3

Extracting Relations from Text: From Word Sequences to Dependency Paths

Razvan C. Bunescu and Raymond J. Mooney

3.1 Introduction

Extracting semantic relationships between entities mentioned in text documents is an important task in natural language processing. The various types of relationships that are discovered between mentions of entities can provide useful structured information to a text mining system [1]. Traditionally, the task specifies a predefined set of entity types and relation types that are deemed to be relevant to a potential user and that are likely to occur in a particular text collection. For example, information extraction from newspaper articles is usually concerned with identifying mentions of people, organizations, locations, and extracting useful relations between them. Relevant relation types range from social relationships, to roles that people hold inside an organization, to relations between organizations, to physical locations of people and organizations. Scientific publications in the biomedical domain offer a type of narrative that is very different from the newspaper discourse. A significant effort is currently spent on automatically extracting relevant pieces of information from Medline, an online collection of biomedical abstracts. Proteins, genes, and cells are examples of relevant entities in this task, whereas subcellular localizations and protein-protein interactions are two of the relation types that have received significant attention recently. The inherent difficulty of the relation extraction task is further compounded in the biomedical domain by the relative scarcity of tools able to analyze the corresponding type of narrative. Most existing natural language processing tools, such as tokenizers, sentence segmenters, part-of-speech (POS) taggers, shallow or full parsers are trained on newspaper corpora, and consequently they incur a loss in accuracy when applied to biomedical literature. Therefore, information extraction systems developed for biological corpora need to be robust to POS or parsing errors, or to give reasonable performance using shallower but more reliable information, such as chunking instead of full parsing.

In this chapter, we present two recent approaches to relation extraction that differ in terms of the kind of linguistic information they use:

1. In the first method (Section 3.2), each potential relation is represented implicitly as a vector of features, where each feature corresponds to a *word sequence* anchored at the two entities forming the relationship. A relation extraction system

is trained based on the subsequence kernel from [2]. This kernel is further generalized so that words can be replaced with word classes, thus enabling the use of information coming from POS tagging, named entity recognition, chunking, or Wordnet [3].
2. In the second approach (Section 3.3), the representation is centered on the shortest *dependency path* between the two entities in the dependency graph of the sentence. Because syntactic analysis is essential in this method, its applicability is limited to domains where syntactic parsing gives reasonable accuracy.

Entity recognition, a prerequisite for relation extraction, is usually cast as a sequence tagging problem, in which words are tagged as being either outside any entity, or inside a particular type of entity. Most approaches to entity tagging are therefore based on probabilistic models for labeling sequences, such as Hidden Markov Models [4], Maximum Entropy Markov Models [5], or Conditional Random Fields [6], and obtain a reasonably high accuracy. In the two information extraction methods presented in this chapter, we assume that the entity recognition task was done and focus only on the relation extraction part.

3.2 Subsequence Kernels for Relation Extraction

One of the first approaches to extracting interactions between proteins from biomedical abstracts is that of Blaschke *et al.*, described in [7, 8]. Their system is based on a set of manually developed rules, where each rule (or frame) is a sequence of words (or POS tags) and two protein-name tokens. Between every two adjacent words is a number indicating the maximum number of intervening words allowed when matching the rule to a sentence. An example rule is "*interaction of (3) <P> (3) with (3) <P>*", where '<P>' is used to denote a protein name. A sentence matches the rule if and only if it satisfies the word constraints in the given order and respects the respective word gaps.

In [9] the authors described a new method ELCS (Extraction using Longest Common Subsequences) that automatically learns such rules. ELCS' rule representation is similar to that in [7, 8], except that it currently does not use POS tags, but allows disjunctions of words. An example rule learned by this system is "- *(7) interaction (0) [between | of] (5) <P> (9) <P> (17)* ." Words in square brackets separated by '|' indicate disjunctive lexical constraints, i.e., one of the given words must match the sentence at that position. The numbers in parentheses between adjacent constraints indicate the maximum number of unconstrained words allowed between the two.

3.2.1 Capturing Relation Patterns with a String Kernel

Both Blaschke and ELCS do relation extraction based on a limited set of matching rules, where a rule is simply a sparse (gappy) subsequence of words or POS tags anchored on the two protein-name tokens. Therefore, the two methods share a common limitation: either through manual selection (Blaschke), or as a result of a greedy learning procedure (ELCS), they end up using only a subset of all possible anchored sparse subsequences. Ideally, all such anchored sparse subsequences would be used as features, with weights reflecting their relative accuracy. However,

explicitly creating for each sentence a vector with a position for each such feature is infeasible, due to the high dimensionality of the feature space. Here, we exploit dual learning algorithms that process examples only via computing their dot-products, such as in Support Vector Machines (SVMs) [10, 11]. An SVM learner tries to find a hyperplane that separates positive from negative examples and at the same time maximizes the separation (margin) between them. This type of max-margin separator has been shown both theoretically and empirically to resist overfitting and to provide good generalization performance on unseen examples.

