Expanding Identifiers to Normalize Source Code Vocabulary

Paper by: Dawn Lawrie & Dave Binkley

Presentation by: Ryan Wiles

Problem Being Solved

- Vocabulary Normalization
- Builds on previous work by the authors in “Normalizing source code vocabulary” [17]
- Source Code contains significant invented vocabulary
- Unify language gap between source code and natural language vocabulary
- IR tools struggle with abbreviations and acronyms
- General purpose solution

Background

- Example Identifiers: sponge_bob, spongeBob, spongebob, hashtable_entry, CMSentry
- Hard Words - based on division markers; i.e. breaks on underscores and Camel Casing, sufficient if a dictionary word like “sponge bob”. Not always easy CM Sentry or CMS entry?
- Soft Words - No dictionary match; Either entire hard word or a substring of it “hashtable” is composed of two soft words “hash” & “table”. Hard words can also be a soft word even if a dictionary match.
- Context matters! Normalized identifiers should make sense.

Approach

- Split hard word to soft words
- Assign meaning to soft words
- In case of abbreviations, may require use of a phrase finder [17] on software artifacts. What does “cms” mean?
- Find the most appropriate soft words based on context
- Context primarily defined via Google 5-gram via Linguistic Data Consortium
- Maximum Coherence Model - a machine translation technique forms basis of the Normalize algorithm

Algorithm

- GenTest() - Splits identifier into hard words and then soft words. Provides possible word expansions; top-ranked and top ten ranked splits are considered [17]
- Acronym Expansion - Source Code and other Software Artifacts are mined for possible acronym matches [21]
- Expand() finds the probabilistic best split of identifiers in a given context based on the strongest cohesion
- Score() looks at the average similarities over all pairs of words and each expanded word with each context word. The highest score is chosen.
- Similarity scores computed as log sim(w_1, w_2) where sim represents their conditional probability of how often w_1 occurs in the context of w_2.

Expand()

- log sim(x, x) = 0
- Cohesion($e_{i,j}$,s,C) = $\sum_{s_k \in s\ !=\ s_i}$ Cohesion′($e_{i,j}$,s_k) + $\sum_{c \in C}$ log sim($e_{i,j}$,c)
- Cohesion′($e_{i,j}$,s_k) = $\sum_{e \in E(s_k)}$ log sim($e_{i,j}$,e)
- Expand′(s_i, C) = $e_{i,j} \in E(s_i)$ s.t. $\forall e \in E(s_i)$, Cohesion($e_{i,j}$, s, C) ≥ Cohesion(e, s, C)
- Expand(s,C) = Expand'(s_1,C)Expand'(s_2,C)… Expand'(s_n,C)

Score()

- Score′(w_i, s, C) = $\sum_{w_j \in \text{Expand}(s,C)} \log \text{sim}(w_i, w_j) + \sum_{c \in C} \log \text{sim}(w_i, c)$
- Score(s, C) = $\left(\sum_{w_j \in \text{Expand}(s,C)} \text{Score}'(wi, s, C)\right) / n(n + c)$
- Normalize(id, C) = Expand(s) s.t. $s \in$ Splits(id) and $\forall s' \in$ Splits(id), Score(s, C) $\geq$ Score(s′, C)

Normalize()

- Performs Expand() & Score()
- Identifies split that has the highest score

Example

- Identifier: strlen => splits [str-len, st-rlen]
- String expansions: str => [steer, string]; len => [lender, length]; st => [stop, string, set]; rlen => [riflemen]
- Similarity scores:

