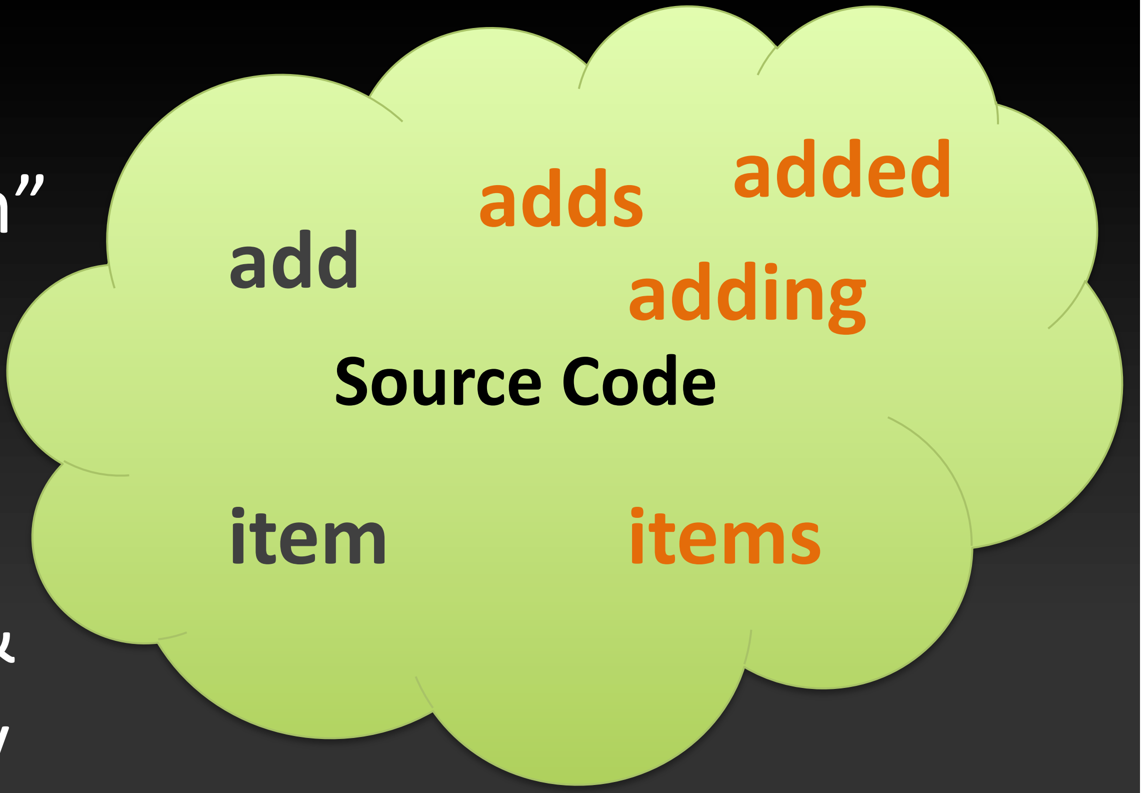
On the Use of Stemming for Concern Location and Bug Localization in Java

Emily Hill
Montclair State University

Shivani Rao, Avinash Kak
Purdue University

Problem: Source Code Search

- Query: “add item”



add adds added adding

Source Code

item items

- Stemming used to strip suffixes & improve recall by reducing words to root form, or stem
  - Widely studied in Information Retrieval (IR)
  - Not so much for SE (very different document style)

A Brief History of Stemming

- Light Stemmers (tend not to stem across parts of speech)
  - Porter (1980): algorithmic (rule-based), simple & efficient
    - Most popular stemmer in IR & SE
    - Snowball (2001): minor rule improvements
  - KStem (1993): morphology-based
    - based on word's structure & hand-tuned dictionary
    - in experiments shown to outperform porter's
- Heavy Stemmers (can overstem, reducing precision)
  - Lovins (1968): algorithmic
  - Paice (1990): algorithmic
  - MStem: morphological (PC-Kimmo), specialized for source code using word frequencies

