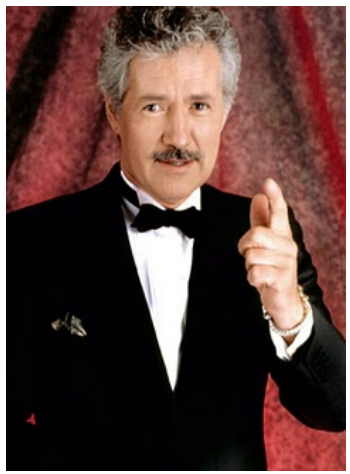
TREBEK

(Text REtrieval Boosted by Exterior Knowledge)



Group 6:
Chuck Curtis, Matt Hohensee,
Nathan Imse

Back to the Drawing Board

- Went back and essentially re-implemented D3
- Changes to Document Retrieval:
  - Slightly more document cleaning in the indexing stage
    - Gave us slightly better MAP with 200 docs/query than we previously got with 1000 docs/query
  - Target token weights boosted to 1.5 query token weights
- Utilized Web Boosting to guide Passage Retrieval
- Utilized Thresholding of PyLucene document retrieval
  - Helped more with runtime than performance

Web Boosting

- urllib2 and BeautifulSoup python libraries
- Simple pronoun replacement for query reformulation
  - Query: When was he born?
  - Target: Fred Durst
  - New Query: When was Fred Durst born?
  - if no pronoun found, then target is concatenated to beginning of query
- Scraped result abstracts from Ask.com
  - Two settings: first page only and first 10 pages
- Why Ask.com?
  - Easy to generate URL's
  - Consistent results

Why Not Use Aranea? That's What All the Cool Kids are Doing...

- Already had most of our scraping in place before the Aranea GoPost exploded
  - didn't want to change horses mid-river
- Our scraping was plenty fast
  - essentially as fast as reading from local caches
    - 40-60 seconds for the TREC 2004 data
- No API's meant that we didn't have to worry about critical methods being deprecated

Web Boosting

- Tested the utility of web text by using it as a "passage" and computing MRR
- Attempted to reduce the average length of the web text while maintaining the MRR

	MRR	Avg # Characters
First page	0.71	2413
First 10 pages	0.88	26839

Web Boosting -- K-Medoids

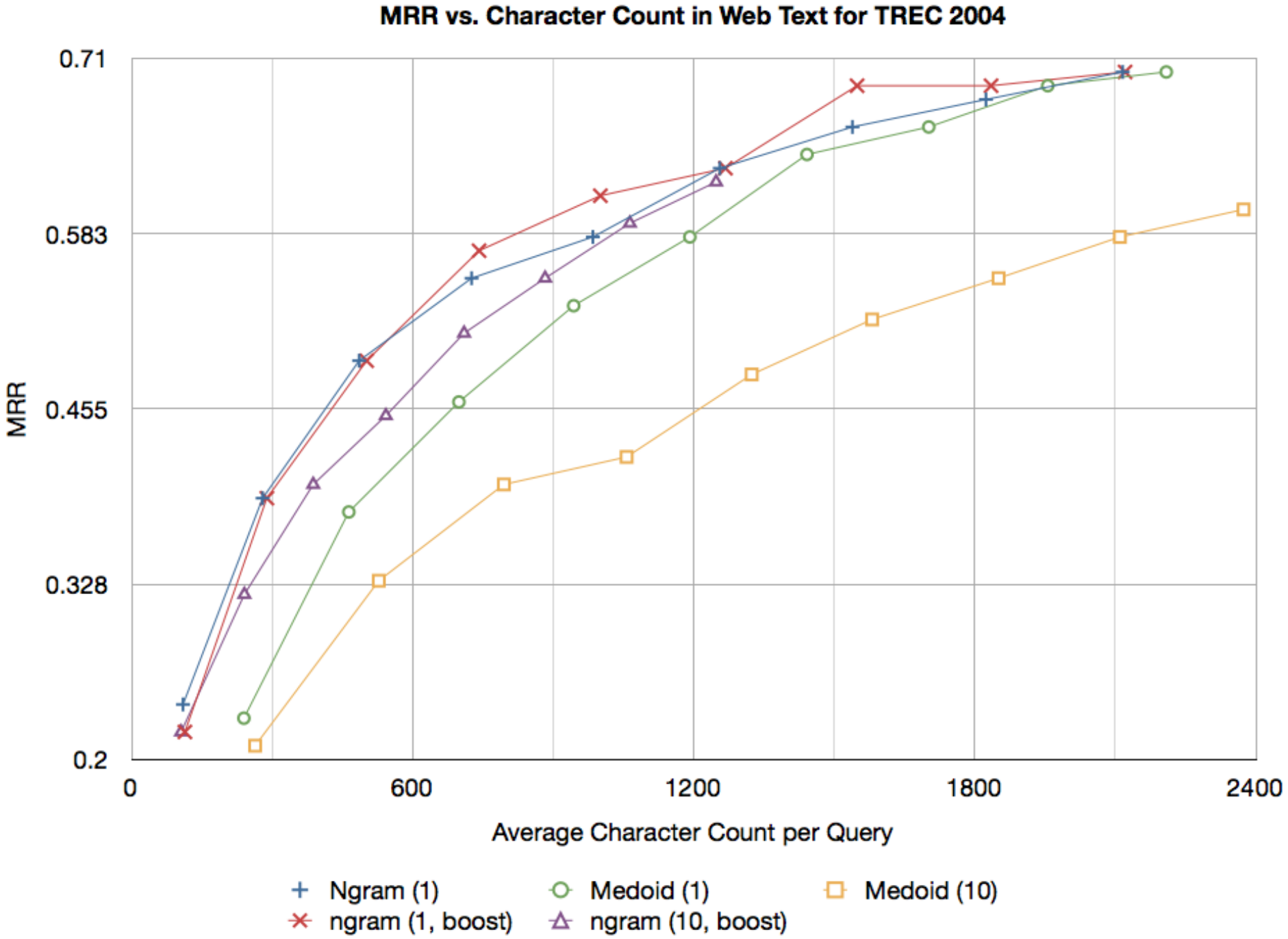
- Had no idea if it would work
- Performed K-Medoid clustering on sentences in the web text
- Cosine Similarity
- Medoids at convergence were assumed to be the more representative sentences
- Relies on repetition of answers in the web text
- Surprisingly good performance
  - not very robust against noise