log sim(stop,riflemen) = $4.5384 \times 10{-}8$

log sim(string,riflemen) = $4.5384 \times 10{-}8$

log sim(set,riflemen) = $4.5384 \times 10{-}8$

log sim(steer,lender) = $4.5384 \times 10{-}8$

log sim(steer,length) = $4.5384 \times 10{-}8$

log sim(string,lender) = $4.5384 \times 10{-}8$

log sim(string,length) = 0.00455585

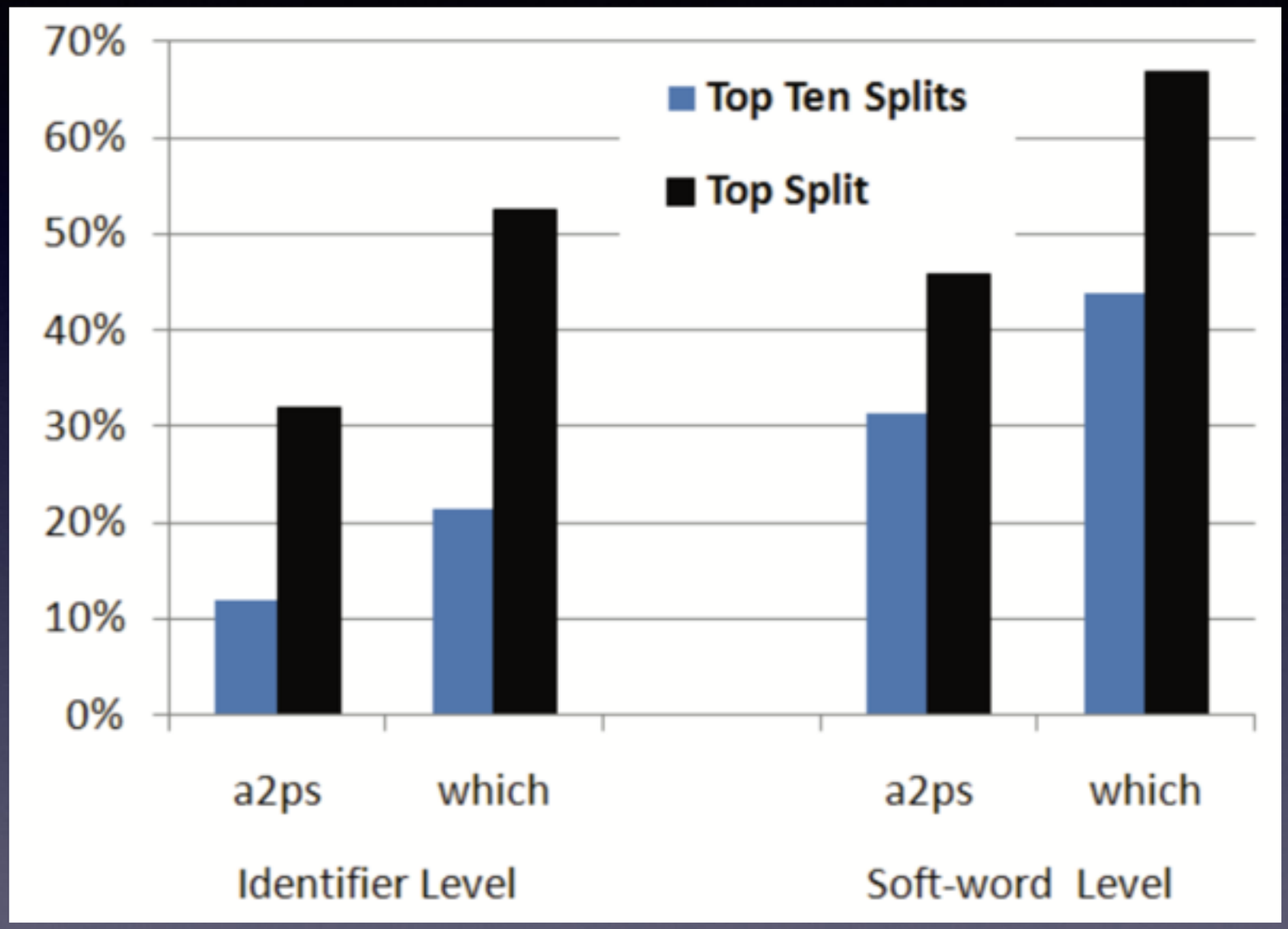
Example continued

- Cohesion′(string, len) = log sim(string, lender) + log sim(string, length)
- Cohesion(string, str-len, ∅) = –32.1713 & Cohesion(steer, str-len, ∅) = –48.7865; Maximal cohesion says we choose “string”
- Similarly “length” maximizes cohesion for “len”
- Expand(str-len, ∅) = string-length; score = –7.778064037
- Expand(st-rlen, ∅) = set-riflemen; score = –24.39324099
- Normalize() maximizes cohesion and returns string-length