Computing the dot-product (i.e., the kernel) between the features vectors associated with two relation examples amounts to calculating the number of common anchored subsequences between the two sentences. This is done efficiently by modifying the dynamic programming algorithm used in the string kernel from [2] to account only for common sparse subsequences constrained to contain the two protein-name tokens. The feature space is further prunned down by utilizing the following property of natural language statements: when a sentence asserts a relationship between two entity mentions, it generally does this using one of the following four patterns:

• **[FB] F**ore–**B**etween: words before and between the two entity mentions are simultaneously used to express the relationship. Examples: 'interaction of $\langle P_1 \rangle$ with $\langle P_2 \rangle$,' 'activation of $\langle P_1 \rangle$ by $\langle P_2 \rangle$.'

• **[B] B**etween: only words between the two entities are essential for asserting the relationship. Examples: '$\langle P_1 \rangle$ interacts with $\langle P_2 \rangle$,' '$\langle P_1 \rangle$ is activated by $\langle P_2 \rangle$.'

• **[BA] B**etween–**A**fter: words between and after the two entity mentions are simultaneously used to express the relationship. Examples: '$\langle P_1 \rangle$ – $\langle P_2 \rangle$ complex,' '$\langle P_1 \rangle$ and $\langle P_2 \rangle$ interact.'

• **[M] M**odifier: the two entity mentions have no words between them. Examples: *U.S. troops* (a ROLE:STAFF relation), *Serbian general* (ROLE:CITIZEN).

While the first three patterns are sufficient to capture most cases of interactions between proteins, the last pattern is needed to account for various relationships expressed through noun-noun or adjective-noun compounds in the newspaper corpora.

Another observation is that all these patterns use at most four words to express the relationship (not counting the two entity names). Consequently, when computing the relation kernel, we restrict the counting of common anchored subsequences only to those having one of the four types described above, with a maximum word-length of four. This type of feature selection leads not only to a faster kernel computation, but also to less overfitting, which results in increased accuracy.

The patterns enumerated above are completely lexicalized and consequently their performance is limited by data sparsity. This can be alleviated by categorizing words into classes with varying degrees of generality, and then allowing patterns to use both words and their classes. Examples of word classes are POS tags and generalizations over POS tags such as Noun, Active Verb, or Passive Verb. The entity type can also be used if the word is part of a known named entity. Also, if the sentence is segmented into syntactic chunks such as noun phrases (NP) or verb phrases (VP), the system may choose to consider only the head word from each chunk, together with the type of the chunk as another word class. Content words such as nouns and verbs can also be related to their synsets via WordNet. Patterns then will consist of sparse subsequences of words, POS tags, generalized POS tags, entity and chunk types, or WordNet synsets. For example, 'Noun of $\langle P_1 \rangle$ by $\langle P_2 \rangle$' is an FB pattern based on words and general POS tags.

3.2.2 A Generalized Subsequence Kernel

Let $\Sigma_1, \Sigma_2, ..., \Sigma_k$ be some disjoint feature spaces. Following the example in Section 3.2.1, Σ_1 could be the set of words, Σ_2 the set of POS tags, etc. Let $\Sigma_\times = \Sigma_1 \times \Sigma_2 \times ... \times \Sigma_k$ be the set of all possible feature vectors, where a feature vector would be associated with each position in a sentence. Given two feature vectors $x, y \in \Sigma_\times$, let $c(x, y)$ denote the number of common features between x and y. The next notation follows that introduced in [2]. Thus, let s, t be two sequences over the finite set $\Sigma_\times$, and let $|s|$ denote the length of $s = s_1...s_{|s|}$. The sequence $s[i{:}j]$ is the contiguous subsequence $s_i...s_j$ of s. Let $\mathbf{i} = (i_1, ..., i_{|\mathbf{i}|})$ be a sequence of $|\mathbf{i}|$ indices in s, in ascending order. We define the length $l(\mathbf{i})$ *of the index sequence* $\mathbf{i}$ *in* s as $i_{|\mathbf{i}|} - i_1 + 1$. Similarly, $\mathbf{j}$ is a sequence of $|\mathbf{j}|$ indices in t.