Web Boosting -- Ngram Overlap...ish

- Found that unigrams were the most effective
- Each sentence in the web text was scored according to the following equation:

$$A = \text{all tokens in web text}$$
$$T = \text{tokens in query Target}$$
$$W = \text{question words}$$
$$f(i,j) = \text{frequency of token } i \text{ in text } j$$
$$w = \text{web text}$$
$$x = \text{sentence}$$

$$S(x) = \sum_{i \in A} \left[\frac{f(i,w) + f(i,x)}{len(x)} \right] + \sum_{j \in T} \left[\frac{14 \cdot f(j,x)}{len(x)} \right] - \sum_{k \in W} \left[\frac{f(k,x)}{len(x)} \right]$$


MRR vs. Character Count in Web Text for TREC 2004
0.71
0.583
0.455
0.328
0.2
MRR
0
600
1200
1800
2400
Average Character Count per Query
Ngram (1)
ngram (1, boost)
Medoid (1)
ngram (10, boost)
Medoid (10)

Passage Retrieval

- D3: sentence-based algorithm
  - scored each 3-sentence window based on overlap with query terms, etc.
  - truncated if it was over 1000 characters
  - this worked reasonably well, but for D4 we want to scale to smaller windows
- Tried 2-sentence window (usually < 1000 char)
  - 0.3567 lenient MRR on first 10 question groups
- Tried extracting "most contentful" 100-char passage
  - based on NEs, titlecasing, digits, etc.
  - 0.2277 lenient on first 10 groups

Passage Retrieval Redux

- Tried using text from web boosting instead of query text
- Crawl through document looking at 1000-, 250-, and 100-char passages
  - Compute cosine similarity to web text
  - Also tried looking at passage content: boosted score slightly if passage contained titlecasing, uppercasing, or digits
  - Query text, target term, answer type not used at all

