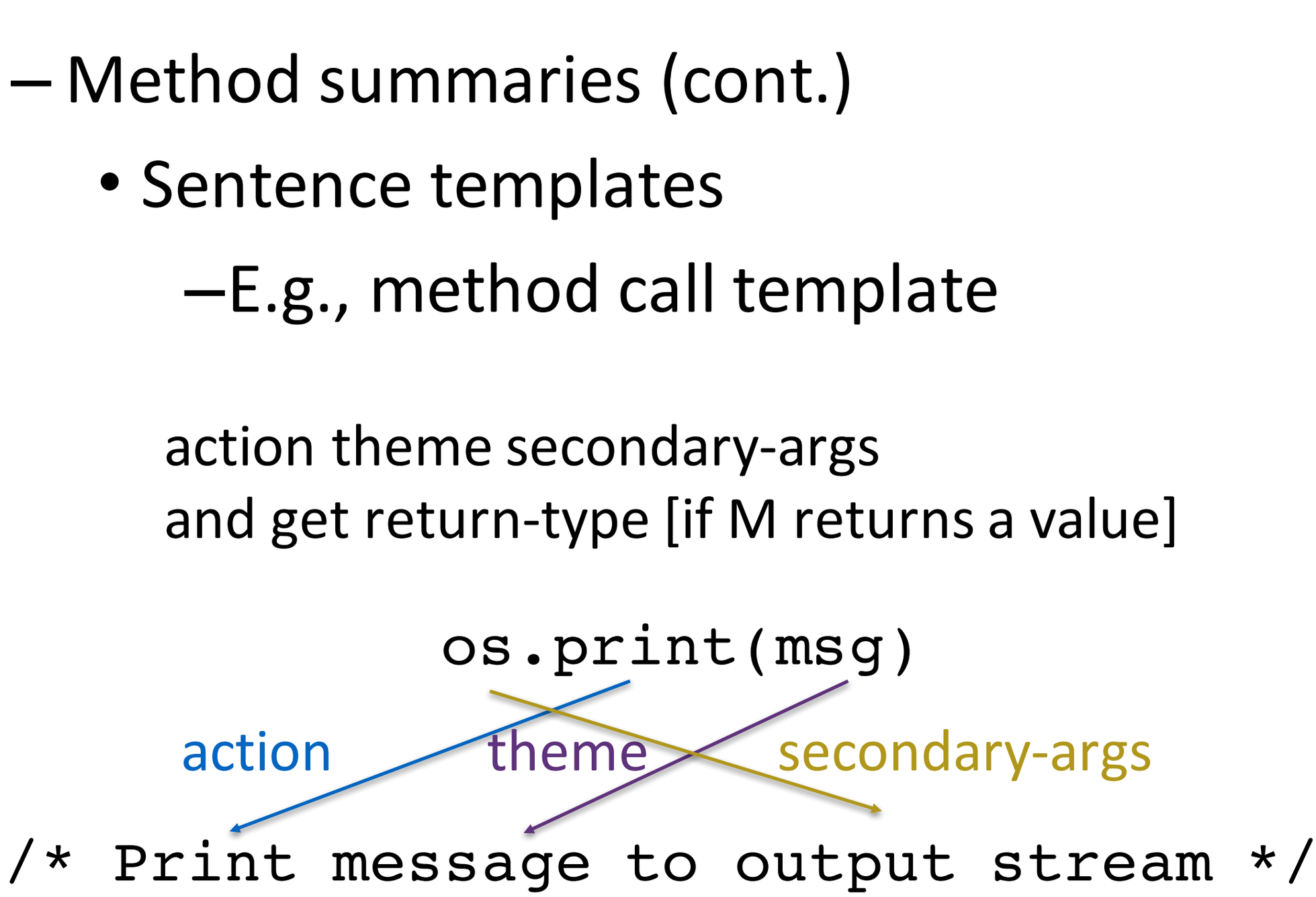
Generating Natural Language Sentences

– Method summaries (cont.)

- Sentence templates

–E.g., method call template

action theme secondary-args
and get return-type [if M returns a value]



Generating Natural Language Sentences

- Class summaries based on class and method stereotypes
  - Text generation
    - General description
    - Stereotype description
    - Behavior description
    - Inner classes enumeration

Generating Natural Language Sentences

- Class summaries based on class and method stereotypes

```
public class MPlayerHandler extends AbstractPlayer {

    public static final boolean GAP = false;

    private static final String LIN
    private static final String WIN

    private static final String QUI
    private static final String SLA

    private Process process;

     * @stereotype CONSTRUCTOR
    public MPlayerHandler() {

     * @stereotype COLLABORATOR
    private static boolean testMPla

     * @stereotype SET
    private void play(AudioFile f)

     * @stereotype COMMAND
    public void finish() {
    ...
```

An AbstractPlayer extension for m player handlers. This entity class consists mostly of mutators to the m player handler's state.

