Chapter 3
Query Segmentation and Tagging

Xuanhui Wang

Abstract Query tagging is an important step for query understanding. It applies traditional natural language processing techniques on query strings. Specific challenges are raised due to the shortness of query strings. In this chapter, we describe techniques proposed in the existing literature on how to achieve meaningful query tagging in the following areas: query segmentation, query syntactic tagging, and query semantic tagging.

3.1 Introduction

Query tagging is an important step for query understanding. It is a process that works with query strings more closely based on Natural Language Processing (NLP) techniques [19]. Traditionally, NLP techniques are developed for documents with well-formed sentences and can be used for Information Retrieval (IR). For example, phases in documents can be identified and used in document index [37]. Thus, it is important to segment queries to match phrases in documents to boost retrieval accuracy. Furthermore, Part-Of-Speech (POS) tags and linguistic structures carry meaningful information to match queries and documents. They are also important for search engines to improve result relevance. However, the keyword-based queries are usually short and lack of sentence structures. It raises challenges to apply NLP on queries directly. In this chapter, we describe techniques proposed in the existing literature on how to overcome these challenges to achieve meaningful query tagging in the following areas: query segmentation, query syntactic tagging, and query semantic tagging.

X. Wang (✉)
Google Research, Mountain View, CA, USA
e-mail: xuanhui@gmail.com

Y. Chang, H. Deng (eds.), *Query Understanding for Search Engines*,
The Information Retrieval Series 46, https://doi.org/10.1007/978-3-030-58334-7_3

3.2 Query Segmentation

Search engines usually provide a search box as the user interface and a few keywords can be used to search web pages to satisfy users' information needs. Due to the simplicity for users to formulate search queries, such a user interface gains popularity and becomes a standard for search engines. However, queries formulated in this interface are generally not complete natural language sentences, but consist of a bunch of keywords. Thus it becomes harder to apply NLP techniques directly on queries. For example, a sentence like "Where can I find Pizza Hut in New York" is likely tagged well by NLP tools than a query "pizza hut new york."

Query segmentation is one of the first steps towards query understanding. It does not involve heavy NLP processes such as Part-Of-Speech (POS) tagging or Named Entity Recognition (NER). Its goal is to split a query string into a few segments. The basic bag-of-words (BOW) model can be thought as segmenting queries based on individual words. Such an approach is simple but can be less meaningful. For Chinese language, most of the individual words have little meaning by themselves and the meaning of a sentence is carried by a sequence of words. However, there are no natural boundaries such as spaces in Chinese language and segmentation is a necessary step for Chinese documents and queries [24, 28]. For English language, spaces are presented inside sentences and individual words obtained in the BOW model are more meaningful compared with Chinese language. However, the BOW model can still be less effective because the meaning of a phrase can be totally different from its individual words. For example, knowing that "new york " is a city name and treating them as a whole is better than treating them as two individual words "new" and "york."

An advanced operator provided by many search engines is the double quotation. A user can enclose several words together by double quotation to mandate that they appear together as an inseparable sequence in retrieved documents. Such an operator is usually used by skilled users and may not be known widely. It also requires additional efforts from end users. For example, when users search for unfamiliar topics, they may not know where to put the double quotation. A search engine that can automatically split a query into meaningful segments is highly likely to improve its overall user satisfaction.

To improve retrieval accuracy and search engine utility, it is necessary to go beyond the BOW model. At a minimum, it is beneficial to know whether some words comprise an entity like an organization name, which makes it possible to enforce word proximity and ordering constraints on document matching, among other things. In this section, we discuss different query segmentation techniques.

3.2.1 Problem Formulation

The problem of query segmentation is to find boundaries to segment queries into a list of semantic blocks. In general, given a query of n words, $x_1 \ldots x_n$, query segmentation is to find boundaries of these n words, $[x_1 \ldots x_{s_1}][x_{s_1+1} \ldots x_{s_2}] \ldots [\ldots x_n]$, with each segment as a well-defined "concept." For example, given query "new york times subscription," a good segmentation is "[new york times] [subscription]," but not "[new york] [times subscription]" because "times subscription" is less meaningful.

For an n-word query, there are $n-1$ possible places for boundaries and thus a total of 2^{n-1} possible segmentations. The goal of query segmentation is to find the most meaningful segmentations, e.g., "[new york times] [subscription]" in the example above. In many cases, there are several possible segmentations for a query that are equally meaningful due to ambiguity. For example, the "two man power saw" example used in [4] can have four different interpretations from Google returned documents and these lead to the following valid segmentations: "[two man power saw]," "[two man] [power saw]," [two] [man] [power saw], and "[two] [man power] [saw]." Thus the problem of query segmentation is usually formulated to find a few good ones.

Due to short length of queries, external resources are commonly used in query segmentation, including web corpora [26, 30], query logs [22], click-through data [18], Wikipedia titles [13, 30], etc. The methods proposed in this area can be classified as heuristic-based, supervised learning, and unsupervised learning approaches. In the following, we use "segment" to represent a semantic segmented block and "segmentation" to represent a valid split of a query with non-overlapping segments.

3.2.2 Heuristic-Based Approaches

Heuristic-based approaches are based on statistics obtained from external resources. They do not rely on any sophisticated learning and have the following two types: one type is to decide whether to put a boundary between two adjacent words and the other is to quantify the connectedness of a segment and break queries by maximizing the overall connectedness.

3.2.2.1 Pointwise Mutual Information

Given an n-word query, the most direct way for query segmentation is to decide whether a boundary should be put at the $n-1$ places. The Pointwise Mutual Information (PMI) approach is to make this decision locally based on the surrounding words. For example, given a query "free computer wallpaper downloads," we would like to

decide whether to put a break between "free" and "computer," between "computer" and "wallpaper," etc. More formally, we would like to decide a break at the place between word x_i and x_{i+1} for $1 \leq i < n$. Intuitively, if two words always appear together in a corpus, it is better to not put a break but keep them in a single segment.

PMI is an information-theoretic measure [4, 8] on term associations. Given any two objects u and v, their PMI is defined as:

$$PMI(u, v) = \log \frac{Pr(u, v)}{Pr(u) \cdot Pr(v)}, \tag{3.1}$$

where $Pr(u, v)$ is the probability of observing u and v appearing together, $Pr(u)$ and $Pr(v)$ are the probability of observing u and v in the given corpus, respectively.

Let us assume that we have a web corpus that has been tokenized into word sequences. We can count the raw frequency denoted as $\#(x_i, x_{i+1})$, $\#(x_i)$, and $\#(x_{i+1})$ from the corpus. Let N denote the total number of words in this corpus. Then we have

$$\begin{aligned} Pr(x_i, x_{i+1}) &= \frac{\#(x_i, x_{i+1})}{N} \\ Pr(x_i) &= \frac{\#(x_i)}{N} \\ Pr(x_{i+1}) &= \frac{\#(x_{i+1})}{N}. \end{aligned} \tag{3.2}$$

PMI between two adjacent words can be used for query segmentation by setting a threshold κ. Apparently, how to choose the parameter κ needs some validation data. For example, Jones et al. [14] used a threshold $\kappa = \log 8$ on Yahoo! search logs. They reported that the PMI method was quite effective in their experiments.

