Semantic Tagging of Web Search Queries

Mehdi Manshadi
University of Rochester
Rochester, NY
mehdih@cs.rochester.edu

Xiao Li
Microsoft Research
Redmond, WA
xiaol@microsoft.com

Abstract

We present a novel approach to parse web search queries for the purpose of automatic tagging of the queries. We will define a set of probabilistic context-free rules, which generates bags (i.e. multi-sets) of words. Using this new type of rule in combination with the traditional probabilistic phrase structure rules, we define a hybrid grammar, which treats each search query as a bag of chunks (i.e. phrases). A hybrid probabilistic parser is used to parse the queries. In order to take contextual information into account, a discriminative model is used on top of the parser to re-rank the n-best parse trees generated by the parser. Experiments show that our approach outperforms a basic model, which is based on Conditional Random Fields.

1 Introduction

Understanding users' intent from web search queries is an important step in designing an intelligent search engine. While it remains a challenge to have a scientific definition of "intent", many efforts have been devoted to automatically mapping queries into different *domains* i.e. *topical classes* such as *product*, *job* and *travel* (Broder et al. 2007; Li et al. 2008). This work goes beyond query-level classification. We assume that the queries are already classified into the correct domain and investigate the problem of *semantic tagging* at the word level, which is to assign a label from a set of pre-defined semantic labels (specific to the domain) to every word in the query. For example, a search query in the *product* domain can be tagged as:

cheap	*garmin*	*streetpilot*	*c340*	*gps*
SortOrder	*Brand*	*Model*	*Model*	*Type*

Many specialized search engines build their indexes directly from relational databases, which contain highly structured information. Given a query tagged with the semantic labels, a search engine is able to compare the values of semantic labels in the query (e.g., *Brand = "garmin"*) with its counterpart values in documents, thereby providing users with more relevant search results.

Despite this importance, there has been relatively little published work on semantic tagging of web search queries. Allan and Raghavan (2002) and Barr et al. (2008) study the linguistic structure of queries by performing part-of-speech tagging. Pasca et al. (2007) use queries as a source of knowledge for extracting prominent attributes for semantic concepts.

On the other hand, there has been much work on extracting structured information from larger text segments, such as addresses (Kushmerick 2001), bibliographic citations (McCallum et al. 1999), and classified advertisements (Grenager et al. 2005), among many others. The most widely used approaches to these problems have been sequential models including *hidden Markov models (HMMs)*, *maximum entropy Markov models (MEMMs)* (Mccallum 2000), and *conditional random fields (CRFs)* (Lafferty et al. 2001)

These sequential models, however, are not optimal for processing web search queries for the following reasons.. The first problem is that the global constraints and long distance dependencies on state variables are difficult to capture using sequential models. Because of this limitation, Viola and Narasimhand (2007) use a discriminative context-free (phrase structure) grammar for extracting information from semi-structured data and report higher performances over CRFs.

Secondly, sequential models treat the input text as an ordered sequence of words. A web search query, however, is often formulated by a user as a bag of keywords. For example, if a user is look-

ing for *cheap garmin gps*, it is possible that the query comes in any ordering of these three words. We are looking for a model that, once it observes this query, assumes that the other permutations of the words in this query are also likely. This model should also be able to handle cases where some local orderings have to be fixed as in the query *buses from New York City to Boston*, where the words in the phrases *from New York city* and *to Boston* have to come in the exact order.

The third limitation is that the sequential models treat queries as unstructured (linear) sequences of words. The study by Barr et al. (2008) on *Yahoo*! query logs suggests that web search queries, to some degree, carry an underlying linguistic structure. As an example, consider a query about finding a local business near some location such as:

seattle wa drugstore 24/7 98109

This query has two constituents: the *Business* that the user is looking for (*24/7 drugstore*) and the *Neighborhood* (*seattle wa 98109*). The model should not only be able to recognize the two constituents but it also needs to understand the structure of each constituent. Note that the arbitrary ordering of the words in the query is a big challenge to understanding the structure of the query. The problem is not only that the two constituents can come in either order, but also that a subconstituent such as *98109* can also be far from the other words belonging to the same constituent. We are looking for a model that is able to generate a hierarchical structure for this query as shown in figure (1).

The last problem that we discuss here is that the two powerful sequential models i.e. MEMM and CRF are discriminative models; hence they are highly dependent on the training data. Preparing labeled data, however, is very expensive. Therefore in cases where there is no or a small amount of labeled data available, these models do a poor job.