Our Contribution

Investigate use of stemming for 2 different types of Java source code search tasks with various queries:

- Bug Localization: find methods in bug fix (IR: Unigram Model) *291 bugs from iBugs dataset (ASPECTJ) with queries:*
  - M291: all 291 bugs, with initial bug description as query (not title)
  - Medium: initial bug description of 126 bugs that contain both title & comments (not much code)
  - Short: title of 126 bugs – Long: title + full comments of 126 (some code)
- Concern Location: find methods implementing a concept of interest, given keyword-style queries (IR: tf-idf)
  - 8 action-oriented concerns from 4 programs (AOC), 48 queries
  - 215 documentation-based concerns from Rhino (Rhino), 645 queries

Analysis Methodology

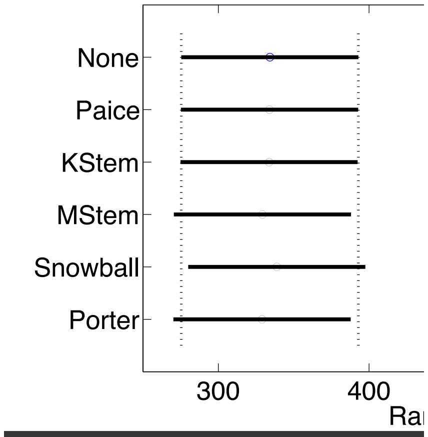
- MAP (Mean Average Precision):
  AP = average precision at each relevant result
- Rank Measure [Hull ’96]:
  rank of relevant documents for each query
- Qsets [Hull ’96]: partition queries into sets:
  - Q_+: stemming helps
  - Q_-: stemming hurts
  - $Q_=$: stemming has no effect
  - Q_{vary}: effect depends on stemmer

Results: Bug Localization

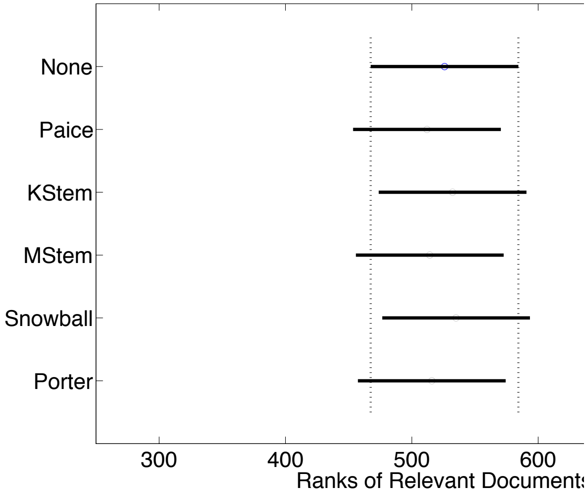
MEAN MAP DIFFERENCE SCORES

Query	Paice	KStem	MStem	Snowball	Porter
Long	-0.02806	-0.006090	-0.004199	-0.008529	-0.01055
Medium	0.002638	0.007970	0.01400	-0.003243	0.007280
M291	-0.006703	-0.002056	-0.003738	-0.008391	-0.002462
Short	-0.008479	0.003343	0.0004492	0.0003093	-0.002758