3.2.2.2 Connexity

The above PMI method only concerns about two adjacent words. It can also be used in n-gram level to measure the connectedness of a segment. Risvik et al. [26] defined a measure called *connexity* based on the following properties for a segment s:

- s is significantly frequent in all resources.
- s has a "good" mutual information.

In their approach, a segment is essentially an n-gram where $2 \leq n \leq 4$. We denote it by $s = x_i \ldots x_j$ and its connexity is defined as

$$connexity(s) = \begin{cases} frequency(s) \cdot PMI(x_i \ldots x_{j-1}, x_{i+1} \ldots x_j) & \text{if } |s| \geq 2 \\ frequency(s) & \text{otherwise}, \end{cases} \tag{3.3}$$

Table 3.1 Query segmentations and their connexity scores for an example query. Segmentations are sorted by the aggregated connexity scores that are computed as summation over all segment scores, as shown by scores in parentheses. The single word segments contribute 0 to the aggregated scores

Connexity	Query: msdn library visual studio
34,259	[msdn library] (5110) [visual studio] (29149)
29,149	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)

where the mutual information is on the longest but complete subsequences and can be computed similarly as for words in Eqs. (3.1) and (3.2).

Based on query logs and web corpora, the connexity of an arbitrary segment s can be computed. The number of possible s becomes exponentially large as n goes larger. In practice, n is capped to a number such as 4 in [26]. Segments with higher connexity are more likely to be coherent concepts. To make the number of segments manageable, thresholds were used on frequency as a pre-processing and thresholds on connexity as post-processing filtering in [26].

The connexity was computed from web corpora offline and stored as a lookup table used for query segmentation. On a high level, the non-overlapping segments can be identified as segmentation candidates to be scored. An example used by Risvik et al. [26] is shown in Table 3.1. The aggregated scores are computed as the summation of the connexity scores over all segments that have at least 2 words. While a brute force way is to enumerate all possible segmentations, the top segmentations can be found based on a dynamic programming approach similar to the one presented in Sect. 3.2.4.1.

3.2.2.3 Naive Segmentation

The connexity measures how coherent a segment is. Another method proposed by Hagen et al. [13] is based on simple statistics of segment frequency and length only. They call this method "Naive Segmentation." The score of a segment s in this method is defined as

$$Score(s) = \begin{cases} 0 & \text{if } |s| = 1 \\ |s|^{|s|} \cdot frequency(s) & \text{if } frequency(s) > 0 \text{ for } |s| \geq 2 \\ -1 & \text{if } frequency(s) = 0 \text{ for } |s| \geq 2. \end{cases} \quad (3.4)$$

A valid segmentation S contains a list of segments that completely cover q without any overlapping. The score of a segmentation is defined as

$$Score(S) = \begin{cases} -1 & \text{if } \exists s \in S, Score(s) < 0 \\ \sum_{s \in S, |s| \geq 2} Score(s) & \text{else .} \end{cases} \tag{3.5}$$

A single word segment has a 0 score and this is implicit in Eq. (3.5). Such a method is purely hand-crafted but was shown to be effective in [13]. The exponential component on the segment length boosts scores of longer segments. It is justified empirically by the connection with the power-law distribution of n-gram frequencies for n-grams that are longer than bigrams. Conversely, the exponential component still favors bigrams compared with the empirical bigram frequency and directly using empirical frequencies for all n-grams dropped by the segmentation accuracy significantly. Favoring bigrams was also observed in the human-generated segmentations [13]. In addition, they further extended this method to leverage Wikipedia titles in the following way:

$$weight(s) = \begin{cases} |s| + \max_{s' \in s, |s'|=2} frequency(s') & \text{if } s \text{ is a title in Wikipedia} \\ frequency(s) & \text{otherwise.} \end{cases} \tag{3.6}$$

And the weight is used to compute the $Score(S)$ for segmentations as follows:

$$Score(S) = \begin{cases} -1 & \text{if } \exists s \in S, weight(s) = 0, |s| \geq 2 \\ \sum_{s \in S, |s| \geq 2} |s| \cdot weight(s) & \text{else .} \end{cases} \tag{3.7}$$

Since the $Score(S)$ is a summation of its components, the top segmentations can also be found through a dynamic programming similar to the one in Sect. 3.2.4.1.

3.2.2.4 Summary

There are two types of heuristic-based approaches. The PMI method is to measure how easy it is to insert a break between two adjacent words and is of the first type. This can be done efficiently. Along the same line, Zhang et al. [38] proposed the eigenspace similarity as a similar measure as PMI and the method belongs to the first type as well. The connexity and naive segmentation methods belong to the second type. In this type, a score that measures coherence of a segment is defined based on a few factors such as segment frequency, length, mutual information between the longest but complete subsequences, and the appearance of the segment in Wikipedia titles. Segment scores are then used to define scores for segmentations that can be used to select the top segmentations. There are a few additional methods that

belong to the second type. For example, Mishra et al. [22] proposed a way to define segment scores to identify the so-called Multi-Word Expression (MWE) and then score segmentations similarly.

3.2.3 Supervised Learning Approaches

Query segmentation based on supervised learning approaches was introduced by Bergsma and Wang [4]. In the supervised learning setting, segmentation is formulated as a function that takes a query q as input and outputs a segmentation $\mathbf{y}$:

$$S : q \rightarrow \mathbf{y}, \tag{3.8}$$

where $\mathbf{y}$ is a $n-1$ dimensional vector with binary values and $y_i = 1$ means that there is break between word x_i and x_{i+1}. Such a setting has a similar flavor as the PMI approach. The difference is that supervised learning approaches learn segmentation function S from training data, while the PMI approach is based on hand-crafted heuristics.