In this paper, we define a hybrid, generative grammar model (section 3) that generates *bags of phrases* (also called *chunks* in this paper). The chunks are generated by a set of *phrase structure* (*PS*) rules. At a higher level, a bag of chunks is generated from individual chunks by a second type of rule, which we call *context-free multiset generating* rules. We define a probabilistic version of this grammar in which every rule has a probability associated with it. Our grammar model eliminates the local dependency assumption made by sequential models and the ordering

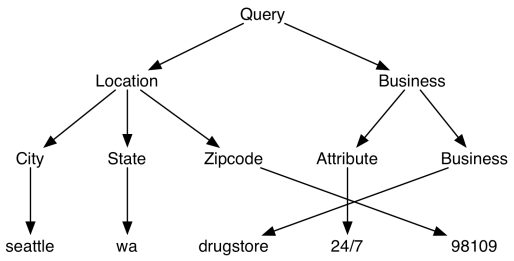

Figure 1. A simple grammar for product domain

constraints imposed by phrase structure grammars (*PSG*). This model better reflects the underlying linguistic structure of web search queries. The model's power, however, comes at the cost of increased time complexity, which is exponential in the length of the query. This, is less of an issue for parsing web search queries, as they are usually very short (2.8 words/query in average (Xue et al., 2004)).

Yet another drawback of our approach is due to the context-free nature of the proposed grammar model. Contextual information often plays a big role in resolving tagging ambiguities and is one of the key benefits of discriminative models such as CRFs. But such information is not straightforward to incorporate in our grammar model. To overcome this limitation, we further present a discriminative re-ranking module on top of the parser to re-rank the n-best parse trees generated by the parser using contextual features. As seen later, in the case where there is not a large amount of labeled data available, the parser part is the dominant part of the module and performs reasonably well. In cases where there is a large amount of labeled data available, the discriminative re-ranking incorporates into the system and enhances the performance. We evaluate this model on the task of tagging search queries in the *product* domain. As seen later, preliminary experiments show that this hybrid generative/discriminative model performs significantly better than a CRF-based module in both absence and presence of the labeled data.

The structure of the paper is as follows. Section 2 introduces a linguistic grammar formalism that motivates our grammar model. In section 3, we define our grammar model. In section 4 we address the design and implementation of a parser for this kind of grammar. Section 5 gives an example of such a grammar designed for the purpose of automatic tagging of queries. Section 6 discusses motivations for and benefits of running a discriminative re-ranker on top of the parser. In section 7, we explain the evaluations

and discuss the results. Section 8 summarizes this work and discusses future work.

2 ID/LP Grammar

Context-free phrase structure grammars are widely used for parsing natural language. The adequate power of this type of grammar plus the efficient parsing algorithms available for it has made it very popular. PSGs treat a sentence as an ordered sequence of words. There are however natural languages that are free word order. For example, a three-word sentence consisting of a subject, an object and a verb in Russian, can occur in all six possible orderings. PSGs are not a well-suited model for this type of language, since six different PS-rules must be defined in order to cover such a simple structure. To address this issue, Gazdar (1985) introduced the concept of ID/LP rules within the framework of *Generalized Phrase Structure Grammar* (*GPSG*). In this framework, *Immediate Dominance* or *ID* rules are of the form:

(1) $A \rightarrow B, C$

This rule specifies that a non-terminal *A* can be rewritten as *B* and C, but it does not specify the order. Therefore *A* can be rewritten as both *BC* and *CB*. In other words the rule in (1) is equivalent to two PS-rules:

(2) $A \rightarrow BC$
$A \rightarrow CB$

Similarly one ID rule will suffice to cover the simple subject-object-verb structure in Russian:

(3) $S \rightarrow Sub, Obj, Vrb$

However even in free-word-order languages, there are some ordering restrictions on some of the constituents. For example in Russian an adjective always comes before the noun that it modifies. To cover these ordering restrictions, Gazdar defined *Linear Precedence* (*LP*) rules. (4) gives an example of a linear precedence rule:

(4) $ADJ < N$

This specifies that ADJ always comes before N when both occur on the right-hand side of a single rule.

