Today's Plan

- Housekeeping
- A few things about HWK 1
- Introduction to HWK 2
- Elias Gamma Encoding

Housekeeping

- Next Monday is a holiday, so class will be held on Wednesday, Feb 18 from 6:30 – 9:30 in c106
- TA online office hours…
  - Are the times good?
  - What can I tell them to make it work better?

HWK1: Indexing Workflow

- Prepare the index
- Parse the AP89 files
- Index each document as it is parsed
- Verify that all documents have been indexed

Elasticsearch Python: Prepare the index

```
idx = elasticsearch.client.IndicesClient(es)
idx.delete('myindex')
idx.create(index='myindex',
        body = '' +
            '{"mappings": {' +
            '"text": {' +
            '"properties": {' +
            '"body": {' +
            '"type" :"string",' +
             '"term_vector": "with_positions_offsets_payloads",' +
            '"store" : true }}}}}')
```

Note that you can index documents without this step, but they will have defaults for "term_vector" which may mean you don't get position information, for example

HWK 1: workflow

To parse the documents:

<DOC>

<DOCNO> AP890110-1029 </DOCNO>

<HEAD> The document header – we are ignoring this for now

</HEAD>

<TEXT>

The text we care about.

</TEXT>

</DOC>

]

There may be more than one text element in each <doc>

HWK 1 (and HWK2) : Parsing the document

1. Iterate through all files in the directory
2. For each file iterate through the lines of the file
    a. If you see <TEXT> set Flag = True
    b. If you see </TEXT> set Flag = False
    c. If you see /<DOCNO>\s*([^<]+)\</
        Capture the document id
    d. If you see </DOC> index the document, clear all variables
    e. else, if Flag == True, aggregate the text

Elasticsearch: Index your documents

Python:

es.index(index='ap89', doc_type='text', id=docno, body={'body': doc_body})

In Python you can check that your method call is correct in the interpreter:

Launch the python interpreter:

Vanessas-MacBook-Pro:~ Vanessa$ python

Python 2.7.9 (v2.7.9:648dcafa7e5f, Dec 10 2014, 10:10:46)

[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin

Type "help", "copyright", "credits" or "license" for more information.

>>>

>>> import elasticsearch

>>> es = Elasticsearch()

>>> es.index(index='myindex', doc_type='text', id='AP891001-1001', body={'body':'this is some text i want to index'})

Elasticsearch Indexing

- Check that your index contains all the documents:

curl 'localhost:9200/_cat/indices?v'

health status index pri rep docs.count docs.deleted store.size pri.store.size

yellow open ap89 5 1 84678 0 504.9mb 504.9mb

grep '<DOCNO>' AP89/AP_DATA/ap89_collection/* | wc –l

84679

Elasticsearch: Ranking

- Get the statistics
  - Per term:
    - Term frequency (TF)
    - Document frequency (DF)
    - Collection Frequency (TTF in elasticsearch
  - Per Document:
    - Document length (|D|)
  - Once for the collection:
    - Average document length (ave(|D|))
    - Vocabulary size (|V|)

Elasticsearch: term statistics

- For each term in the query, get all the documents:

```
results = es.search(index='myindex', q='body:'+term, size=1000000)
docs = [doc['_id'] for doc in results['hits']['hits']]
```


```
"hits" : {
   "total" : 1000,
   "max_score" : 1.0,
   "hits" : [ {
     "_index" : "bank",
     "_type" : "account",
     "_id" : "1",
     "_score" : 1.0, "_source" : {"account_number":1,"balance":
39225,"firstname":"Amber","lastname":"Duke","age":32,"gender":"M","address":"880 Holmes
Lane","employer":"Pyrami","email":"amberduke@pyrami.com","city":"Brogan","state":"IL"}
   }, {
     "_index" : "bank",
     "_type" : "account",
     "_id" : "6",
        …
```

Term Statistics (Java)

```
// get average doc length
      StatisticalFacet f = getStatsOnTextTerms(client, index, type, null, null);
      double avgLenDoc =  f.getMean();
// get current document length
          double lenDoc =  getStatsOnTextTerms(client, index, type, "docno",
key).getTotal();
```

http://www.elasticsearch.org/guide/en/elasticsearch/reference/current/modules-advanced-scripting.html

Some things to remember

- Remove the stop words from the queries using the stop list provided with the data
- Your mean average precision should be around 0.15
  - If you are getting around 0.05, this is essentially a random result