The training data for supervised learning consists of a collection of pairs $\{(q, \mathbf{y})\}$. A set of features $\Phi(q, \mathbf{y})$ can be defined for each training instance. The score of a segmentation $\mathbf{y}$ is

$$Score(q, \mathbf{y}; \mathbf{w}) = \mathbf{w} \cdot \Phi(q, \mathbf{y}). \tag{3.9}$$

The training is thus to find the best $\mathbf{w}^*$ on the training data so that for q,

$$Score(q, \mathbf{y}; \mathbf{w}^*) \geq Score(q, \mathbf{z}; \mathbf{w}^*), \forall \mathbf{z} \neq \mathbf{y}. \tag{3.10}$$

Such a $\mathbf{w}^*$ is usually not existent and slack variables can be used in the Support Vector Machine (SVM) setting. After the parameter $\mathbf{w}^*$ is learnt, the segmentation function gives the output for an input q:

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y}} Score(q, \mathbf{y}; \mathbf{w}^*). \tag{3.11}$$

The above formulation can be solved by structured classifiers [32] where all the $n-1$ decisions are jointly made. However, in reality, a simpler classification framework, where each binary decision was made for each position i for $1 \leq i < n$ based on its context, was shown to be not only efficient, but also effective. Specifically, at each position i, the following context was considered:

$$\{\ldots, x_{L_2}, x_{L_1}, x_{L_0}, x_{R_0}, x_{R_1}, x_{R_2}, \ldots\}, \tag{3.12}$$

where L_i and R_i are the indexes on the left and right side of the position i, respectively. Based on the context, a set of features were defined in [4]:

- Decision-boundary features. The set of features in this category concern about x_{L_0} and x_{R_0} from indicator functions such as is_free (e.g., 1 is fired for this feature if x_{L_0} is word "free" and 0 otherwise.), to POS tags (e.g., DT JJ is fired as a feature when POS of x_{L_0} is DT and POS of x_{R_0} is JJ.). In additional, the PMI between (x_{L_0}, x_{R_0}) in Eq. (3.1) or its raw counts in Eq. (3.2) were also used as features.
- Context features. This set of features concern about n-grams in the context. For x_{L_1} and x_{R_1}, they collected token-level features using indicator functions, POS tags on bigram $[x_{L_1}x_{L_0}]$ and $[x_{R_0}x_{R_1}]$, counts on trigram $[x_{L_1}x_{L_0}x_{R_0}]$ and $[x_{L_0}x_{R_0}x_{R_1}]$. If context tokens were not available at this position, a feature was fired to indicate this. Furthermore, if tokens x_{L_2} and x_{R_2} were available, token-level, bigram, trigram, and fourgram counts from web or a query database were also included.
- Dependency features. A feature in this category is motivated by the work in noun phrase parsing to capture whether x_{L_0} is more likely to modify a later token such as x_{R_1}. For an example of "female bus driver," we might not wish to segment "female bus" because "female" has a much stronger association with "driver" than with "bus." Thus as features, the pairwise counts between x_{L_0} and x_{R_1} and then x_{L_1} and x_{R_0} were included. Features from longer range dependencies did not improve performance in their evaluation.

3.2.3.1 Summary

There are other supervised learning methods. Kale et al. [15] formulated query segmentation as the same classification problem as above. However, they did not use hand-crafted features. Rather, they directly use the low-dimensional word embedding vectors that were pre-trained from query logs. Yu et al. [36] proposed a query segmentation method based on Conditional Random Fields (CRF).

The advantage of supervised learning approaches lies in that they can incorporate any information as features and then learn a function to combine them. For example, the raw counts used in the PMI approach can be used as features. Their combination formula is automatically determined from training data, while the formula is pre-defined in the heuristic-based approach. Supervised learning approaches give better segmentation accuracy; however, the prerequisite is training data that is usually manually segmented by human annotators.

3.2.4 Unsupervised Learning Approaches

Unsupervised learning approaches do not rely on human annotated training data and are more sophisticated than the heuristic-based approaches. A representative work of this line is by Tan and Peng [30]. In their work, a generative model was proposed for query segmentation in which a query is generated by repeatedly sampling well-formed segments (called "concepts") in a probabilistic manner.

Formally, let $P(s)$ be the probability of a segment s and S be a segmentation for a query. The likelihood of S is

$$P(S) = P(s_1)P(s_2|s_1)\ldots. \tag{3.13}$$

Under the Independent and Identically Distributed (IID) assumption for all s, we have a unigram-like model

$$P(S) = \prod_i P(s_i) \tag{3.14}$$

since $P(s_i|s_1, \ldots) = P(s_i)$. Assume that we know $P(s)$ for any segment s, $P(S)$ can be used to select the top segmentations for a query. Given a query, we can enumerate all different segmentations and score them. However, this is not feasible for longer queries given that there are 2^{n-1} segmentations for an n-word query. An efficient dynamic programming is presented in the following section.

3.2.4.1 Dynamic Programming for Top Segmentations

In practice, segmentation enumeration is infeasible except for short queries. However, the IID assumption of the unigram model makes it possible to use dynamic programming to compute the top k segmentations [30]. The algorithm is summarized in Algorithm 1. In this algorithm, for any i, the best k segmentations for partial query $x_1 \ldots x_i$ are stored in $B[i]$. $B[n]$ stores the best k segmentations for the n-word query and is constructed by comparing the options when the last break in the query is placed at different positions of $[1..n-1]$, together with the default segmentation that treats the whole query as the single segment. The complexity of this algorithm is $O(n \cdot k \cdot m \cdot \log(k \cdot m))$, where n is the query length, m is the maximum allowed segment length, and k is the number of best segmentations to keep. It is clear that $m \leq n$. Also, m is implicit in the algorithm and is related to the variable j. To be more accurate, j should range in $[i-m, i-1]$ in Algorithm 1 because a segment longer than m has $P(s) = 0$.

Such a dynamic programming is generic and can be easily adapted to the connexity and naive segmentation methods in Sect. 3.2.2 by changing the computed scores that are stored in $B[i]$.

Algorithm 1 Find top segmentations

Input: query $x_1 ... x_n$, segment probability distribution $P(s)$.
Output: top k segmentations for query.

Let $B[i]$ be the top k segmentations for the partial query $x_1 ... x_i$.
For $b \in B[i]$,
$b.segs$: list of segments for the partial query.
$b.prob$: likelihood of $segs$ for the partial query.
for all $i \in [1..n]$ **do**
  Let $s = x_1 ... x_i$
  if $P(s) > 0$ **then**
    Let $new.segs = \{s\}, new.prob = P(s)$
    $B[i] = \{new\}$
  end if
  for all $j \in [1..i-1]$ **do**
    for all $b \in B[j]$ **do**
      Let $s = x_j .. x_i$
      if $P(s) > 0$ **then**
        Let $new.segs = b.segs \cup \{s\}, new.prob = b.prob \times P(s)$
        $B[i] = B[i] \cup \{new\}$
      end if
    end for
  end for
  Sort $b \in B[i]$ by $b.prob$ and truncate $B[i]$ to size k
end for
return B[n]

3.2.4.2 Parameter Estimation

The main question is how to estimate $P(s)$. This can be done based on some of the heuristic-based approaches in Sect. 3.2.2 or just raw frequencies of n-grams. Though raw frequencies for longer n-grams (e.g., $n > 5$) are very sparse and hard to compute, Tan and Peng [30] proposed a way to estimate lower bounds of raw frequencies for any n-gram and that can be used to estimate $P(s)$. However, as noted in [13], the lower bound can become loose and regress to 0. This effectively excludes too long n-grams from being segments. In general, only n-grams up to a cap (e.g., 5) are considered as potential segments.