Evaluation

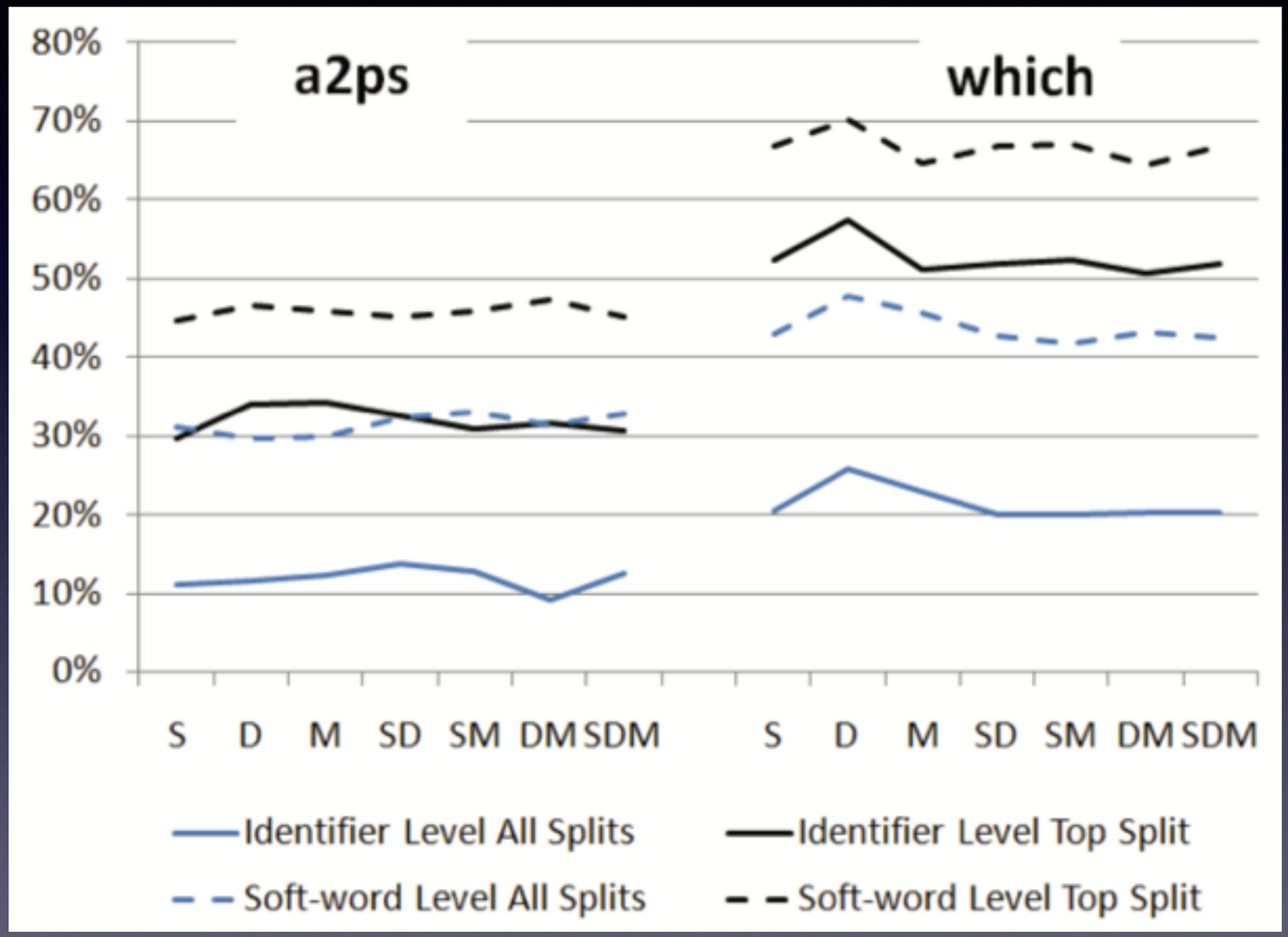
- Two programs evaluated: which, a2ps
- Oracle evaluation to define gold standard for identifier expansion

Evaluation Continued



Performance of Normalize averaged over the seven vocabulary sets studied

Evaluation Continued



Accuracy of identifier expansion based on vocabulary used

Summary of Work

- First general purpose identifier expander
- Normalize can expand abbreviations from arbitrary source code
- Performance does not significantly degrade when only using the source code
- Identifier normalization is a critical step in using source code improving IR tools and techniques

Interpretation of Work

- Software artifacts, principally the documentation, can improve identifier normalization, however unfortunate extraneous vocabulary can compromise results
- Including software documentation can typically improve results
- Their soft word Level Top Split either on source or documentation can improve accuracy, other artifacts don’t necessarily improve accuracy
- Method of inclusion of software artifacts in cohesion is not defined
- Identifier Normalization is a hard problem: Maybe in tool use recommend renaming badly named identifiers

Strengths

- Improved performance over Field et al: 66% accuracy vs 40% when compared against Oracle definitions
- Use of additional software artifacts is not necessary, but can boost performance
- Generic approach, not limited to a restricted syntactic context or special purpose rules

Weaknesses

- Lack of domain context from other artifacts can hurt accuracy, failed expansions in a2ps without documentation
- Some artifacts can reduce accuracy, user manual in which
- Computationally expensive, identifier expansion can take up to 8 seconds
- Some stop words frequently used in identifiers can reduce accuracy. i.e. “is”

Take Aways

- Abbreviations and Acronyms in source code present challenges to IR tools and techniques
- Methods exist to help normalize the invented vocabulary in source code
- There are techniques leverage context to improve identifier expansion improve accuracy
- Don’t contribute to the problem. Use literate programming to create easily understandable identifiers

Questions?