Query Segmentation for Web Search

Knut Magne Risvik
Fast Search & Transfer ASA
P.O. Box 4452 Hospitalsløkkan
NO-7418 Trondheim, Norway
kmr@fast.no

Tomasz Mikołajewski
Fast Search & Transfer ASA
Rindermarkt 7
D-80331 München, Germany
tomasz@fast.no

Peter Boros
Fast Search & Transfer ASA
P.O. Box 4452 Hospitalsløkkan
NO-7418 Trondheim, Norway
boros@fast.no

ABSTRACT

This paper describes a query segmentation method for search engines supporting inverse lookup of words and phrases. Data mining in query logs and document corpora is used to produce segment candidates and compute *connexity* measures. Candidates are considered in context of the whole query, and a list of the most likely segmentations is generated, with each segment attributed with a connexity value. For each segmentation a *segmentation score* is computed from connexity values of non-trivial segments, which can be used as a sorting criterion for the segmentations. We also point to a relevancy improvement in query evaluation model by means of *proximity penalty*.

Keywords

web search, query processing, data mining, query segmentation, query evaluation

1. INTRODUCTION

Web Search engines are rapidly emerging into the most important application of the World Wide Web. Several challenges arise when trying to make the web searchable[5].

Search engines like AllTheWeb[1], Google[3] and AltaVista[2] are usually based on a kernel supporting inverse lookups of words and phrases. On top of these kernel features, techniques such as detection of proper phrases (e.g. *new york* → *"new york"*) and removal of stopwords or stop-phrases (e.g *how can i get information about X* → *X*).

These techniques reduce the query into a form that is more likely to express the topic(s) that are asked for, and in a suitable manner for a word-based or phrase-based inverse lookup, and thus improve precision of the search. For instance, a query like *where can i find pizza hut in new york* will most likely have better precision in a word/phrase match intersection when rewritten into a form like *"pizza hut" "new york"*.

Problems with this query rewriting occur when there are ambiguities in phrasing and anti-phrasing alternatives. For instance the query *free computer wallpaper downloads* will be rewritten into *"free computer" wallpaper downloads* if phrasing were done by a leftmost-longest approach (which is a common approach) instead of more natural *free "computer wallpaper" downloads*.

In this paper we will describe how we use data mining in query logs and document corpora to derive information that can be used to segment queries into words and phrases with a number indicating *connexity*.

WWW2003, May 20–24, 2003, Budapest, Hungary.
ACM xxx.

2. MINING LOGS AND CORPORA

Query logs yield a highly interesting data that may be useful for various tasks. Whatever query content specific application (statistical overview, generation of related queries or triggering relevant flash-ins) is considered, a recognition of meaningful and nonsplittable phrases in each multiword query remains one of its core prerequisites.

A sequence $S = w_1 \dots w_n$ (with $2 \leq n \leq 4$) of query tokens (words, numbers, special symbols) is considered a potentially meaningful phrase if the following conditions hold:

1. S is significantly frequent in all resources.
2. S has a 'good' mutual information.

Both above conditions make up a central notion for the segmentation of queries, a *connexity* of a sequence $S = w_1 \dots w_n$, which is defined as a product of the global frequency of the segment $\mathrm{freq}(S)$ and the mutual information I between longest but complete subsequences of S.

$$\mathrm{conn}(S) = \mathrm{freq}(S) \cdot \mathrm{I}(w_1 \dots w_{n-1}, w_2 \dots w_n) \qquad (1)$$

It is assumed that connexity of a single token is equivalent to its frequency i.e. $\mathrm{conn}(w_1) = \mathrm{freq}(w_1)$.

The connexity value presented here is computed from a selected sample of our query logs whose characteristics is: approx. 600 million original query lines and 105 million lines in its normalized frequency list Q_{normfreq}. Most of the operations related to the computation of connexity are carried out on Q_{normfreq} or on its representations.

For the sake of a brief characterization of Q_{normfreq} we split it into subsets according to the number of tokens in a line. Table 1 shows how many lines each subset of Q_{normfreq} consists of and how many new (not seen in other subsets) segments $S = w_1 \dots w_n$ (with $1 \leq n \leq 4$) are estimated in each subset.

tokens in a line	1	2	3	4	5	6
lines $\cdot 10^6$	11.4	30.8	30.7	19.9	8.4	3.4
new $1 \div 4$ segments $\cdot 10^6$	11.4	34.6	41	34.6	13.3	9.1

Table 1: Token and segment numbers in Q_{normfreq}

The total number of $1 \div 4$-token segments that appear in Q_{normfreq} is estimated to $|S_{1234}| \simeq 144 \cdot 10^6$. A manageable subset $S'_{1234} \subset S_{1234}$ must be chosen to satisfy feasibility conditions on a single query processing host i.e. search node (as defined in [5]):

- Query processing speed ≥ 5000 queries/second,
- disk accesses excluded – full database that maps the segments S'_{1234} to their connexities must be kept in main memory,

- assuming 256MB RAM on a search node, the maximal size of the database must be less than 200 MB.

We have applied two quite rough rules-of-thumb for scaling down the number of segments $|S'_{1234}|$ by selecting only:

- Segments with the frequency ≥ 5 and
- segments with the connexity value over a fix threshold that completes $|S'_{1234}|$ to $12.5 \cdot 10^6$.