Although very intuitive, ID/LP rules are not widely used in the area of natural language processing. The main reason is the time-complexity issue of ID/LP grammar. It has been shown that parsing ID/LP rules is an NP-complete problem (Barton 1985). Since the length of a natural language sentence can easily reach 30-40 (and sometimes even up to 100) words, ID/LP grammar is not a practical model for natural language syntax. In our case, however, the time-complexity is not a bottleneck as web search queries are usually very short (2.8 words per query in average). Moreover, the nature of ID rules can be deceptive as it might appear that ID rules allow any reordering of the words in a valid sentence to occur as another vaild sentence of the language. But in general this is not the case. For example consider a grammar with only two ID rules given in (5) and consider S as the start symbol:

(5) $S \rightarrow B, c$
$B \rightarrow d, e$

It can be easily verified that *dec* is a sentence of the language but *dce* is not. In fact, although the permutation of subconstituents of a constituent is allowed, a subconstituent can not be pulled out from its mother consitutent and freely move within the other constituents. This kind of movement however is a common behaviour in web search queries as shown in figure (1). It means that even ID rules are not powerful enough to model the free-word-order nature of web search queries. This leads us to define to a new type of grammar model.

3 Our Grammar Model

3.1 The basic model

We propose a set of rules in the form:

(6) $S \rightarrow \{B, c\}$
$B \rightarrow \{D, E\}$
$D \rightarrow \{d\}$
$E \rightarrow \{e\}$

which can be used to generate *multisets* of words. For the notation convenience and consistancy, throughout this paper, we show terminals and non-terminals by lowercase and uppercase letters, respectively and sets and multisets by bold font uppercase letters. Using the rules in (6) a sentence of the language (which is a multiset in this model) can be derived as follows:

(7) $S \Rightarrow \{B, c\} \Rightarrow \{D, E, c\} \Rightarrow \{D, e, c\} \Rightarrow \{d, e, c\}$

Once the set is generated, it can be realized as any of the six permutation of *d, e,* and *c*. Therefore a single sequence of derivations can lead to six different strings of words. As another example consider the grammar in (8).

(8)
$Query \rightarrow \{Business, Location\}$
$Business \rightarrow \{Attribute, Business\}$
$Location \rightarrow \{City, State\}$
$Business \rightarrow \{drugstore\} \mid \{Resturant\}$
$Attribute \rightarrow \{Chinese\} \mid \{24/7\}$
$City \rightarrow \{Seattle\} \mid \{Portland\}$
$State \rightarrow \{WA\} \mid \{OR\}$

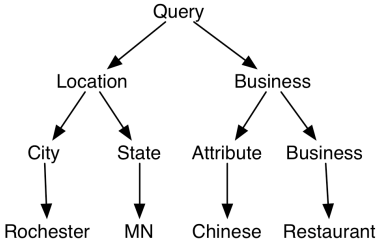

Figure 2. A CFSG parse tree

where *Query* is the start symbol and by $A \rightarrow B|C$ we mean two differnet rules $A \rightarrow B$ and $A \rightarrow C$. Figures (2) and (3) show the tree structures for the queries *Restaurant Rochester Chinese MN*, and *Rochester MN Chinese Restaurant*, respectively. As seen in these figures, no matter what the order of the words in the query is, the grammar always groups the words *Resturant* and *Chinese* together as the *Business* and the words *Rochester* and *MN* together as the *Location*. It is important to notice that the above grammars are context-free as every non-terminal *A,* which occurs on the left-hand side of a rule *r*, can be replaced with the set of terminals and non-terminals on the right-hand side of *r*, no matter what the context in which *A* occurs is.

More formally we define a *Context-Free multiSet generating Grammar* (*CFSG*) as a 4-tuple $G=(\boldsymbol{N}, \boldsymbol{T}, S, \boldsymbol{R})$ where

- $\boldsymbol{N}$ is a set of *non-terminals;*
- $\boldsymbol{T}$ is a set of *terminals;*
- $S \in \boldsymbol{N}$ is a special non-terminal called *start symbol,*
- $\boldsymbol{R}$ is a set of rules $\{A^i \rightarrow \boldsymbol{X}^j\}$ where A^i is a non-terminal and $\boldsymbol{X}^j$ is a *set* of terminals and non-terminals.

Given two multisets $\boldsymbol{Y}$ and $\boldsymbol{Z}$ over the set $\boldsymbol{N} \cup \boldsymbol{T}$, we say $\boldsymbol{Y}$ dervies $\boldsymbol{Z}$ (shown as $\boldsymbol{Y} \Rightarrow \boldsymbol{Z}$) iff there exists A, $\boldsymbol{W}$, and $\boldsymbol{X}$ such that:

$\boldsymbol{Y} = \boldsymbol{W} + \{A\}$[1]
$\boldsymbol{Z} = \boldsymbol{W} + \boldsymbol{X}$
$A \rightarrow \boldsymbol{X} \in \boldsymbol{R}$

