Passage Retrieval with Vector Space and Query-Level Aspect Models

Raymond Wan[1] rwan@kuicr.kyoto-u.ac.jp
Vo Ngoc Anh[2] vo@csse.unimelb.edu.au
Hiroshi Mamitsuka[1] mami@kuicr.kyoto-u.ac.jp

[1] Bioinformatics Center, Institute for Chemical Research, Kyoto University, Gokasho, Uji, 611-0011, Japan
[2] Department of Computer Science and Software Engineering, Faculty of Engineering, University of Melbourne, Victoria, 3010, Australia

Abstract

In 2007, we refined our passage retrieval system from last year's Genomics Track. This year's system is still composed of two parts where the first part identifies paragraphs relevant to the query and the second part uses a probabilistic word-based aspect model to isolate sections of texts within each relevant paragraph. The changes for this year further integrates the two parts and aims to correct some of the problems from last year.

1. Introduction

Query processing consists of two phases. In the first phase, a paragraph retrieval system constructs an index and uses it to output a ranked list of relevant paragraphs using a variant of the vector space model (VSM). Next, a passage extraction system based on the aspect model scores the words in each paragraph to extract a section of relevant text (a passage).

2. Method

The two main components of our systems are described here, and their combination is described in the next section.

2.1 Vector Space Model

The variant of VSM employed in our experiment is the impact-based retrieval approach, in which:

- each paragraph or query is represented as an n-dimensional vector (n is the number of distinct words),
- the similarity degree between a query and a paragraph is reported as an integer instead of a floating point value, and
- the similarity degree is calculated as the scalar product of the respective query and paragraph vectors.

2.2 Aspect Model

We adapt the aspect model for scoring sections of texts which are most relevant to the query. Originally, the aspect model has been used to associate words to documents as a means of information retrieval. We employ one-mode factor analysis (instead of two-mode) where words are associated with each other using probabilitic latent semantic analysis and the Expectation-Maximization algorithm. The aim is to start from a matrix of word co-occurrences (where a word co-occurs with another word if they exist in the same paragraph) and end up with a matrix of scores through the use of k clusters or latent states.

The co-occurrence score for the words i and j where $k = |Z|$ is defined as:

$$c(i,j) = p(i,j) = \sum_{z \in Z} p(i|z)p(j|z)p(z). \qquad (1)$$

We used 100 clusters and a stopping condition of 50 iterations or a change in maximum likelihood of less than 0.0001.

3. TREC 2007 System

Our two models are implemented separately and labeled as a paragraph-level retrieval system and a passage extraction system, respectively. A final post-processing step is used for ranking the passages. The two systems and the tasks performed by each system are summarized in the following table and Figure 1.

System	Task
Paragraph-level retrieval	Indexing
	Querying
Passage extraction	Score derivation
	Passage scoring
Post-processing	Score merging

3.1 System Changes

Compared to our system last year, several changes were applied. One important change is how passages are scored, which will be described shortly. Otherwise, the other significant changes are[1]:

Feature	2007 System
Aspect Model	Query-level
Indexing Level	Paragraph only
External Databases	No
Stemming	"Simple"
Passages per paragraph	1

3.2 Scoring Passages

The score derivation step of passage extraction creates a matrix of scores of size m by n, where m is the number of unique words in the query and n is the number of unique words in the collection.

Each word j in the paragraph is scored against every word i of the query q. If a paragraph word matches the query word exactly, then a score of c_{max} is added to the paragraph word, where c_{max} is the maximum score in the matrix. Otherwise, the matrix is consulted. Thus, the score $s(q,j)$ of a paragraph word against the query is:

$$s(q,j) = \frac{|D|}{f_j} \times (1 + \log f_{d,j}) \sum_{i \in q} \bar{c}(i,j). \qquad (2)$$

The inner summation (denoted as Method 3 in our notebook paper) is given as:

$$\bar{c}(i,j) = \begin{cases} \ln(1 + \frac{|D|}{f_i}) \times c_{\text{max}} & \text{if } i = j \\ \ln(1 + \frac{|D|}{f_i}) \times c(i,j) & \text{otherwise} \end{cases} \qquad (3)$$

In these formulas $f_{d,j}$ is the frequency of j in paragraph d and $|D|$ is the number of paragraphs. The frequency of a query word or a paragraph word in the collection are given as f_i or f_j, respectively.