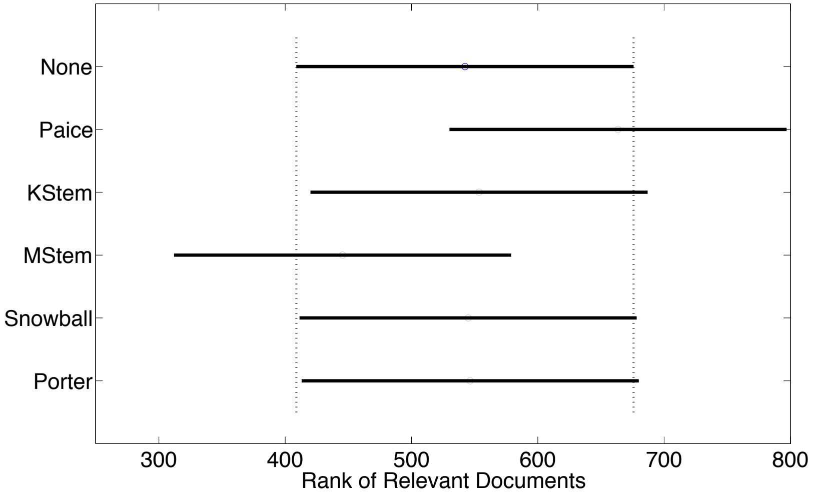
Long



M291

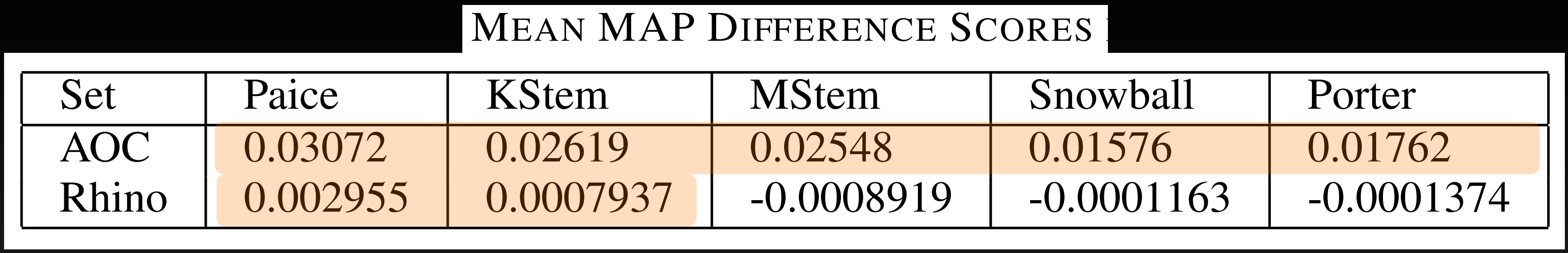


Short



- Stemming plays more of a role for shorter queries

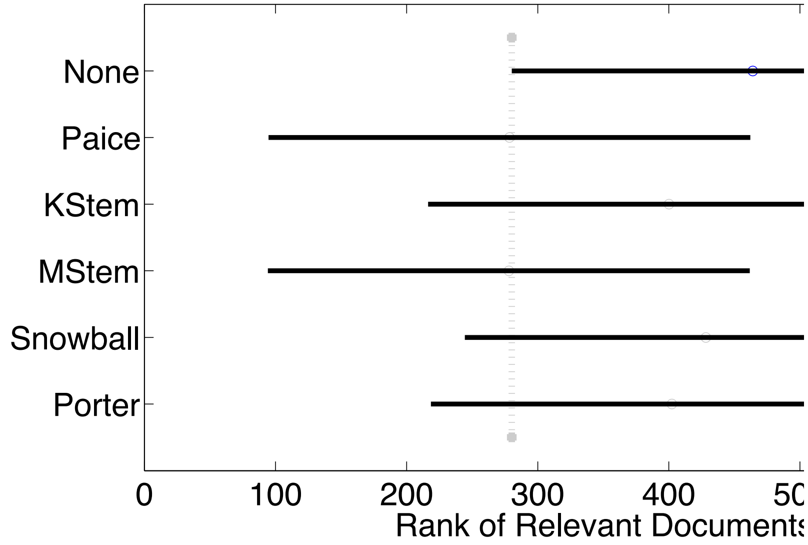
Results: Concern Location