Here $\Rightarrow^*$ is defined as the *reflexive transitive* closure of $\Rightarrow$. Finally we define the language of multisets generated by the grammar G (shown as $\boldsymbol{L}(G)$) as

$L = \{ \boldsymbol{X} \mid \boldsymbol{X}$ *is a multiset over* $\boldsymbol{N} \cup \boldsymbol{T}$ *and* $S \Rightarrow^* \boldsymbol{X}\}$

The sequence of $\Rightarrow$ used to derive $\boldsymbol{X}$ from S is called a derivation of $\boldsymbol{X}$. Given the above definitions, *parsing* a multiset $\boldsymbol{X}$ means to find all (if any) the derivations of $\boldsymbol{X}$ from S. [2]

[1] If X and Y are two multisets, X+Y simply means appending X to Y. For example {a, b, a} + {b, c, d} = {a, b, a, b, c, d}.

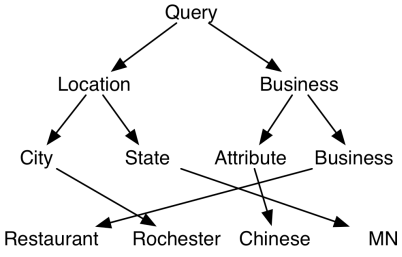

Figure 3. A CFSG parse tree

3.2 Probabilisic CFSG

Very often a sentence in the language has more than one derivation, that is the sentence is syntactically ambiguous. One natural way of resolving the ambiguity is using a probabilistic grammar. Analogous to *PCFG* (Manning and Schütze 1999), we define the probabilistic version of a CFSG, in which every rule $A^i \rightarrow \boldsymbol{X}^j$ has a probability $P(A^i \rightarrow \boldsymbol{X}^j)$ and for every non-terminal A^i, we have:

$$\Sigma_j P(A^i \rightarrow \boldsymbol{X}^j) = 1 \quad (9)$$

Consider a sentence $w_1 w_2 \ldots w_n$, a parse tree T of this sentence, and an interior node v in T labeled with A_v and assume that $v_1, v_2, \ldots v_k$ are the children of the node v in T. We define:

$$\alpha(v) = P(A_v \rightarrow \{A_{v1} \ldots A_{vk}\})\alpha(v_1) \ldots \alpha(v_k) \quad (10)$$

with the initial conditions $\alpha(w_i)=1$. If u is the root of the tree T we have:

$$P(w_1 w_2 \ldots w_n, T) = \alpha(u) \quad (11)$$

The parse tree that the probabilistic model assigns to the sentence is defined as:

$$T_{max} = \text{argmax}_T (P(w_1 w_2 \ldots w_n, T)) \quad (12)$$

where T ranges over all possible parse trees of the sentence.

4 Parsing Algorithm

4.1 Deterministic parser

The parsing algorithm for the CFSG is straightforward. We used a modified version of the *Bottom-Up Chart Parser* for the phrase structure grammars (Allen 1995, see 3.4). Given the grammar $G=(\boldsymbol{N},\boldsymbol{T},S,\boldsymbol{R})$ and the query $q=w_1 w_2 \ldots w_n$, the algorithm in figure (4) is used to parse q. The algorithm is based on the concept of an *active arc*. An *active arc* is defined as a 3–

[2] Every sentence of a language corresponds to a vector of $|\boldsymbol{T}|$ integers where the k^{th} element represents how many times the k^{th} terminal occurs in the multi-set. In fact, the languages defined by grammars are not interesting but the derivations are.

tuple *(r, **U**, **I**)* where *r* is a rule $A \rightarrow \boldsymbol{X}$ in ***R***, ***U*** is a subset of ***X***, and ***I*** is a subset of *{1, 2 ...n}* (where *n* is the number of words in the query). This active arc tries to find a match to the right-hand side of *r* (i.e. ***X**)* and suggests to replace it with the non-terminal *A*. ***U*** contains the part of the right-hand side that has not been matched yet. Therefore when an arc is newly created ***U***=***X***. Equivalently, $\boldsymbol{X} \backslash \boldsymbol{U}$[3] is the part of the right hand side that has so far been matched with a subset of words in the query, where ***I*** stores the positions of these words in *q*.