It allows managing:
- mute;
- volume; and
- next with no gap.

It also allows:
- finishing m player handler;
- handling next;
- playing audio file f;
- stopping m player handler;
- playing m player handler; and
- handling previous.

Generating Natural Language Sentences

- Release notes by organizing changes hierarchically and by using sentence templates
  - Identifying and prioritizing code changes from the versioning systems
    - Files added, removed, moved
    - Classes added, removed, renamed, moved
    - Methods changed (signature, visibility, source code, or set of thrown exceptions)
    - …

Generating Natural Language Sentences

- Release notes by organizing changes hierarchically and by using sentence templates
  - Sentence templates
    - Deleted file: “File <file name> has been removed.”
    - Added class: class summaries (JSummarizer)

Generating Natural Language Sentences

- Release notes by organizing changes hierarchically and by using sentence templates
  - Other changes considered
    - Licensing
    - Documentation
    - Libraries
    - Refactorings
    - Issues

TECHNIQUE

ARENA

Automatic RElease Notes generAtor - Apache Commons Codec 1.7

New Features

- CODEC-136 Use Charset objects when possible, create Charsets class for required character encodings
- CODEC-133 Add classes for MD5/SHA1/SHA-512-based Unix crypt(3) hash variants.
- CODEC-88 Base32 encoder
- CODEC-63 Implement NYSIIS

Bug fixes

- CODEC-157 DigestUtils: Add MD2 APIs
- CODEC-156 DigestUtils: add APIs named after standard alg name SHA-1
- CODEC-155 DigestUtils.getDigest(String) should throw IllegalArgumentException instead of RuntimeException
- CODEC-152 DigestUtils.getDigest(String) looses the orginal exception
- CODEC-147 BeiderMorse phonetic filter give uncertain results
- CODEC-132 BeiderMorseEncoder OOM issues
- CODEC-131 DoubleMetaphone javadoc contains dead links
- CODEC-130 Base64InputStream.skip skips underlying stream, not output
- CODEC-96 Base64 encode() method is no longer thread-safe, breaking clients using it as a shared BinaryEncoder

Improvements

- CODEC-151 Remove unnecessary attempt to fill up the salt variable in UnixCrypt
- CODEC-150 Remove unnecessary call to Math.abs()
- CODEC-148 More tests and minor things
- CODEC-143 StringBuffer could be replaced by StringBuilder for local variables
- CODEC-139 DigestUtils: add updateDigest methods and make methods public.
- CODEC-138 Complete FilterInputStream interface for BaseNCodecInputStream

Deprecated Code Components

Added Code Components

Refactored Source Code Files

Other Changes

Known Issues

Generating Natural Language Sentences

- Developers
  - Accuracy
  - Content Adequacy
  - Conciseness
  - Importance
  - In-field study

SUBTASK

Generating Natural Language Sentences

- Tools used:
  - Software Word Usage Model (SWUM)
  - JSummarizer for generating class summaries

Outline

- Introduction
- Background on IR and NLP
- SE tasks using IR and NLP
- Hands-on

Tools to install

- srcML: for transforming source code to xml: http://www.srcml.org/downloads.html
- XOM: for parsing xml files and querying them with XPath: http://www.xom.nu/
- WordNet: for extracting semantic relations: http://wordnet.princeton.edu/wordnet/download/current-version/
- JWNL, Java API for WordNet: https://sourceforge.net/projects/jwordnet/files/jwnl/JWNL%201.4/jwnl14-rc2.zip/download
- Stanford CoreNLP for grammatical analysis: http://stanfordnlp.github.io/CoreNLP/download.html

Preparations

- Download the source code: http://www.veneraarnaoudova.ca/wp-content/uploads/2016/07/SaTToSE_16-TR_NLP.zip
- Change `ProjectConfiguration`
- Parse source code with SrcML
- Run `SaTToSETest`

TODOs

- Extract name and preceding comment of attributes and methods
- Find antonyms in the name and comments of an attribute
- Check if method names start with a verb

More TODOs

- XPath: Fix `getPreceedingCommentsForAttribute` to collect only the comments for the attribute that is passed as parameter.
- Preprocessing: remove stopwords before querying WordNet for semantic relations
- WordNet: change `startsWithVerb` to use WordNet in addition to POS tagging to reduce false negatives
- Stanford CoreNLP: extract negation relations and incorporate them in the antonym detection