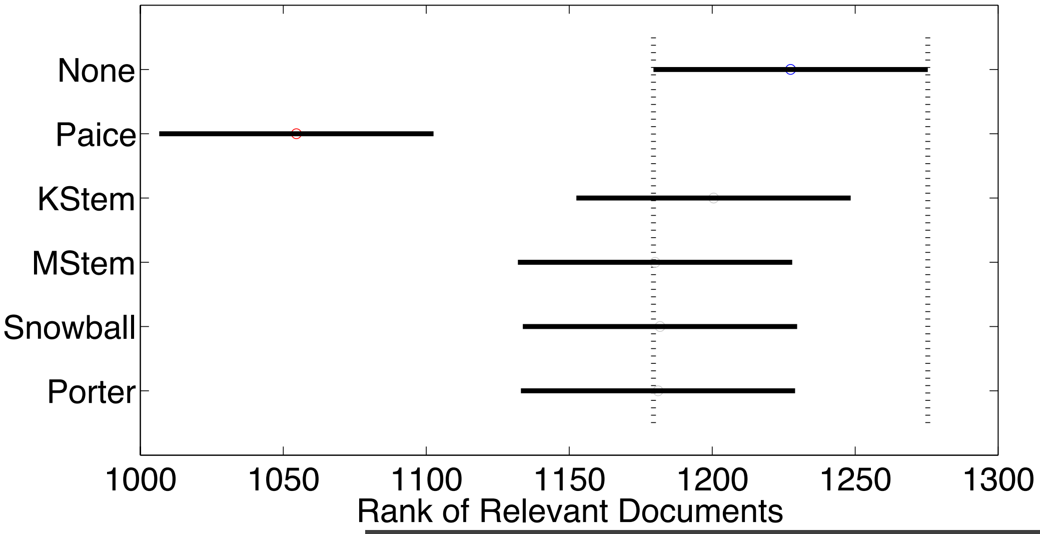
MEAN MAP DIFFERENCE SCORES

Set	Paice	KStem	MStem	Snowball	Porter
AOC	0.03072	0.02619	0.02548	0.01576	0.01762
Rhino	0.002955	0.0007937	-0.0008919	-0.0001163	-0.0001374