One drawback of using raw frequency is that such a method may favor partial segmentation. For example, the frequency for n-gram "york times" is larger than or equal to the frequency of "new york times." Thus $P(\text{york times}) \geq P(\text{new york times})$. However, "york times" is unlikely to appear alone; $P(\text{york times})$ should be very small.

Tan and Peng [30] proposed an expectation–maximization (EM) algorithm for the parameter estimation. The EM algorithm is an iterative procedure that starts with a random guess of parameters and refines them in each iteration. The E-step can be thought as automatically segmenting the texts in a probabilistic manner using the current set of estimated parameter values. Then in the M-step, a new set of parameter values are calculated to maximize the complete likelihood of the data

which is augmented with segmentation information. The two steps alternate until a termination condition is reached.

The EM algorithm can be applied to any collection of texts to give an estimation of $P(s)$. This is infeasible to the web corpus. In [30], a query-dependent pseudo corpus was constructed for every query by counting all the matched n-grams of the query in a corpus:

$$D = \{(x, c(x))|x \in q\}. \tag{3.15}$$

D is enhanced with a dummy n-gram z with count $c(z) = N - \sum_i c(x_i)|x_i|$, where N is the corpus length. Note the difference between n-grams and segments in this context.

Given D, EM uses the minimum description length principle to find the optimal parameters $P(s)$. We use a shorthand θ to represent all parameters. Given the current parameter θ, the description length of an n-gram x is $-\log P(x|\theta)$ and

$$P(x|\theta) = \sum_{S_x} P(S_x|\theta), \tag{3.16}$$

where S_x varies over all possible segmentations of x. All S_x's are the hidden variables in the EM algorithm. The description length of D is

$$-\log P(D|\theta) = -\sum_{x \in q} c(x) \cdot \log P(x|\theta) - c(z) \log(1 - \sum_{x \in q} P(x|\theta)). \tag{3.17}$$

EM algorithm is to find the optimal $\hat{\theta}$:

$$\hat{\theta} = \arg\min(-\log P(D|\theta)) = \arg\max \log P(D|\theta) = \arg\max P(D|\theta). \tag{3.18}$$

The concrete EM algorithm used to find a local optimal θ is the variant Baum–Welch algorithm from [9]. In the E-step, it uses a dynamic programming called the *forward–backward* algorithm that can efficiently compute the probability of forming a segment $[x_i, \ldots x_j]$ between the i-th and j-th positions in an n-gram x, denoted as $P([x_i, \ldots x_j]|x)$. Concretely, let the *forward probability* $\alpha_i(x)$ be the probability of generating any complete segmentation such that the first i words are $x_1 \ldots x_i$. Then $\alpha_0(x) = 1$ and

$$\alpha_i(x) = \sum_{j=0}^{i-1} \alpha_j(x) P(s = [x_{j+1} \ldots x_i]). \tag{3.19}$$

Similarly, let the *backward probability* $\beta_i(x)$ be the probability of generating any complete segmentation such that the last i words are $x_{n-i+1} \ldots x_n$. Then

$\beta_n(x) = 1$ and

$$\beta_i(x) = \sum_{j=i+1}^{n} \beta_j(x) P(s = [x_i \dots x_{j-1}]). \tag{3.20}$$

Notice that $P(x|\theta) = \alpha_n(x) = \beta_0(x)$ and

$$P([x_i, \dots x_j]|x) = \frac{\alpha_{i-1}(x) P(s = [x_i \dots x_j]) \beta_{j+1}(x)}{P(x|\theta)}. \tag{3.21}$$

Then the M-step can reestimate

$$P(s) \propto \sum_{x \in D} c(x) \sum_{i=0}^{|x|} \sum_{j=i}^{|x|} P([x_i \dots x_j]|x) \mathbf{I}\{s = [x_i \dots x_j]\}, \tag{3.22}$$

where $\mathbf{I}$ is an indicator function. The forward–backward algorithm is more efficient than directly estimating $P(S_x|x)$ for all S_x and x, given that only $\alpha_i(x)$ and $\beta_i(x)$ are needed to be computed.

The EM algorithm can be extended to handle Maximum A Posteriori (MAP) estimation with a prior $P(\theta)$. Then the learning is to find

$$\hat{\theta} = \arg\max P(D|\theta) P(\theta) = \arg\max(\log P(D|\theta) + \log P(\theta)), \tag{3.23}$$

where

$$\log P(\theta) = \gamma \sum_{s: P(s|\theta) > 0} \log P(s|\theta) \tag{3.24}$$

and γ is a hyper-parameter to the model. Techniques like lexicon deletion proposed in [9] are used in [30] when the objective can be increased if a segment s is deleted from the parameters $P(s)$.

3.2.4.3 External Sources

The main problem of a purely unsupervised approach is that it only tries to optimize the statistical aspects of the concepts; there is no linguistic consideration involved to guarantee that the output concepts are well-formed. For example, the query "history of the web search engine" favors the "[history of the] [web search engine]." This is because "history of the" is a relatively frequent pattern in the corpus. To address this issue, external resources like Wikipedia titles and anchor texts/aliases were used as well-formed concepts to address the problem in the previous example.

In [30], the above EM algorithm is extended to incorporate Wikipedia as a regularization term

$$\lambda \sum_{s \in Wikipedia} count(s) \log P(s), \tag{3.25}$$

where the summation is over all the Wikipedia titles or anchors and $count(s)$ is the count of s in titles or anchors. Technically, such a variant belongs to semi-supervised learning.

3.2.4.4 Summary

Unsupervised learning approaches have a unique advantage that no labeled data is needed. Existing approaches mainly use EM as their main algorithms. For example, Peng et al. [24] used it on Chinese language segmentation. Li et al. [18] leveraged clicked documents to bias the estimation of query segmentation towards bigrams appeared in clicked documents. These demonstrated the flexibility of unsupervised learning approaches in different applications and the ability to incorporate different external resources.

3.2.5 Applications

Query segmentation can be used to improve retrieval accuracy in the n-gram model or term-dependency model [21]. In particular, Bendersky et al. [2] compared using simple n-grams or query segmentation in the term-dependency model and found that query segmentation can reduce the number of term-dependency relations. It in turn reduced the query latency while still maintaining the retrieval effectiveness.

Query segmentation provides phrases that can be used as units in IR models. Wu et al. [34] combined the BOW model and query phrase model together to derive ranking features. A learning-to-rank model was trained based on the enlarged set of features. Such a model was shown to be able to improve relevance ranking.

It should be noted that most of the above methods assume flat segmentations for queries. More advanced nested segmentation was proposed to segment queries into tree structures [27], where a hierarchical segmentation was built up by recursively merging smaller segments to bigger ones for a query. Such a tree structure was used to define a proximity factor in document scoring. We direct interested readers to [27] for more details.

