# Text Technologies: Assessment 1
# A Search Engine for Scientific Documents

Aaron Cronin · s0925570 · The University of Edinburgh

## 1 Introduction

The aim of this assignment is to produce a system for ranking a set of potential documents by how strongly they match a supplied query. The magnitude of this score is irrelevant, the only goal is that each score is proportional to the estimated relevancy of the document to the query. Thus, documents are retrieved by sorting in order of descending relevancy index and returning the first $n$.

## 2 Prior Work

### 2.1 Ranking

**tf · idf** A reliable method for ranking documents is $tf \cdot id$ summation:

$$s(Q,D) = \sum_{w} \mathbf{tf}_{w,Q} \cdot \frac{\mathbf{tf}_{w,D}}{\mathbf{tf}_{w,D} + \frac{k\|D\|}{avg\|D\|}} \cdot \mathbf{log}_2 \frac{\|C\|}{\mathbf{df}_w} \tag{1}$$

`Score s for query Q and document D.`

Scoring by $tf \cdot idf$ has the following properties: Repeating a word in a query causes it to become more important to the ranking process, and matching that word multiple times in the document will cause the score to increase – although multiple repetitions have diminishing returns to combat the effects of *clumping*. Finally, the reward for containing a word increases if the word occurs infrequently throughout the collection of documents.

### 2.2 Tokenisation

The input string was lowercased and split at punctuation. Punctuation was defined as being non-word characters, those matching the regular-expression `\W`.

## 3 Failures

### 3.1 Tokenisation

**Including 3-grams** In order to rank documents higher if they contained words in the same order as the query, 3-grams were included in addition to the basic tokens. The string `'Cheap food in london'` would be tokenised as `['cheap', 'food', 'in', 'london', 'cheap food in', 'food in london']`. This was not effective at increasing the precision of the scientific-document search as English has clearly defined word boundaries and positioning of words is only loosely tied to meaning. In addition, this greatly reduced the performance of the system due to vastly increased number of tokens to iterate through.

## 4 Improvements

### 4.1 Tokenisation

**Split and Merge** Tokenising at nonword boundaries can cause punctuated words such as `pattern-matching` to be split into two tokens that convey little of the original meaning. To combat this, *split-and-merge* appends each original, punctuated, word to the generated tokens. The additional tokens were obtained by splitting on whitespace with the regular-expression `\s` then filtering for tokens that start and end with `\w` but contain `\W`. This addition to tokenisation improved the precision of the system as it ensured that tokens carrying significant meaning were not lost.

**Repeating Titles** Tokens that were originally in title-case such as `Mandlebrot` were appended in their original form to the list of lowercased tokens. This can be considered a naive attempt at *entity recognition*. The duplication resulted in a doubled score component for document-query pairs matching a title-case word, as the token would match once in lowercase form and again in title-case. This alteration in implicit weighting improved the precision of the retrieval system.

**Token Alteration** Tokens corresponding to the digits $[1, 10]$ were replaced with their plain-

English counterparts. This insured that a query including `'Three years'` would match a document containing `'3 years'` and vice-versa, and increased the precision of the system.

**Stopword Removal** *Stopwords* are defined as parts of the English language that convey very little meaning, such as `the, and, they`. The tokens were filtered by comparison with the `nltk` English stopword list. Removing them avoids a document being rewarded for containing words that all text tends to contain. The **idf** portion of the ranking function attempts to reduce their impact but precision was found to improve after these tokens were removed. In addition, the reduced number of tokens improved performance.

**Stemming** *Stemming* is the process of reducing groups of words with the same morphological root to the same token. For example, `swimming`, `swim` and `swimmer` all converge to `swim`. This causes words that convey roughly the same meaning to produce tokens that will match each other. The ability to match originally distinct words greatly improved the search engine's precision. Multiple stemmers were trialled: the 1979 *Porter* stemmer, `nltk`'s *Snowball* stemmer, and the 1990 *Lancaster* stemmer. In experiments, the *Lancaster* stemmer was observed to improve precision the most.

### 4.2 Ranking

**Pseudo Relevance Feedback** Empirical data suggests that documents relevant to a query will match tokens between themselves at rate higher than when compared with an irrelevant document. This observation can be used to improve a user's query by expanding it with tokens obtained from documents estimated to be relevant. The following algorithm was used to implement feedback:

```
In response to a query:
* Rank documents using tf-idf.
* Select 4 most relevant documents.
* Gather their 60 most frequent words.
* Expand the original query with the
new tokens with count 1 for each.
* Rank documents with tf-idf again.
```

Pseudo-relevance feedback had an extremely significant impact on the system, improving the precision greatly at the expense of only a slight reduction in performance - due to the increase in the number of query tokens processed. The constants governing the number of documents and tokens used for expansion were selected as those that performed best during testing.

**Adjusting the Dampening Factor,** $k$ Originally, a value of 2.0 for $k$ was provided. The application was modified to allow $k$ to be provided via a command-line argument and a script was produced to iterate over possible $k$, $[1.0, 5.0]$ to find the maximum precision – as reported by *TREC*. The search response included the following results:

```
1.4 => 0.5620
1.5 => 0.5588
1.6 => 0.5622
1.7 => 0.5741
1.8 => 0.5552
1.9 => 0.5529
2.0 => 0.5382
```

Accordingly, 1.7 was chosen for the value of $k$ as the precision was significantly higher than that of the default.

## 5 Results

As demonstrated in the above log-file section, the system achieved a precision 0.5741 for the provided set of documents and queries, as determined by *TREC* evaluation. This is a significant level of performance and should be regarded as a success when considering the time taken to produce the retrieval system. Progress was assisted greatly by comprehensive unit-testing and profiling to ensure that iteration of design could be rapid and with low risk of introducing error.

## 6 Acknowledgements

The solution includes a backport of `Python 2.7`'s `Counter` class, aiding performance – obtained online. Guidance into which augmentations to attempt was taken from *Search Engines: Information Retrieval in Practice*, which presents a variety of options to improve retrieval alongside evaluation of their merits. Critical discussion of the field of *IR* portrayed in the textbook occurred with student *Jacob Essex*, in addition to discussion of data-structures and topics concerned with the efficient implementation of the exercise in `Python`. The overall direction of the project was not influenced by any collaboration, but the time taken to arrive at a performant solution was decreased by sharing understanding and interpretation of openly available material.