We have chosen to represent the database of segments and their connexities with means of finite state devices as defined e.g. in [4]. Each segment is represented as an entry in the device and combined with a perfect hash method to achieve the mapping to its connexity. This twofold (segment strings + connexity values) representation consist of a 140MB string representation part and 50MB connexity values part which fulfils the feasibility condition for size. Thanks to a very efficient lookup ($\sim$ 0.5M lookups/second), it is possible to realize 25000 segmentations/second of average queries. The time may vary depending on the complexity of segmentation procedure which we shall discuss in the next section. The compact segment representation has an important disadvantage for updating of the database. The database is always constructed as a whole structure with no possibility of updates of a single entry. However, the whole database creation time from Q_{normfreq} i.e. collecting of segments, scaling down to S'_{1234} and its finite state representation is carried out within 3 hours on a single host. This allows for daily updates.

All the above procedures have been applied to frequency list of query logs Q_{normfreq} and its derivatives. Another interesting source for fine tuning the connexity values of segments are textual parts of web documents – the web corpus. The connexity of a segment S can be modified by the mutual information values of its usage in a document as opposed to its querying. However, our current experience is to give more weight to the second perspective i.e. user's expectation of segments expressed in query logs.

3. QUERY SEGMENTATION

The database of segments and connexities may be applied to an arbitrary text, preferably query, for splitting it into segments according to a segmentation procedure. The procedure matches all possible subsequences of the given tokenized and normalized query q in the segments database and assigns connexity value to each matched segment (segments valuation). Recognized nonoverlapping segments make up a single segmentation $\mathrm{segm}(q)$. The segmentation procedure computes $\mathrm{score}(\mathrm{segm}(q))$ i.e. a segmentation score for each segmentation. For a query q:

```
msdn library visual studio
```

the computed $\mathrm{segm}(q)$ are as follows (sorted by $\mathrm{score}(\mathrm{segm}(q))$, connexities of segments in square brackets):

```
34259:  (msdn library)[5110] (visual studio)[29149]
29149:  msdn[47658] library[209682] (visual studio)[29149]
5110:  (msdn library)[5110] visual[23873] studio[53622]
41:  (msdn library visual studio)[41]
7:  msdn[47658] (library visual studio)[7]
0:  msdn[47658] library[209682] visual[23873] studio[53622]
```

The function $\mathrm{score}(\mathrm{segm}(q))$ computes the sum of connexities for segments of the length ≥ 2 in the above procedure. This scoring gives us a sorting criterion and a measure for 'how far from each other' the particular segmentation alternatives are. In addition to that we may keep the function parametrizable in order to make the scoring depend on e.g. the query contents related aspects.

The problem of query mining and segmentation seems to be self-referential. The better we are able to segment the queries, the more reliable are the connexity values we collect for the resulting segments. This leads to the improved mining of logs via iteration of segmentation procedure. We can assume that most multiword queries should allow to get complete segmentation i.e. the whole query is covered by segments longer that one token. This feature of the query is used for boosting the connexity values for recognized segments which unifies two approaches:

- Static connexity computation and
- integration of dynamically selected segments from the best of all segmentations only.

The essential goal of query segmentation is how the segments' connexity can be integrated into the general framework of query answering in the search engine. This goal becomes easier to solve if we look at segments' connexity in a query as a *counterpart of proximity* value in documents. Proximity of two words w_1 and w_2, $\mathrm{proximity}(w_1, w_2)$, can be described as a function of the number of tokens between them. The function has typically higher values for lower distances between w_1 and w_2. Connexity values can be used to modify the shape of the $\mathrm{proximity}$ function, e.g. by means of a *proximity penalty* for larger w_1 to w_2 distances. In the following segmentations

```
3458:  (msdn universal)[3458] subscription[10260]
2932:  (msdn universal subscription)[2932]
24:  msdn[47658] (universal subscription)[24]
```

we see

$\mathrm{conn}(\texttt{msdn},\texttt{universal}) \gg \mathrm{conn}(\texttt{universal},\texttt{subscription})$.

This can be interpreted as higher proximity penalty for occurrences of `msdn` and `universal` as opposed to lower proximity penalty for a cooccurrence of `universal` and `subscription`. In general, the higher the static $\mathrm{conn}(w_1 \ldots w_n)$, the higher proximity penalty for the coocurrence of $\{w_1, \ldots, w_n\}$.

4. CONCLUSIONS AND OUTLOOK

We presented some insights into the query segmentation methodology. The tests are based on a sample query logs of our search engine [1]. We sketched our technology and gave exemplary analysis traces and numbers. Our current experience lets us conclude that proper segmentation of queries, expecially finding in them the core of the user's inquiry and its weight/importance (i.e. connexity), brings a considerable quality gain for search. If integrated into an appropriate ranking formula, query segmentation and scoring delivers a new valuable block of information into query answering pipeline.

In general, one of the future issuses is to integrate further means for modifying connexity values depending on various factors like document's features where the segment occurs e.g. its quality, popularity etc.

5. REFERENCES

[1] Alltheweb. http://www.alltheweb.com.

[2] Altavista. http://www.altavista.com/.

[3] Google. http://www.google.com/.

[4] M. Mohri. Minimization algorithms for sequential transducers. *Theoretical Computer Science*, 234(1–2):177–201, 2000.

[5] K. M. Risvik and R. Michelsen. Search engines and web dynamics. *Computer Networks (Amsterdam, Netherlands: 1999)*, 39(3):289–302, 2002.