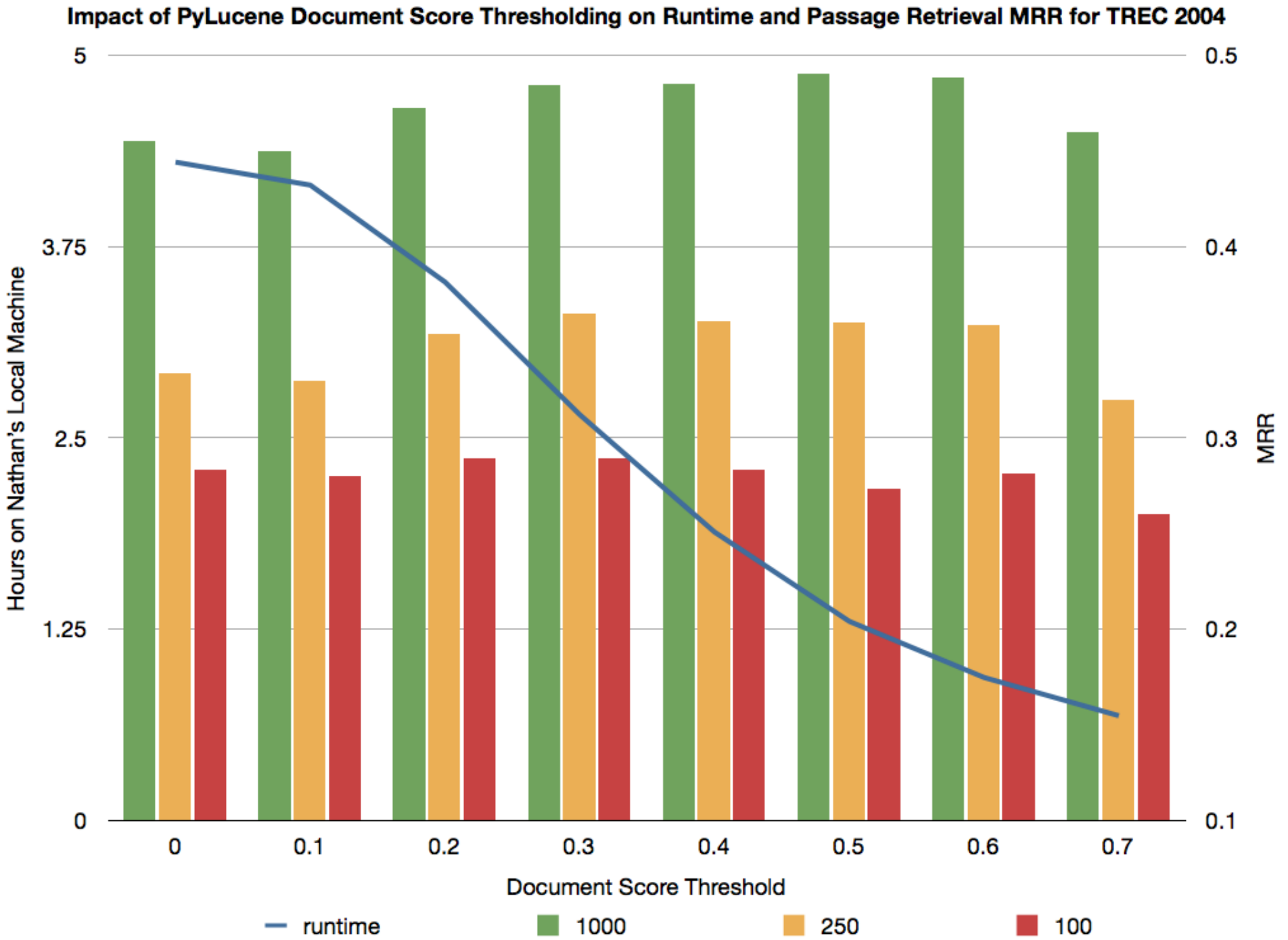
I'll take "Passage Retrieval" for $400, Alex

Results on first 10 question groups from TREC-2004:

Window size	Increment	Lenient MRR using cosine sim only	Lenient MRR using cosine sim and content score	Run time
1000	500	0.5214	0.5412	~15m
250	125	0.3804	0.3300	~18m
250	50	0.3978	---	~45m
100	50	0.2689	0.2414	~20m

Final system:

- no content scoring
- increment = half of window size


Impact of PyLucene Document Score Thresholding on Runtime and Passage Retrieval MRR for TREC 2004
Hours on Nathan's Local Machine
MRR
5
3.75
2.5
1.25
0
0.5
0.4
0.3
0.2
0.1
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
Document Score Threshold
runtime
1000
250
100

Final Results

	1000 chars	250 chars	100 chars
2004 Strict	0.309	0.247	0.188
2004 Lenient	0.488	0.359	0.281
2005 Strict	0.243	0.147	0.117
2005 Lenient	0.461	0.273	0.208

Improvement over D3

	D3	D4	% Change
2004 Strict	0.2168	0.309	+42.5%
2004 Lenient	0.3112	0.488	+56.8%
2005 Strict	0.2428	0.243	+0.1%
2005 Lenient	0.3795	0.461	+21.5%

If Only We Had More Time...

- Utilize query classification from D2 in our answer extraction
- Try things like FrameNet and Pattern Searching
- If we could get a concise answer from the web data, then we would try:
  - feeding it into our PyLucene queries
  - use more of a search than similarity-based algorithm among the documents
- Clean the TREC-related paper abstracts from the web text