An active arc is *completed* when ***U***=*Ø*. Every completed active arc can be reduced to a tuple *(A, **I**)*, which we call a *constituent*. A constituent *(A, **I**)* shows that the non-terminal *A* matches the words in the query that are positioned at the numbers in *I*. Every constituent that is built by the parser is stored in a data structure called *chart* and remains there throughout the whole process. *Agenda* is another data structure that temporarily stores the constituents. At initialization step, the constituents $(w_1, \{1\}), \ldots (w_n, \{n\})$ are added to both *chart* and *agenda*. At each iteration, we pull out a constituent from the agenda and try to find a match to this constituent from the remaining list of terminals and non-terminals on the right-hand side of an active arc. More precisely, given a constituent *c=(A, **I**)* and an active arc γ = $(r{:}B \rightarrow \boldsymbol{X}, \boldsymbol{U}, \boldsymbol{J})$, we check if $A \in \boldsymbol{U}$ and $\boldsymbol{I} \cap \boldsymbol{J} = \emptyset$; if so, γ is *extendable* by *c*, therefore we extend γ by removing *A* from ***U*** and appending ***I*** to ***J***. Note that the extension process keeps a copy of every active arc before it extends it. In practice every active arc and every constituent keep a set of pointers to its children constituents (stored in *chart*). This information is necessary for the termination step in order to print the parse trees. The algorithm succeeds if there is a constituent in the chart that corresponds to the start symbol and covers all the words in the query, i.e. there is a constituent of the form *(S, {1,2,....n})* in the *chart*.

4.2 Probabilistic Parser

The algorithm given in figure (4) works for a deterministic grammar. As mentioned before, we use a probabilistic version of the grammar. Therefore the algorithm is modified for the probabilistic case. The probabilistic parser keeps a probability *p* for every active arc and every constituent:

$\gamma = (r, \boldsymbol{U}, \boldsymbol{J}, p_\gamma)$

$c = (A, \boldsymbol{I}, p_c)$

When extending γ using *c*, we have:

(13) $p_\gamma \leftarrow p_\gamma \, p_c$

When creating *c* from the completed active arc γ :

(14) $p_c \leftarrow p_\gamma \, p(r)$

Although search queries are usually short, the running time is still an issue when the length of the query exceeds 7 or 8. Therefore a couple of techniques have been used to make the naïve algorithm more efficient. For example we have used pruning techniques to filter out structures with very low probability. Also, a dynamic programming version of the algorithm has been used, where for every subset ***I*** of the word positions and every non-terminal *A* only the highest-ranking constituent *c=(A, **I**, p)* is kept and the rest are ignored. Note that although more efficient, the dynamic programming version is still exponential in the length of the query.

5 A grammar for semantic tagging

As mentioned before, in our system queries are already classified into different *domains* like *movies, books, products,* etc. using an automatic query classifier. For every domain we have a *schema*, which is a set of pre-defined tags. For example figure (5) shows an example of a schema for the *product* domain. The task defined for this system is to automatically tag the words in the query with the tags defined in the schema:

cheap	*garmin*	*streetpilot*	*c340*	*gps*
|	|	|	|	|
SortOrder	*Brand*	*Model*	*Model*	*Type*

Initialization:
For each word w_i in *q* add $(w_i, \{i\})$ to *Chart* and to *Agenda*
For all $r{:}\ A \rightarrow \boldsymbol{X}$ in ***R***, create an active arc *(r, **X**, {})* and add it to the list of active arcs.

Iteration
Repeat
Pull a constituent *c = (A, **I**)* from *Agenda*
For every active arc $\gamma = (r{:}B \rightarrow \boldsymbol{X}, \boldsymbol{U}, \boldsymbol{I})$
Extend γ using *c* if extendable
If ***U***=*Ø* add *(B, **I**)* to *Chart* and to *Agenda*
Until *Agenda* is empty

Termination
For every item *c=(S, {1..n})* in *Chart*, return the tree rooted at *c*.

Figure 4. An algorithm for parsing deterministic CFSG

[3] $A \backslash B$ is defined as $\{x \mid x \in A \ \& \ x \notin B\}$

Type: *Camera, Shoe, Cell phone, …*
Brand: *Canon, Nike, At&t, …*
Model: *dc1700, powershot, ipod nano*
Attribute: *1GB, 7mpixel, 3X, …*
BuyingIntenet: *Sale, deal, …*
ResearchIntent: *Review, compare, …*
SortOrder: *Best, Cheap, …*
Merchant: *Walmart, Target, …*

Figure 5. Example of schema for product domain

We mentioned that one of the motivations of parsing search queries is to have a deeper understanding of the structure of the query. The evaluation of such a deep model, however, is not an easy task. There is no Treebank available for web search queries. Furthermore, the definition of the tree structure for a query is quite arbitrary. Therefore even when human resources are available, building such a Treebank is not a trivial task. For these reasons, we evaluate our grammar model on the task of automatic tagging of queries for which we have labeled data available. The other advantage of this evaluation is that there exists a CRF-based module in our system used for the task of automatic tagging. The performance of this module can be considered as the baseline for our evaluation.