Let $\Sigma_\cup = \Sigma_1 \cup \Sigma_2 \cup ... \cup \Sigma_k$ be the set of all possible features. We say that the sequence $u \in \Sigma_\cup^*$ is a (sparse) subsequence of s if there is a sequence of $|u|$ indices $\mathbf{i}$ such that $u_k \in s_{i_k}$, for all $k = 1, ..., |u|$. Equivalently, we write $u \prec s[\mathbf{i}]$ as a shorthand for the component-wise '$\in$' relationship between u and $s[\mathbf{i}]$.

Finally, let $K_n(s, t, \lambda)$ (Equation 3.1) be the number of weighted sparse subsequences u of length n common to s and t (i.e., $u \prec s[\mathbf{i}]$, $u \prec t[\mathbf{j}]$), where the weight of u is $\lambda^{l(\mathbf{i})+l(\mathbf{j})}$, for some $\lambda \leq 1$.

$$K_n(s, t, \lambda) = \sum_{u \in \Sigma_\cup^n} \sum_{\mathbf{i}: u \prec s[\mathbf{i}]} \sum_{\mathbf{j}: u \prec t[\mathbf{j}]} \lambda^{l(\mathbf{i})+l(\mathbf{j})} \tag{3.1}$$

Let $\mathbf{i}$ and $\mathbf{j}$ be two index sequences of length n. By definition, for every k between 1 and n, $c(s_{i_k}, t_{j_k})$ returns the number of common features between s and t at positions i_k and j_k. If $c(s_{i_k}, t_{j_k}) = 0$ for some k, there are no common feature sequences of length n between $s[\mathbf{i}]$ and $t[\mathbf{j}]$. On the other hand, if $c(s_{i_k}, t_{j_k})$ is greater than 1, this means that there is more than one common feature that can be used at position k to obtain a common feature sequence of length n. Consequently, the number of common feature sequences of length n between $s[\mathbf{i}]$ and $t[\mathbf{j}]$, i.e., the size of the set $\{u \in \Sigma_\cup^n | u \prec s[\mathbf{i}], u \prec t[\mathbf{j}]\}$, is given by $\prod_{k=1}^{n} c(s_{i_k}, t_{j_k})$. Therefore, $K_n(s, t, \lambda)$ can be rewritten as in Equation 3.2:

$$K_n(s, t, \lambda) = \sum_{\mathbf{i}: |\mathbf{i}|=n} \sum_{\mathbf{j}: |\mathbf{j}|=n} \prod_{k=1}^{n} c(s_{i_k}, t_{j_k}) \lambda^{l(\mathbf{i})+l(\mathbf{j})} \tag{3.2}$$

We use λ as a decaying factor that penalizes longer subsequences. For sparse subsequences, this means that wider gaps will be penalized more, which is exactly the desired behavior for our patterns. Through them, we try to capture head-modifier dependencies that are important for relation extraction; for lack of reliable dependency information, the larger the word gap is between two words, the less confident we are in the existence of a head-modifier relationship between them.

To enable an efficient computation of K_n, we use the auxiliary function $K_n^{'}$ with a definition similar to K_n, the only difference being that it counts the length from the beginning of the particular subsequence u to the end of the strings s and t, as illustrated in Equation 3.3:

$$K_n^{'}(s, t, \lambda) = \sum_{u \in \Sigma_\cup^n} \sum_{\mathbf{i}: u \prec s[\mathbf{i}]} \sum_{\mathbf{j}: u \prec t[\mathbf{j}]} \lambda^{|s|+|t|-i_1-j_1+2} \tag{3.3}$$

An equivalent formula for $K_n^{'}(s,t,\lambda)$ is obtained by changing the exponent of λ from Equation 3.2 to $|s|+|t|-i_1-j_1+2$.

Based on all definitions above, K_n is computed in $O(kn|s||t|)$ time, by modifying the recursive computation from [2] with the new factor $c(x,y)$, as shown in Figure 3.1. As in [2], the complexity of computing $K_i^{'}(s,t)$ is reduced to $O(|s||t|)$ by first evaluating another auxiliary factor $K_i^{''}(s,t)$. In Figure 3.1, the sequence sx is the result of appending x to s (with ty defined in a similar way). To avoid clutter, the parameter λ is not shown in the argument list of K and K', unless it is instantiated to a specific constant.

$$
\begin{aligned}
K_0^{'}(s,t) &= 1,\ for\ all\ s,t \\
K_i^{'}(s,t) &= 0,\ if\ \min(|s|,|t|) < i \\
K_i^{''}(s,\emptyset) &= 0,\ for\ all\ i,s \\
K_i^{''}(sx,ty) &= \lambda K_i^{''}(sx,t) + \lambda^2 K_{i-1}^{'}(s,t)\cdot c(x,y) \\
K_i^{'}(sx,t) &= \lambda K_i^{'}(s,t) + K_i^{''}(sx,t) \\
K_n(s,t) &= 0,\ if\ \min(|s|,|t|) < n \\
K_n(sx,t) &= K_n(s,t) + \sum_j \lambda^2 K_{n-1}^{'}(s,t[1:j-1])\cdot c(x,t[j])
\end{aligned}
$$

Fig. 3.1. Computation of subsequence kernel.