4. Results

Three runs were submitted and several *sets* of runs were performed after results were released. Only one set of these additional runs are reported in this poster. However, several of its parameters were varied (the number of results from the VSM and the weights for score merging) and are indicated by asterisks in the table below:

ID	VSM results	Scoring method	VSM:AM	
kyoto1	1000	3	100	0
kyoto2	5000	3	0	100
kyoto3	5000	1	50	50
Method-3	*	3	*	*

ID	Document	Aspect	Passage	Passage2
kyoto1	0.1892	0.1208	0.0474	0.0209
kyoto2	0.1191	0.0302	0.0235	0.0054
kyoto3	0.1022	0.0312	0.0204	0.0065
Median	0.1897	0.1311	0.0565	0.0377

The performance of our submitted runs was below the median (grey horizontal lines). After our runs were submitted, we noticed bugs in the "passage scoring" part, as well as how character positions were calculated. After fixing these problems, we performed an additional set of run by varying several parameters. The MAP for one set of results is shown below. The dotted, dashed, and solid lines represent 1,000, 5,000, and 10,000 results from the VSM, respectively.

As the graphs show, our performance is optimal when VSM alone is used for ranking. Nevertheless, we are still below the median, except for our Passage2 scores which are slightly above.

In the future, we plan to further investigate the effect of the parameters in our system.

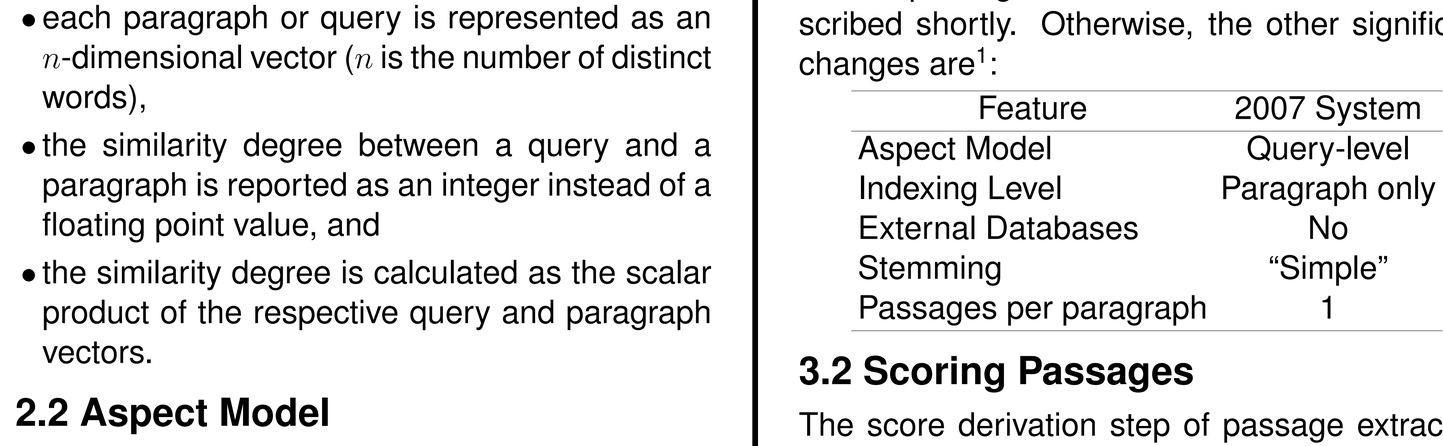

Figure 1: System overview.

[1] "Simple" stemming is a light stemming algorithm which considers only regular endings such as -s, -es, -ed, -ly, and -ing.