We have manually designed a grammar for the purpose of automatic tagging. The resources available for training and testing were a set of search queries from the *product* domain. Therefore a set of CFSG rules were written for the *product* domain. We defined very simple and intuitive rules (shown in figure 6) that could easily be generalized to the other domains

Note that *Type*, *Brand*, *Model*, … could be either *pre-terminals* generating word tokens, or non-terminals forming the left-hand side of the phrase structure rules. For the *product* domain, *Type* and *Attribute* are generated by a phrase structure grammar. *Model* and *Attribute* may also be generated by a set of manually designed regular expressions. The rest of the tags are simply pre-terminals generating word tokens. Note that we have a lexicon, *e.g..,* a *Brand* lexicon, for all the tags except *Type* and *Attribute*. The model, however, extends the lexicon by including words discovered from labeled data (if available). The gray color for a non-terminal on the right-hand side (*RHS*) of some rule means that the non-terminal is *optional* (see *Query* rule in figure (6)). We used the optional non-terminals to make the task of defining the grammar easier. For example if we consider a rule with *n* optional non-terminals on its RHS, without optional non-terminals we have to define 2^n different rules to have an equivalent grammar. The parser can treat the optional non-terminals in different ways such as pre-compiling the rules to the equivalent set of rules with no optional non-terminal, or directly handling optional non-terminals during the parsing. The first approach results in exponentially many rules in the system, which causes sparsity issues when learning the probability of the rules. Therefore in our system the parser handles optional non-terminals directly. In fact, every non-terminal has its own probability for not occurring on the RHS of a rule, therefore the model learns n+1 probabilities for a rule with *n* optional non-terminals on its RHS: one for the rule itself and one for every non-terminal on its RHS. It means that instead of learning 2^n probabilities for 2^n different rules, the model only learns n+1 probabilities. That solves the sparsity problem, but causes another issue which we call *short length preference*. This occurs because we have assumed that the probability of a non-terminal being optional is independent of other optional non-terminals. Since for almost all non-terminals on the RHS of the query rule, the probability that the non-terminal does not exist in an instance of a query is higher than 0.5, a null query is the most likely query that the model generates! We solve this problem by conditioning the probabilities on the length of queries. This brings a trade-off between the two other alternatives: ignoring sparsity problem to prevent making many independence assumptions and making a lot of independence assumptions to address the sparsity issue.

Unlike sequential models, the grammar model is able to capture critical global constraints. For example, it is very unlikely for a query to have more than one *Type*, *Brand*, etc. This is an important property of the product queries that can help to resolve the ambiguity in many cases. In practice, the probability that the model learns for a rule like:

Query → *{Brand*, Product*, Model*, …}*
*Brand** → *{Brand}*
*Brand** → *{Brand*, Brand}*
*Type** → *{Type}*
*Type** → *{Type*, Type}*
*Model** → *{Model}*
*Model** → *{Model*, Model}*
…

Figure 6. A simple grammar for product domain

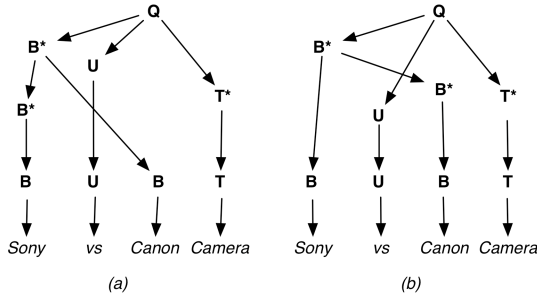

Figure 7. Two equivalent CFSG parse trees

*Type** → *{Type*, Type}*
compared to the rule:
*Type** → *Type*
is very small; the model penalizes the occurrence of more than one *Type* in a query. Figure (7a) shows an example of a parse tree generated for the query *"Canon vs Sony Camera"* in which *B*, *Q*, and *T* are abbreviations for *Brand*, *Query*, and *Type,* and *U* is a special tag for the words that does not fall into any other tag categories and have been left unlabeled in our corpus such as *a, the, for, etc.* Therefore the parser assigns the tag sequence *B U B T* to this query. It is true that the word "*vs"* plays a critical role in this query, representing that the user's intention is to compare the two brands; but as mentioned above in our labeled data such words has left unlabeled. The general model, however, is able to easily capture these sorts of phenomena.