3.3 Query Syntactic Tagging

Syntactic analysis is usually conducted over complete sentences in NLP. Its goal is to understand a sentence's grammatical constituents, POS of words, and their syntactic relations. The task of query syntactic tagging is to apply NLP techniques to search queries and is depicted by the examples in Fig. 3.1 that was used by Sun et al. [29]. In this figure, we have 3 queries, the POS tags of each word in the queries and the syntactic relations among words (e.g., head-modifier relations in noun phrases). Specifically, for query "cover iphone 6 plus," the relation tells us that the head token is "cover," indicating its intent is to shop for iphone covers, instead of iphones; for query "distance earth moon," the head is "distance," indicating its intent is to find the distance between the earth and the moon; for query "faucet adapter female," the intent is to find a female faucet adapter. Such knowledge is crucial for search engines to show relevant pages because correctly identifying the head of the queries (e.g., covers instead of iphones) in the examples can boost pages with matched topics [29].

Syntactic analysis of search queries is important for a variety of tasks such as better query refinement and improved query-document matching [1]. However, search queries are different from well-formed sentences in the following aspects. First, search queries are short and have only keywords. Second, capitalization is in general missing. Third, word order in a query is fairly free. All these are important sentence characteristics that syntactic parsing relies on. Thus significant challenges arise when applying syntactic parsing NLP techniques on search queries. In this section, we review how different methods proposed in the literature overcome these challenges for query syntactic tagging.

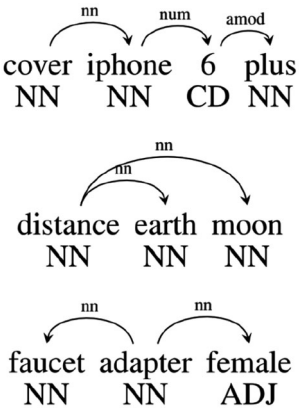

Fig. 3.1 Examples of query syntactic tagging used by Sun et al. [29]. For each query example, the POS tag for each word is shown below the word. The syntactic relation is denoted by arrows that point from heads to modifiers. Search queries are not well-formed, compared with natural language sentences

3.3.1 Syntactic Structures for Search Queries

Barr et al. [1] sampled a few thousand queries from the Yahoo! search engine logs in August 2006 and asked human annotators to label POS tags of these queries. They compute the statistics of the tags and the results are shown in Table 3.2. As shown in this table, they are very different from the Brown or Penn tag sets given that many standard POS tags are extremely sparse in web search queries. For example, there are about 90 tags in the Brown tag set, but there are only 19 unique classes for Web search queries. There are 35 types of verbs and 15 types of determiners in the Brown tag set, but there is only a single label of verbs that accounts for 2.35% of the tags and a single determiner that accounts for 0.7% in the web queries. Furthermore, the most common tag in Table 3.2 is proper-noun, which constitutes 40% of all query terms, and proper nouns and nouns together constitute 71% of query terms. By contrast, in the Brown corpus, the most common tag is noun and it constitutes about 13% of terms.

Barr et al. [1] also showed examples of different ways of capitalization used in web search queries and reported that capitalization in queries was inconsistent. On a sample of 290, 122 queries, only 16.8% contained some capitalization, while 3.9% of them are all capitalized. Though capitalization is an important clue to identify proper nouns in NLP, it becomes noisy to use when tagging queries.

Table 3.2 Tags used for labeling POS of words in web search queries from [1]. The counts are for the number of tokens appearing in the sampled queries

POS tag	Example	Count	Percentage
Proper-noun	texas	3384	40.2%
Noun	pictures	2601	30.9%
Adjective	big	599	7.1%
URI	ebay.com	495	5.9%
Preposition	in	310	3.7%
Unknown	y	208	2.5%
Verb	get	198	2.4%
Other	conference06-07	174	2.1%
Comma	,	72	0.9%
Gerund	running	69	0.8%
Number	473	67	0.8%
Conjunction	and	65	0.8%
Determiner	the	56	0.7%
Pronoun	she	53	0.6%
Adverb	quickly	28	0.3%
Possessive	?s	19	0.2%
Symbol	(	18	0.2%
Sentence-ender	?	5	0.1%
Not	n?t	2	0.0%

3.3.2 Supervised Learning Approaches

In NLP, POS taggers for complete sentences are trained in a supervised manner based on a set of labeled data. The trained taggers for a language can be generally applied to any texts in that language. For example, the Brill Tagger [7] and the Stanford Tagger [31] are freely available and can be used to tagger English texts. Barr et al. [1] took these taggers off the shelf and applied them on their human annotated queries. They found that the accuracy from these standard taggers was well below a simple baseline that tags a query term based on the most frequent POS tag obtained from their own labeled queries. Their findings highlighted the need to train a specific POS tagger for web queries.

Based on the queries with labeled POS tags, the most basic supervised learning approach is called the Most Common Tag in [1]. In this method, a mapping between a word and a tag is constructed. The tag for a word is the most common tag counted from the labeled query data. In the prediction phase, the POS tags for a query are just a simple lookup from the mapping for each individual words. Though fairly simple, this method was shown to be better than the standard taggers [1].

Barr et al. [1] also proposed to train a Brill Tagger using the labeled query data, instead of the standard Treebank data. They found that the trained tagger outperformed the Most Common Tag one, showing the promise of more advanced supervised learning approaches. Due to the difference between queries and sentences, some commonly useful features are not available for queries. A promising direction for supervised learning approaches is to craft those missing features to improve the tagging accuracy. For example, capitalization information is noisy in query data. Barr et al. [1] proposed an automatic query term capitalization method based on their capitalization statistics in search results. They found that this can boost the Brill tagger significantly given that proper nouns are the most frequent in web queries and capitalization is an indicative feature to tag proper nouns.

3.3.3 Transfer Learning Approaches

Supervised learning approaches need labeled data. Given the vast amount of search queries, creating a labeled data set with sufficient coverage and diversity is challenging. However, tagging well-formed sentences is well-studied. This motivates many tagging methods based on the transfer learning principle. The common strategy in the existing approaches is to leverage top retrieved results or clicked ones to help query tagging. We review them in this section.

3.3.3.1 Simple Transfer Methods

Bendersky et al. [3] proposed a pseudo-relevance feedback approach to tag queries. In this approach, a pre-trained tagger is used to tag both query terms and sentences from top retrieved documents. For each term, there are two multinomial distributions over POS tags: one based on the tagging results of the given query and the other based on the POS tag counts from the top retrieved documents. The two distributions are interpolated to give the final distribution of POS tags for each individual query term.

Keyaki et al. [16] used a similar methodology to the one used by Bendersky et al. [3], but proposed to precompute the tags for web documents offline. This can reduce the heavy computation needed in the pseudo-relevance feedback approach used in [3]. Specifically, the proposed method has the following two steps:

Offline. Given a web corpus, morphological analysis is conducted on every sentence in the corpus based on standard NLP methods. The POS tag of each term in a sentence is obtained. The output of this step is a large collection of sentences with POS tags on all terms. This is a pre-computing step and conducted offline.

