# Passage Retrieval

Group 6
Chuck Curtis, Matt Hohensee,
Nathan Imse

# Building The Index -- Lucene Style!

- Used PyLucene
  - Documentation = "See Lucene Java Documentation"
- First attempt: split documents into individual files
  - 6+ GB
- Second attempt: store document text in index
  - 3.47 GB
- Stemming
  - 0.1004 => 0.1062 MAP
  - 200 docs/query
- Expanded Stop List
  - 0.2969 => 0.3111 MAP
  - 1000 docs/query + Target Concatenation

# Lucene in the Sky with Diamonds -- future experimentation

- Current system
  - Indexed only body of text
    - Title + Text
- Proposed System
  - Store document title in separate field
    - Possibly boost the weight
  - Synonyms?
    - Other groups didn't report positive results
    - Would put it in the Analyzer, not in QE module
    - Common vs Rare words
      - Which gives the most bang for the buck?

# Query Expansion

- Attempted to emulate the Xu & Croft (1996) Local Context Analysis (LCA) algorithm
- Tried a couple of variations for the IDF numbers
  - IDF_t = log(N/count_t)
  - N = number of top passages
  - N = number of total documents in the corpus
- Query expansion was giving us some noisy concepts
  - e.g. ,", i
  - filtered out non-alphanumeric concepts and concepts less than 3 characters long
- Experimented with weighting
- Query expansion never boosted our results, and always had a negative impact (up to ~40% reduction)
- BOO

# Passage retrieval

- Based on the ISI approach
- ISI used only weighting of various matching terms to re-rank all the sentences in all the documents and keep the top 300
- Sample ISI weighting
  - Exact match of proper names gets a bonus
  - Upper case matches of more than one word get a bonus
  - Lower case matches get a smaller bonus
  - etc.

# Passage retrieval

- We used a sliding window, 3 sentences long (did not get a chance to try different window sizes)
  - Overlapping 3-sentence windows
  - This was rarely over 1000 characters - truncated the end when it was
- Each 3-sentence passage got a score based on term matching (details on next slide)
- Highest 20 passages per document were returned

# Passage retrieval - term matching

- In each 3-sentence passages:
  - count query term overlap: remove first word if it is a wh-word; remove stopwords
  - count expanded term overlap: look for all terms in expanded query (lower-cased and stemmed), and if found, add the weight assigned by Lucene
  - count bigram overlap (lower-cased)
  - count trigram overlap (lower-cased)
  - count occurrences of named entities in query
  - count occurrences of "target" word
- All these are weighted heuristically and added together for a total score

# Final System and Results

- No query expansion
- PorterStemFilter
- NLTK Stop Word List
- 200 documents per query
- 3-sentence windows

| | **Training data (TREC 2004)** | **Test data (TREC 2005)** |
|---|---|---|
| **MAP** | 0.3103 | 0.3078 |
| **MRR (strict)** | 0.2168 | 0.2428 |
| **MRR (lenient)** | 0.3112 | 0.3795 |

# Evaluation "Paradox"

- Evaluation is very dependent on previous work
  - Encourages finding the same relevant documents as earlier systems
  - Penalizes finding relevant documents not found by previous systems
- Solutions?
  - Create a Passage Retrieval system that can automatically extract documents and passages from a corpus, so that we have a better evaluation
    - Requires solving the same problem we're working on
  - Lock a bunch of poor graduate students in a room and have them make manual decisions on the corpus
    - Give them free snack food