A more careful look at the grammar shows that there is another parse tree for this query as shown in figure (7b). These two trees basically represent the same structure and generate the same sequence of tags. The number of trees generated for the same structure increases exponentially with the number of equal tags in the tree. To prevent this over-generation we used rules analogous to GPSG's LP rules such as:

B < B*

which allows only a unique way of generating a bag of the *Brand* tags. Using this LP rule, the only valid tree for the above query is the one in figure (7a).

6 Discriminative re-ranking

By using a context-free grammar, we are missing a great source of clues that can help to resolve ambiguity. Discriminative models, on the other hand, allow us to define numerous features, which can cooperate to resolve the ambiguities. Similar studies in parsing natural language sentences (Collins and Koo 2005) have shown that if, instead of taking the most likely tree structure generated by a parser, the n-best parse trees are passed through a discriminative re-ranking module, the accuracy of the model will increase significantly. We use the same idea to improve the performance of our model. We run a *Support Vector Machine (SVM)* based re-ranking module on top of the parser. Several contextual features (such as bigrams) are defined to help in disambiguation. This combination provides a framework that benefits from the advantages of both generative and discriminative models. In particular, when there is no or a very small amount of labeled data, a parser could still work by using unsupervised learning approaches to learn the rules, or by simply using a set of hand-built rules (as we did above for the task of semantic tagging). When there is enough labeled data, then a discriminative model can be trained on the labeled data to learn contextual information and to further enhance the tagging performance.

7 Evaluation

Our resources are a set of 21000 manually labeled queries, a manually designed grammar, a lexicon for every tag (except *Type* and *Attribute*), and a set of regular expressions defined for *Models* and *Attributes*. Note that with a grammar similar to the one in figure (6), generating a parse tree from a labeled query is straightforward. Then the parser is trained on the trees to learn the parameters of the model (probabilities in this case). We randomly extracted 3000, out of 21000, queries as the test set and used the remaining 18000 for training. We created training sets with different sizes to evaluate the impact of training data size on tagging performance.

Three modules were used in the evaluation: the *CRF-based* model[4], the *parser*, and the *parser plus the SVM*-based re-ranking. Figure (8) shows the learning curve of the word-level F-score for all the three modules. As seen in this plot, when there is a small amount of training data, the parser performs better than the *CRF* module and *parser+SVM* module performs better than the other two. With a large amount of training data, the *CRF* and *parser* almost have the same performance. Once again the *parser+SVM* module

[4] The CRF module also uses the lexical resources and regular expressions. In fact, it applies a deterministic context free grammar to the query to find all the possible groupings of words into chunks and uses this information as a set of features in the system.

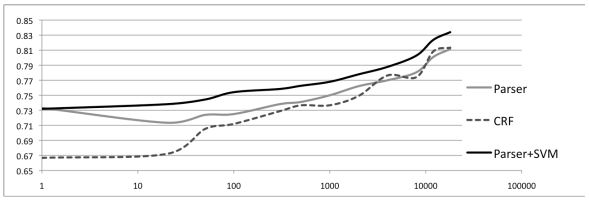

Figure 8. The learning curve for the three modules

outperforms the other two. These results show that, as expected, the *CRF*-based model is more dependent on the training data than the p*arser*. *Parser+SVM* always performs at least as well as the *parser*-only module even with a very small set of training data. This is because the rank given to every parse tree by the parser is used as a feature in the *SVM* module. When there is a very small amount of training data, this feature is dominant and the output of the re-reranking module is basically the same as the *parser*'s highest-rank output. Table (1) shows the performance of all three modules when the whole training set was used to train the system. The first three columns in the table show the word-level precision, recall, and F-score; and the last column represents the query level accuracy (a query is considered correct if all the words in the query have been labeled correctly). There are two rows for the *parser+SVM* in the table: one for *n=2* (i.e. re-ranking the 2-Best trees) and one for *n=10*. It is interesting to see that even with the re-ranking of only the first two trees generated by the *parser*, the difference between the accuracy of the *parser+SVM* module and the *parser*-only module is quite significant. Re-ranking with a larger number of trees (*n>10*) did not increase performance significantly.

8 Summary

We introduced a novel approach for deep parsing of web search queries. Our approach uses a grammar for generating multisets called a context-free multiset generating grammar (CFSG). We used a probabilistic version of this grammar. A parser was designed for parsing this type of grammar. Also a discriminative re-ranking module based on a support vector machine was used to take contextual information into account. We have used this system for automatic tagging of web search queries and have compared it with a CRF-based model designed for the same task.