Online. When a query is issued, sentences (with POS tag for each term) that contain two or more query terms are retrieved from the sentence collection created during offline computation. Then appropriate POS tags of query terms are obtained based on the POS tags of terms appearing in the retrieved sentences. With regard to a single term query, the most frequently appearing POS tag in the web corpus is tagged to the query term.

It can be seen that this method is designed to work with the following two properties: (1) capitalization information in query is not used and (2) word order in queries does not matter. In fact, they relied on the sentences in the web corpus to provide a high accuracy tagging. The shortcoming of this method is on the online retrieval part given that sentences are used as retrieval units for queries. The retrieval accuracy could be lower due to the short length of sentences and this can affect the query tagging accuracy in turn.

3.3.3.2 Learning Methods

Ganchev et al. [11] employed a more complete transfer learning method based on search logs. Search logs consist of both queries and "relevant" search results that are either retrieved by a search engine or clicked by end users. The training data in the "source" domain was human annotated sentences. A supervised POS tagger was trained based on the "source" training data and applied to the search result snippets. The POS tags on these snippets were then transferred to queries. In this process, the tag of a query term was the most frequent tag of the term in the tagged snippets. This simple transfer process produced a set of noisy labeled queries. Then a new query

tagger was trained based on the combination of the "source" training data and the noisy labeled query data. The pre-trained tagger can be used by itself to tag input queries. Ganchev et al. [11] compared using clicked documents and the top retrieved results and found that both methods performed similarly.

Sun et al. [29] proposed to transfer both POS tags and dependency parsing results from clicked sentences to queries. A click sentence is a well-formed sentence that (1) contains all query tokens and (2) appears in the top clicked documents of the query. For each sentence, both POS tags of individual terms and dependency between terms were constructed. While it is simple to transfer POS tags from sentences to queries similarly as previous methods, it is challenging to transfer the dependency relations because not all words in a sentence appear in queries. Sun et al. [29] proposed heuristics to handle the following cases and an uninformative "dep" relation was also introduced:

- Directly connected (46%)
- Connected via function words (24%)
- Connected via modifiers (24%)
- Connected via a head noun (4%)
- Connected via a verb (2%)

Sun et al. [29] also proposed methods to infer a unique dependency tree for a query and refine dependency labels for the placeholder "dep." All these resulted in a query treebank without additional manual labeling. A syntactic parser was then trained from the web query treebank data and shown to be more accurate than standard parsers.

3.3.4 Summary

For query syntactic tagging, the majority of existing approaches transfer information from sentences in search results or snippets to search queries. POS tags of queries and documents can also be used to define matching features to improve ranking accuracy [1]. In contrast to the POS tagging, dependency parsing is not fully exploited for web search. Recent work by Tsur et al. [33] and Pinter et al. [25] focused identifying queries with question intents and their syntactic parsing. A query treebank is created and can be used to further study query syntactic tagging.

3.4 Query Semantic Tagging

The problem of query semantic tagging is to assign labels, from a set of pre-defined semantic ones, at word level. Such labels are usually domain-specific. An example from [20] of query semantic tagging is in the following where the labels are in parentheses and all the labels are in the *product* domain.

cheap (SortOrder) **garmin** (Brand) **steetpilot** (Model) **c340** (Model) **gps** (Type)

Semantic labels can be used to provide users with more relevant search results. For example, many specialized search engines build their indexes directly from a relational database where structured information or labels are available in the documents (e.g., Brand = "garmin"). Query semantic labels can thus be used to match documents more accurately. In this section, we discuss named entity recognition on a coarse level and grammar-based approaches in a fine-grained domain-specific level.

3.4.1 Named Entity Recognition

As shown in [12], about 71% of search queries contain named entities. Given their high percentage, identifying named entities, as known as Named Entity Recognition (NER), becomes an important task for web search. For named entities, the classes of labels include "Game," " Movie," "Book," "Music," etc. Given a query, the tasks of NER are to identify which words in the query represent named entities and classify them into different classes.

For NER tasks, Guo et al. [12] found that only 1% of the named entity queries contain more than 1 entity and the majority of named entity queries contain exactly a single one. Thus a named entity query can be thought as containing two parts: entities and contexts. For example "harry porter walkthrough" contains entity "harry porter" and context "walkthrough" and the context indicates "harry porter" should be labeled as "Game." Without this context, the query "harry porter" can also be a "Book" or "Movie." This shows that the classes for named entities can be ambiguous and its context in the query helps disambiguate them.

Traditional NER is mainly performed on natural language texts [6] and a supervised learning approach based on hand-crafted features is exploited (e.g., whether the word is capitalized or whether "Mr." or "Ms." is before the word). These features can be extracted and utilized in the NER tasks for natural language texts. However, directly applying them on queries would not perform well, because queries are very short and are not necessarily in standard forms. In the current literature, weakly supervised methods are proposed for NER on queries.

3.4.1.1 Template-Based Approach

A template-based approach was proposed by Paşca [23] that aimed to extract named entities from search logs based on a small set of seeds. This method does not need

hand-crafted extraction patterns nor domain-specific knowledge. An example of the procedure is displayed in Fig. 3.2. The procedure starts with a set of seed instances in a category ("Drug" in the example) and proceeds with the following steps:

Step 1 Identify query templates that match the seed instances. For each seed instance, all queries containing this instance are located. The prefix and suffix of each matched query become one template and templates from all matched queries of all the seed instances become a collection of templates.

Step 2 Identify candidate instances. Based on the collection of templates from Step 1, this step is to match the template against all queries in the search logs. The non-template parts of the matched queries become the candidates. The assumption of this step is that instances belonging to the target category should share the templates.

Step 3 Internal representation of the candidate instances. For each candidate, each template matched in Step 2 becomes a dimension in the template vector used as the internal representation. All templates form a signature vector for the candidate.

Step 4 Similarly, internal representation of the seed instances is created. These vectors are aggregated together as the reference vector for the target category.

Step 5 All the candidate instances are then ranked by the similarity between its signature vector and the reference vector for the category.

All the steps for the category "Drug" are shown in Fig. 3.2. The seed instances are {*phentermine, viagra, vicodin, vioxx, xanax*} and the output of the method is an enlarged set of list {*viagra, phentermine, ambien, adderall, vicodin, hydrocodone, xanax, vioxx, oxycontin, cialis, valium, lexapro, ritalin, zoloft, percocet,* ... }.

As seen in the data flow, this method only needs a very small number of seed instances and is thus weakly supervised. Paşca [23] used tens of target categories and tens of seed entities in each target category. The method, though simple, is shown to be effective in discovering more named entities in target categories.

3.4.1.2 Weakly Supervised Learning Approach

The approach in [23] is based on heuristics. Inspired by it, Guo et al. [12] formulate a topic model based learning approach in a more principled manner. It follows the same staring points as [23] where a set of seed instances of each target class is provided. We thus call such an approach weakly supervised learning approach.