AOC

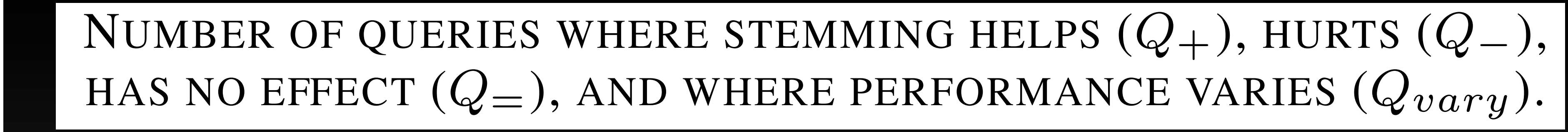


Rhino



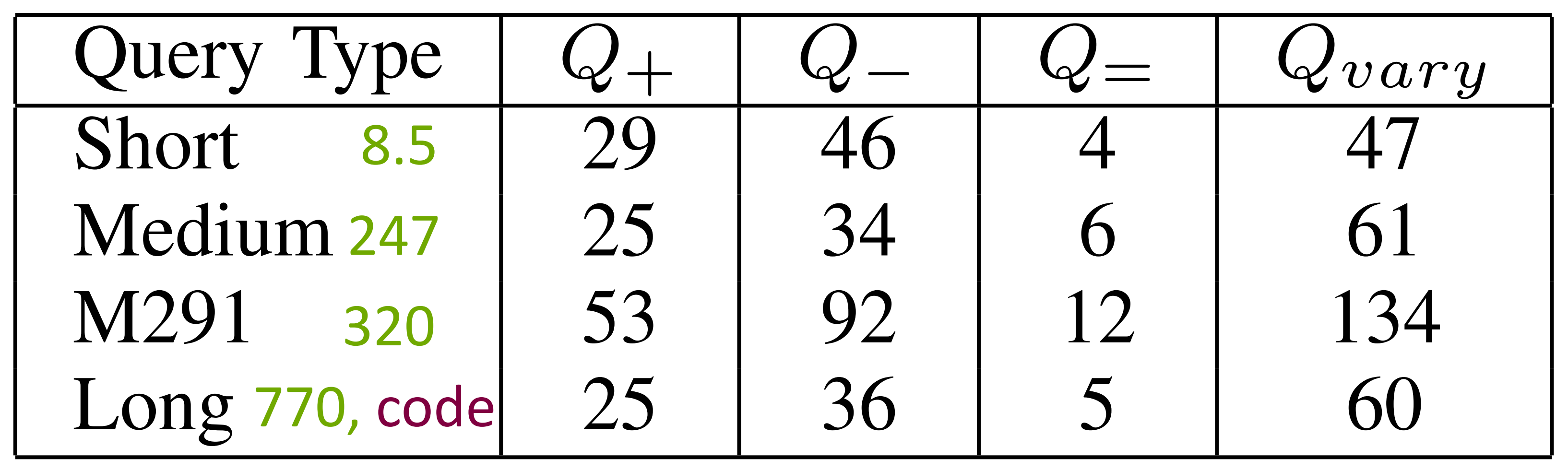
- Paice significantly outperforms other stemmers for Rhino, points to possible interaction with tf-idf

Results: Qsets



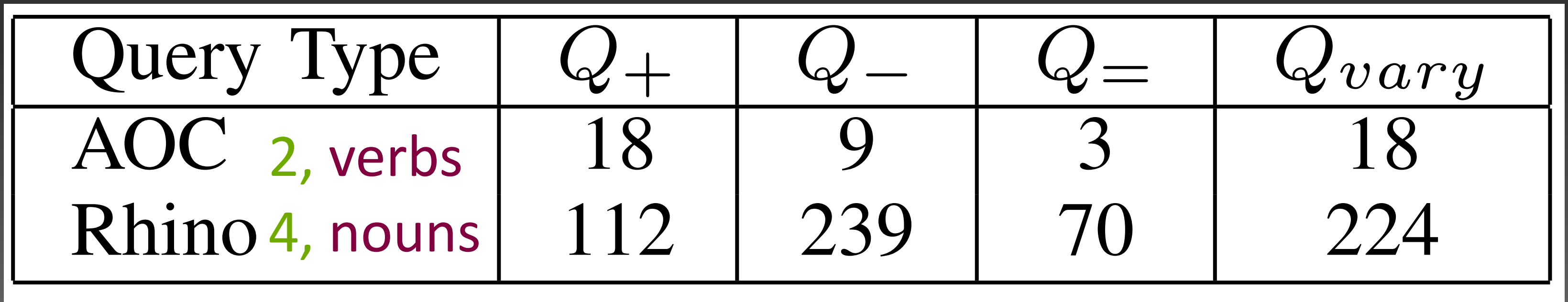
NUMBER OF QUERIES WHERE STEMMING HELPS (Q_+), HURTS (Q_-), HAS NO EFFECT ($Q_=$), AND WHERE PERFORMANCE VARIES (Q_{vary}).

- Bug Localization Mean number of words in query, notes



Query Type	Q_+	Q_-	$Q_=$	Q_{vary}
Short 8.5	29	46	4	47
Medium 247	25	34	6	61
M291 320	53	92	12	134
Long 770, code	25	36	5	60

- Concern Location



Query Type	Q_+	Q_-	$Q_=$	Q_{vary}
AOC 2, verbs	18	9	3	18
Rhino 4, nouns	112	239	70	224

Conclusion & Discussion

- So far, success of any particular stemmer situation dependent (we can’t yet generalize)
  - Stemmer success seems dependent on query nature & retrieval model
- Are there other variables missing from our model of the problem, or is this due to the nature of stemming/searching itself?
  - Query length, presence of code/identifiers
  - Query difficulty (how well matches code words)
- Future Work: explore the interaction between retrieval model, query length/type, & stemmer