Train No = 18000 Test No = 3000	P	R	F	Q
CRF	0.815	0.812	0.813	0.509
Parser	0.808	0.814	0.811	0.494
Parser+SVM (n = 2)	0.823	0.827	0.825	0.531
Parser+SVM (n = 10)	0.832	0.835	**0.833**	**0.555**

Table 1. The results of evaluating the three modules

The parser performs much better when there is a small amount of training data, but an adequate lexicon for every tag. This is a big advantage of the parser model, because in practice providing labeled data is very expensive but very often the lexicons can be easily extracted from the structured data on the web (for example extracting movie titles from *imdb* or book titles from *Amazon)*.

Our hybrid model (parser plus discriminative re-ranking), on the other hand, outperforms the other two modules regardless of the size of the training data.

The main drawback with our approach is to completely ignore the ordering. Note that although strict ordering constraints such as those imposed by PSG is not appropriate for modeling query structure, it might be helpful to take ordering information into account when resolving ambiguity. We leave this for future work. Another interesting and practically useful problem that we have left for future work is to design an unsupervised learning algorithm for CFSG similar to its phrase structure counterpart: *inside-outside* algorithm (Baker 1979). Having such a capability, we are able to automatically learn the underlying structure of queries by processing the huge amount of available unlabeled queries.

Acknowledgement

We need to thank Ye-Yi Wang for his helpful advices. We also thank William de Beaumont for his great comments on the paper.

References

Allan, J. and Raghavan, H. (2002) *Using Part-of-speech Patterns to Reduce Query Ambiguity*, Proceedings of SIGIR 2002, pp. 307-314.

Allen, J. F. (1995) *Natural Language Understanding*, Benjamin Cummings.

Baker, J. K. (1979) *Trainable grammars for speech recognition*. In Jared J. Wolf and Dennis H. Klatt, editors, Speech communication papers presented at the 97th Meeting of the Acoustical Society of America, MIT, Cambridge, MA.

Barton, E. (1985) *On the complexity of ID/LP rules*, Computational Linguistics, Volume 11, Pages 205-218.

Barr, C., Jones, R., Regelson, M., (2008) *The Linguistic Structure of English Web-Search Queries*, In Proceedings of EMNLP-08: conference on Empirical Methods in Natural Language Processing.

Broder, A., Fontoura, M., Gabrilovich, E., Joshi, A., Josifovski, V., and Zhang, T. (2007) *Robust classification of rare queries using web knowledge*. In Proceedings of SIGIR'07

Collins, M., Koo, T., (2005) *Discriminative Reranking for Natural Language Parsing*, Computational Linguistics, v.31 p.25-70.

Gazdar, G., Klein, E., Sag, I., Pullum, G., (1985) *Generalized Phrase Structure Grammar*, Harvard University Press.

Grenager, T., Klein, D., and Manning, C. (2005) *Unsupervised learning of field segmentation models for information extraction*, In Proceedings of ACL-05.

Kushmerick, N., Johnston, E., and McGuinness, S. (2001). *Information extraction by text classification*, In Proceedings of the IJCAI-01 Workshopon Adaptive Text Extraction and Mining.

Li, X., Wang, Y., and Acero, A. (2008) *Learning query intent from regularized click graphs*. In Proceedings of SIGIR'08

Manning, C., Schütze, H. (1999) *Foundations of Statistical Natural Language Processing*, The MIT Press, Cambridge, MA.

McCallum, A., Freitag, D., Pereira, F. (2000) *Maximum entropy markov models for information extraction and segmentation*, Proceedings of the Seventeenth International Conference on Machine Learning, Pages: 591 - 598

McCallum, A., Nigam, K., Rennie, J., and Seymore, K. (1999) *A machine learning approach to building domain-specific search engines*, In IJCAI-1999.

Pasca, M., Van Durme, B., and Garera, N. (2007) *The Role of Documents vs. Queries in Extracting Class Attributes from Text*, ACM Sixteenth Conference on Information and Knowledge Management (CIKM 2007). Lisboa, Portugal.

Viola, P., Narasimhan, M., *Learning to extract information from semi-structured text using a discriminative context free grammar* SIGIR 2005: 330-337.

Xue, GR, HJ Zeng, Z Chen, Y Yu, WY Ma, WS Xi, WG Fan, (2004), *Optimizing web search using web click-through data*, Proceedings of the thirteenth ACM international conference.