In this approach, a query having one named entity is represented as a triple (e, t, c), where e denotes named entity, t denotes context of e, and c denote class of e. Note that t can be empty (i.e., no context). Then the goal of NER here becomes to find the triple (e, t, c) for a given query q, which has the largest joint probability $Pr(e, t, c)$. The joint probability is factorized:

$$Pr(e, t, c) = Pr(e)Pr(c|e)Pr(t|e, c) = Pr(e)Pr(c|e)Pr(t|c). \tag{3.26}$$

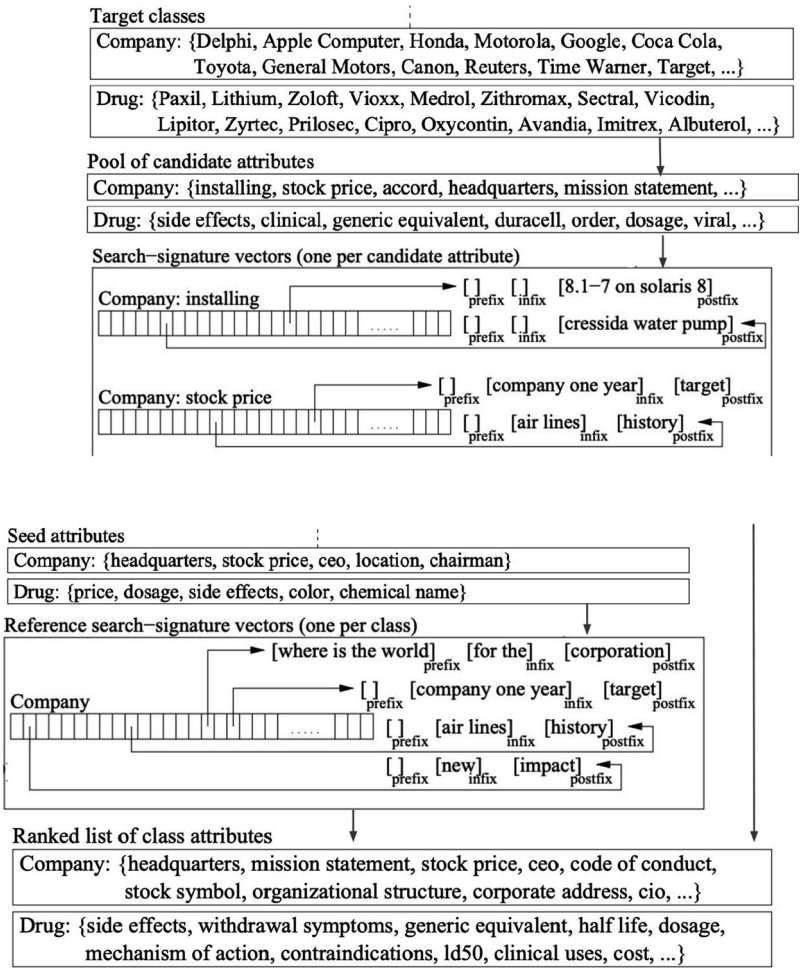

Fig. 3.2 Data flow overview of the template-based approach for named entity recognition in [23]

It is assumed that context only depends on class, but not on any specific named entity. This is similar to [23] where the reference vector of a target category only depends on templates.

In search logs, only query strings are available: contexts and name entities are not explicitly labeled, nor classes of entities. Guo et al. [12] took a weakly supervised approach where a set of named entities are collected and labeled as seeds.

$$S = \{(e, c)\} \tag{3.27}$$

Starting from the seed S, a list of training instances $\{(e, t)\}$ are obtained as follows:

Step 1 For each $e \in S$, collect all queries that contain e from search logs. All the queries obtained thus have entities and contexts separated and form a data set $D_1 = \{(e, t)\}$.

Step 2 Estimate $Pr(c|e)$ and $Pr(t|c)$ based on S and D_1 using the Weakly Supervised LDA (WS-LDA) model described below.

Step 3 For each $t \in D_1$, collect all queries that contain t from search logs and form $D_2 = \{(e, t)\}$ for all t.

Step 4 With $Pr(t|c)$ fixed, estimate $Pr(c|e)$ based on WS-LDA for all $\{e : e \notin S\}$. $Pr(e)$ is also estimated as proportional to the empirical frequency of e in D_2.

The proposed WS-LDA model is based on the traditional LDA model [5]. Given a data set of $D = \{(e, t)\}$, we can treat e as the "document" and t as the "words" in LDA model. Class c becomes the hidden variable and then

$$Pr(e, t) = Pr(e) \sum_{c} Pr(c|e) Pr(t|c) \tag{3.28}$$

Such a formulation is easy to be mapped to the LDA framework and the parameter $Pr(c|e)$ and $Pr(t|c)$ can be estimated by fitting the model to the data D.

The WS-LDA model leverages the seed S data set as weakly supervised signals to serve two purposes: (1) the set of classes are pre-defined in WS-LDA; (2) the estimated $Pr(c|e)$ should be close to S. The latter is achieved by introducing a regularization term in the LDA objective function to be maximized:

$$\sum_{(e,c) \in S} Pr(c|e). \tag{3.29}$$

In this way, $Pr(c|e)$ and $Pr(t|c)$ estimated from WS-LDA optimize the fitness on the search logs and stay closely to the seed labels in S as well.

The WS-LDA model also provides a natural way to give prediction for an input query. For an input query q, it can enumerate all possible segmentations of (e, t) of q and label q with class c based on the parameters estimated from WS-LDA:

$$(e, t, c)^* = \arg \max_{(e,t,c)} Pr(q, e, t, c) = \arg \max_{(e,t,c):(e,t)=q} Pr(e, t, c). \tag{3.30}$$

The WS-LDA approach was further extended to incorporated search sessions [10] in which the adjacent queries in a search session were used to improve the class label prediction for the named entities. Furthermore, building a taxonomy of named entities search intents based on unsupervised learning approaches such as hierarchical clustering was proposed by Yin and Shah [35].

3.4.2 *Fine-Grained Tagging*

Fine-grained query semantic tagging has a strong NLP flavor than the methods for NER above. For example, Manshadi and Li [20] defined a context-free grammar for queries in the "product" domain. The design choice of being context-free is to accommodate the loose order property of words in queries. Based on the grammar, a parse tree was constructed for an input query and the nodes in the parse tree (e.g., Brand or Model) were the semantic tags for the queries. Standard tagging methods such as Conditional Random Fields (CRF) [17] have a strong assumption on the order of the input sentences and is thus less effective for query tagging. Manshadi and Li [20] compared their grammar-based approach with CRF models and found that their methods performed better. This demonstrates the unique challenges of query semantic tagging and special design choices such as context-free grammar are critical for this task.