HWK 2: Indexing

- Two things you will re-use from HWK 1:
  - Parse the collection
  - Score the documents
- JSON need not be generated
- Parsing in this case means:
  - Stemming
  - Removing stop words
  - Converting the textual terms to integers
- Your MAP results (trec_eval) should be the same independent of the indexing

Parsing the document

1. Iterate through all files in the directory
2. For each file iterate through the lines of the file
    a. If you see <TEXT> set Flag = True
    b. If you see </TEXT> set Flag = False
    c. If you see /<DOCNO>\s*([^<]+)\</
        Capture the document id
    d. If you see </DOC> index the document, clear all variables
    e. else, if Flag == True, aggregate the text

Parsing the document, II

3. Convert all text to lower case
4. Tokenize the text, recording the position of each term
5. Remove (or replace) punctuation
   - "the car wash" is 3 tokens
   - "est. 1975" is 2 tokens
   - "Dan's car wash" is ??? tokens
   - "175.85.0.101" is ??? Tokens?
   - https://www.elasticsearch.org/index.php is ??? Tokens?
6. Convert all text to lower case

Parsing the document, III

- Remove stop words ("stopping")
- Stem the collection ("stemming")
- Convert all tokens to integers
- int in java is 32 bits,
  - represents -2^{31} to $2^{31} - 1$.
  - If your vocabulary is longer than this, you will need to use a long
    - (The vocab is unlikely to be larger than this in AP89)

Inverted Index

- Recall that the output you want is:

<term_id> : <doc_id>[position list];<doc_id>[position list];…

- The document ids should be in sorted order to make query processing efficient
- The mapping from term to term_id has to be stored, because the query will also have to be converted into a term_id

Searching the Index

- The index is a file of the form:

<term_id> : <doc_id>[position list];<doc_id>[position list];…

- How do we search this?

Searching the inverted index

- Option 1: iterate through the file to find the matching key terms
  - In the worst case, the entire vocabulary is searched for every term in the query
- Option 2: Sort the keys, and do binary search
  - Worst case is O(log n) for search (sorting is O(n log n) or O(nk) for radix sort, but this is done in advance, not at search time)

We can do better…

Searching the Inverted Index

- Space is cheap
- Store a table of term_id, file offset (in bytes)
- This can be stored in memory (for instance in a hash table)
- Lookup in the hash table to get the byte offset is O(1), and accessing the byte offset in the file is O(1)

HWK 2

- Ultimately you produce:
  - A file that contains the mapping of terms to term_ids
  - A file that contains the mapping of term_ids to byte offsets in the index file
  - An inverted index file that contains term_ids, and postings lists, and possibly other statistics
  - Possibly a file with corpus statistics: total vocab size, number of documents, ave doc length, total number of terms
  - If you want to retrieve the original document, you will need a file that has a mapping from document_id to original docno, original file name, plus byte offsets in the ap89 file.

HWK 2, II

- You will need to write the code to:
  - parse the collection to produce these files
    - With and without stopping and stemming
  - Parse the queries in the same way the collection was parsed
  - Efficiently search the index
  - Scores the documents
- Verify that your models have the same performance with your index that they had with Elasticsearch.

From Last time…

- Delta Encoding
- Elias Gamma Encoding

Elias Gamma Encoding

Decimal number = $2^N + k$	Binary number	Gamma encoding (N zeros)
$1 = 2^0 + 0$	1	1 (zero 0's)
$2 = 2^1 + 0$	10	0 10 (1 zero)
$3 = 2^1 + 1$	11	0 11 (1 zero)
$4 = 2^2 + 0$	100	00 100 (2 zeros)
$5 = 2^2 + 1$	101	00 101 (2 zeros)
$6 = 2^2 + 2$	110	00 110
$7 = 2^2 + 3$	111	00 111
$8 = 2^3 + 0$	1000	000 1000 (3 zeros)
$9 = 2^3 + 1$	1001	000 1001 (pattern forming…)
$10 = 2^3 + 2$	1010	000 1010
$11 = 2^3 + 3$	1011	000 1011

Elias Gamma Encoding

- Let N = floor($\log_2 x$) be the highest power of 2, such that $2^N \leq x < 2^{N+1}$
- Write out N zeros
- Append the binary form of x

Elias Gamma Decoding

- The gamma encoded number is symmetrical:
  - N zeros "1" N digits
- Read and count zeros from the stream until you reach a one.
  - Count of zeros = N
- The one that was read is the first digit of an integer, and indicates the 2^N portion of the number
- Read the remaining N digits of the integer