Exercise 6: Heap’s Law & Zipf’s Law

- For solutions see Moodle and Jupyter
Lecture last week: Ranked Retrieval

Inverted index adapted to ranked retrieval

\[ \frac{tf_{tq} \times idf_t^2 \times tf_{td}}{||q|| \times ||d||} \]
Exercise 7

BONUS TIME
Exercise 7: Vector space models

- 12/16 points needed to pass
- Only 1 coding question
- Exercise starts Friday, 19th April, 11:00
- Exercise ends Thursday, 2nd May, 23:59
Exercise 7: Vector space models

**SMART notation:**

<table>
<thead>
<tr>
<th>Term frequency</th>
<th>Document frequency</th>
<th>Normalization</th>
</tr>
</thead>
<tbody>
<tr>
<td>n (natural)</td>
<td>n (no)</td>
<td>n (none)</td>
</tr>
<tr>
<td>l (logarithm)</td>
<td>t (idf)</td>
<td>c (cosine)</td>
</tr>
<tr>
<td>a (augmented)</td>
<td>p (prob idf)</td>
<td>u (pivoted unique)</td>
</tr>
<tr>
<td>b (boolean)</td>
<td></td>
<td>b (byte size)</td>
</tr>
<tr>
<td>L (log ave)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Term frequency</th>
<th>Document frequency</th>
<th>Normalization</th>
</tr>
</thead>
<tbody>
<tr>
<td>(tf_{t,d})</td>
<td>(N/\text{df}_t)</td>
<td>(1/(\sqrt{w_1^2 + w_2^2 + \ldots + w_M^2}))</td>
</tr>
<tr>
<td>(1 + \log(tf_{t,d}))</td>
<td>(\max{0, \log(N-\text{df}_t)/\text{df}_t})</td>
<td>(1/\text{ut}) (Section 6.4.4)</td>
</tr>
<tr>
<td>(0.5 + \frac{0.5\times tf_{t,d}}{\max_i(tf_{i,d})})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\begin{cases} 1 &amp; \text{if } tf_{t,d} &gt; 0 \ 0 &amp; \text{otherwise} \end{cases})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\frac{1+\log(tf_{t,d})}{1+\log(\text{ave}<em>e \in d(tf</em>{t,d}))})</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Exercise 7: Vector space models

- Programming:
  - Ranked retrieval system based on vector space model
  - Can handle free form text queries
- Your part:
  - Build the index
  - Implement helper function calculating term frequency, document frequency, inverse document frequency, tf-idf weight and cosine normalization
  - Build scoring function
Exercise 7: Vector space models

- Build the index
  - For each term we need to know:
    1. Number of documents in which the term appears
    2. Number of times a term appears for each of the documents that contain the term
  - Up to you how to build such a data structure

```python
def add_document(doc):
    ...  
    Add a document to the inverted index. Returns the document's ID
    ...
    global documents, docid_counter, the_index
    # do not re-add the same document.
    if doc in documents.values():
        return
    docid = docid_counter
    documents[docid] = doc
    docid_counter += 1
    print("Adding document %s to inverted index with document ID %d" % (doc, docid))
    for term in tokenize_document(doc):
        # TODO: collect term frequencies and document frequencies per term.
pass
```
Exercise 7: Vector space models

- Implement helper functions:
  - How they work depends on your data structure
  - Implement \( tf(\text{term}, \text{docid}) \)
    - Returns term frequency \( tf_{\text{term,docid}} \) for the term \( \text{term} \) in the document with id \( \text{docid} \).

```python
def tf(term, docid):
    ...
    Calculate term frequency for term in docid. Return 0 if term not in index,
    or term does not appear in document.
    ...
    # TODO: implement
    return 0
```
Exercise 7: Vector space models

- Implement helper functions:
  - Implement \( \text{df}(\text{term}) \)
    - Returns document frequency \( df_{\text{term}} \) of a term \( \text{term} \)

```python
def df(term):
    ...

    Extract frequency of term for document with id docid from index
    ...

    # TODO: implement
    return 0
```
Exercise 7: Vector space models

- Implement helper functions:
  - Implement \( \text{idf}(\text{term}) \)
    - Returns inverse document frequency \( \text{idf}_{\text{term}} \) of a term \( \text{term} \)
    - \( \text{idf}_{\text{term}} = \log\left(\frac{N}{\text{df}_{\text{term}}}\right) \)
    - Reuse previous implemented \( \text{df} \)

```python
def idf(term):
    # Compute idf_t for a term
    # TODO: implement
    return 0
```
Exercise 7: Vector space models

- Implement helper functions:
  - Implement \( \text{tf\_idf}(\text{term}, \text{docid}) \)
    - Returns tf-idf weighting \( \text{tf\_idf}_{\text{term},\text{docid}} \) for the term \( \text{term} \) in the document with id \( \text{docid} \).
    - \( \text{tf\_idf}_{\text{term},\text{docid}} = \text{tf}_{\text{term},\text{docid}} \cdot \text{idf}_{\text{term}} \)
  - Reuse \( \text{tf} \) and \( \text{idf} \)

```python
def tf_idf(term, docid):
    ...
    # TODO: implement
    return 0
```
Exercise 7: Vector space models

- Implement helper functions:
  - Implement `norm_cosine(docid)`
    - Returns cosine normalization $C_{docid}$ of all weights for a document with id `docid`.
    - $C_{docid} = \forall \text{ term } \in \text{ document with id docid: } \sqrt{\sum \text{tf}_{term,docid}^2}$

```python
def norm_cosine(docid):
    ...

    Compute cosine normalization for docid
    ...

    # TODO: implement
    return 0
```
Exercise 7: Vector space models

- Implement scoring function:

```python
def cosine_score(query, K=10):
    ...
    Compute cosine scores for query `query` and return the top K results according to Figure 6.14.
    ...
    # Initialize arrays so we can use docids as indices
    scores = [0] * (len(documents.keys())+1)
    length = [0] * (len(documents.keys())+1)
    # Precompute length array values -- these are the normalization factors
    for d in documents.keys():
        length[d] = norm_cosine(d)
    # TODO: implement
    # 1. compute scores for each document, store in scores array
    # 2. Normalize using the length array
    # 3. Return top-K: sort by descending score and return first K elements of result.
    return []
```
Exercise 7: Vector space models

- Implement scoring function:
  1. Compute scores for each document and store them in scores array
     - Use tf_idf * idf as scoring function
  2. Normalize using length array
     - Divide the score of each document through its length
  3. Return top-K
     - Use sorted() method

- Check your implementation with the examples
- Questions or things that are unclear?
  - Contact us on Piazza