3.5 Conclusions

In this chapter, we reviewed the existing literature on query tagging. We classified them into query segmentation, query syntactic tagging, and query semantic tagging. We reviewed a few representative methods for each category and discussed their pros and cons. This chapter is just a starting point for the work in query tagging. It is by no means exhaustive in the research area. Our hope is to give an introduction to this exciting but challenging areas and an overview of the existing work. There are a few future directions for query tagging. (1) There is still considerable room to improve the accuracy of each query tagging task. For example, more and more user interaction data is accumulated over time for search engines. How to explore this huge amount of data as external resources to boost each tagging task is worth studying. (2) The recent development of deep learning techniques has advanced the NLP techniques. How to leverage the newly developed NLP techniques on query tagging is also an interesting direction. (3) Query tagging can benefit other IR tasks such as query suggestion. It looks promising to study how to leverage query tagging on these related tasks.

References

1. Cory Barr, Rosie Jones, and Moira Regelson. The linguistic structure of English web-search queries. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 1021–1030, 2008.
2. Michael Bendersky, W. Bruce Croft, and David A. Smith. Two-stage query segmentation for information retrieval. In *Proceedings of the 32nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 810–811, 2009.

3. Michael Bendersky, W. Bruce Croft, and David A. Smith. Structural annotation of search queries using pseudo-relevance feedback. In *Proceedings of the 19th ACM Conference on Information and Knowledge Management*, pages 1537–1540, 2010.
4. Shane Bergsma and Qin Iris Wang. Learning noun phrase query segmentation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 819–826, 2007.
5. David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent Dirichlet allocation. *J. Mach. Learn. Res.*, 3: 993–1022, 2003.
6. Andrew Eliot Borthwick. *A Maximum Entropy Approach to Named Entity Recognition*. PhD thesis, 1999.
7. Eric Brill. Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Comput. Linguistics*, 21 (4): 543–565, 1995.
8. Kenneth Ward Church and Patrick Hanks. Word association norms, mutual information, and lexicography. *Comput. Linguistics*, 16 (1): 22–29, 1990.
9. Carl de Marcken. *Unsupervised language acquisition*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1996.
10. Junwu Du, Zhimin Zhang, Jun Yan, Yan Cui, and Zheng Chen. Using search session context for named entity recognition in query. In *Proceeding of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 765–766, 2010.
11. Kuzman Ganchev, Keith B. Hall, Ryan T. McDonald, and Slav Petrov. Using search-logs to improve query tagging. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, pages 238–242, 2012.
12. Jiafeng Guo, Gu Xu, Xueqi Cheng, and Hang Li. Named entity recognition in query. In *Proceedings of the 32nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 267–274, 2009.
13. Matthias Hagen, Martin Potthast, Benno Stein, and Christof Bräutigam. Query segmentation revisited. In *Proceedings of the 20th International Conference on World Wide Web*, pages 97–106, 2011.
14. Rosie Jones, Benjamin Rey, Omid Madani, and Wiley Greiner. Generating query substitutions. In *Proceedings of the 15th international conference on World Wide Web*, pages 387–396, 2006.
15. Ajinkya Kale, Thrivikrama Taula, Sanjika Hewavitharana, and Amit Srivastava. Towards semantic query segmentation. *CoRR*, abs/1707.07835, 2017.
16. Atsushi Keyaki and Jun Miyazaki. Part-of-speech tagging for web search queries using a large-scale web corpus. In *Proceedings of the Symposium on Applied Computing*, pages 931–937, 2017.
17. John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 282–289, 2001.
18. Yanen Li, Bo-June Paul Hsu, ChengXiang Zhai, and Kuansan Wang. Unsupervised query segmentation using clickthrough for information retrieval. In *Proceeding of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 285–294, 2011.
19. Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, USA, 1999. ISBN 0-262-13360-1.
20. Mehdi Manshadi and Xiao Li. Semantic tagging of web search queries. In *Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing*, pages 861–869, 2009.
21. Donald Metzler and W. Bruce Croft. A Markov random field model for term dependencies. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 472–479, 2005.
22. Nikita Mishra, Rishiraj Saha Roy, Niloy Ganguly, Srivatsan Laxman, and Monojit Choudhury. Unsupervised query segmentation using only query logs. In *Proceedings of the 20th International Conference on World Wide Web*, pages 91–92, 2011.

23. Marius Pasca. Weakly-supervised discovery of named entities using web search queries. In *Proceedings of the Sixteenth ACM Conference on Information and Knowledge Management*, pages 683–690, 2007.
24. Fuchun Peng, Fangfang Feng, and Andrew McCallum. Chinese segmentation and new word detection using conditional random fields. In *Proceedings of the 20th International Conference on Computational Linguistics*, pages 562–568, 2004.
25. Yuval Pinter, Roi Reichart, and Idan Szpektor. Syntactic parsing of web queries with question intent. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 670–680, 2016.
26. Knut Magne Risvik, Tomasz Mikolajewski, and Peter Boros. Query segmentation for web search. In *Proceedings of the Twelfth International World Wide Web Conference*, 2003.
27. Rishiraj Saha Roy, Anusha Suresh, Niloy Ganguly, and Monojit Choudhury. Improving document ranking for long queries with nested query segmentation. In *Proceedings of the 38th European Conference on IR Research*, pages 775–781, 2016.
28. Richard Sproat, Chilin Shih, William Gale, and Nancy Chang. A stochastic finite-state word-segmentation algorithm for Chinese. *Comput. Linguistics*, 22 (3): 377–404, 1996.
29. Xiangyan Sun, Haixun Wang, Yanghua Xiao, and Zhongyuan Wang. Syntactic parsing of web queries. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1787–1796, 2016.
30. Bin Tan and Fuchun Peng. Unsupervised query segmentation using generative language models and Wikipedia. In *Proceedings of the 17th International Conference on World Wide Web*, pages 347–356, 2008.
31. Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, 2003.
32. Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. *J. Mach. Learn. Res.*, 6: 1453–1484, 2005.
33. Gilad Tsur, Yuval Pinter, Idan Szpektor, and David Carmel. Identifying web queries with question intent. In *Proceedings of the 25th International Conference on World Wide Web*, pages 783–793, 2016.
34. Haocheng Wu, Yunhua Hu, Hang Li, and Enhong Chen. A new approach to query segmentation for relevance ranking in web search. *Inf. Retr. J.*, 18 (1): 26–50, 2015.
35. Xiaoxin Yin and Sarthak Shah. Building taxonomy of web search intents for name entity queries. In *Proceedings of the 19th International Conference on World Wide Web*, pages 1001–1010, 2010.
36. Xiaohui Yu and Huxia Shi. Query segmentation using conditional random fields. In *Proceedings of the First International Workshop on Keyword Search on Structured Data*, pages 21–26, 2009.
37. ChengXiang Zhai. Fast statistical parsing of noun phrases for document indexing. In *Proceedings of the 5th Applied Natural Language Processing Conference*, pages 312–319, 1997.
38. Chao Zhang, Nan Sun, Xia Hu, Tingzhu Huang, and Tat-Seng Chua. Query segmentation based on eigenspace similarity. In *Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 185–188, 2009.