3.2.3 Computing the Relation Kernel

As described at the beginning of Section 3.2, the input consists of a set of sentences, where each sentence contains exactly two entities (protein names in the case of interaction extraction). In Figure 3.2 we show the segments that will be used for computing the relation kernel between two example sentences s and t. In sentence s, for instance, x_1 and x_2 are the two entities, s_f is the sentence segment before x_1, s_b is the segment between x_1 and x_2, and s_a is the sentence segment after x_2. For convenience, we also include the auxiliary segment $s_b^{'} = x_1 s_b x_2$, whose span is computed as $l(s_b^{'}) = l(s_b) + 2$ (in all length computations, we consider x_1 and x_2 as contributing one unit only).

The relation kernel computes the number of common patterns between two sentences s and t, where the set of patterns is restricted to the four types introduced in Section 3.2.1. Therefore, the kernel $rK(s,t)$ is expressed as the sum of four subkernels: $fbK(s,t)$ counting the number of common fore–between patterns, $bK(s,t)$ for between patterns, $baK(s,t)$ for between–after patterns, and $mK(s,t)$ for modifier patterns, as in Figure 3.3. The symbol $\mathbb{1}$ is used there as a shorthand for the indicator function, which is 1 if the argument is true, and 0 otherwise.

The first three sub-kernels include in their computation the counting of common subsequences between $s_b^{'}$ and $t_b^{'}$. In order to speed up the computation, all these

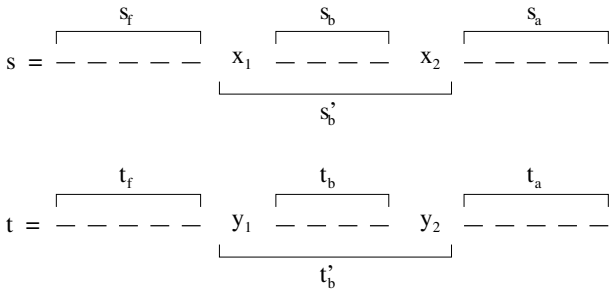


Fig. 3.2. Sentence segments.

$$
\begin{aligned}
rK(s,t) &= fbK(s,t) + bK(s,t) + baK(s,t) + mK(s,t) \\
bK_i(s,t) &= K_i(s_b, t_b, 1) \cdot c(x_1, y_1) \cdot c(x_2, y_2) \cdot \lambda^{l(s_b^{'})+l(t_b^{'})} \\
fbK(s,t) &= \sum_{i,j} bK_i(s,t) \cdot K_j^{'}(s_f, t_f), \quad 1 \le \mathrm{i},\ 1 \le \mathrm{j},\ \mathrm{i}+\mathrm{j} < \mathrm{fb}_{\max} \\
bK(s,t) &= \sum_{i} bK_i(s,t), \quad 1 \le \mathrm{i} \le \mathrm{b}_{\max} \\
baK(s,t) &= \sum_{i,j} bK_i(s,t) \cdot K_j^{'}(s_a^{-}, t_a^{-}), \quad 1 \le \mathrm{i},\ 1 \le \mathrm{j},\ \mathrm{i}+\mathrm{j} < \mathrm{ba}_{\max} \\
mK(s,t) &= \mathbb{1}(s_b = \emptyset) \cdot \mathbb{1}(t_b = \emptyset) \cdot c(x_1, y_1) \cdot c(x_2, y_2) \cdot \lambda^{2+2},
\end{aligned}
$$

Fig. 3.3. Computation of relation kernel.

common counts are calculated separately in bK_i, which is defined as the number of common subsequences of length i between $s_b^{'}$ and $t_b^{'}$, anchored at x_1/x_2 and y_1/y_2 respectively (i.e., constrained to start at x_1 in $s_b^{'}$ and y_1 in $t_b^{'}$, and to end at x_2 in $s_b^{'}$ and y_2 in $t_b^{'}$). Then fbK simply counts the number of subsequences that match j positions before the first entity and i positions between the entities, constrained to have length less than a constant fb_{max}. To obtain a similar formula for baK we simply use the reversed (mirror) version of segments s_a and t_a (e.g., s_a^- and t_a^-). In Section 3.2.1 we observed that all three subsequence patterns use at most 4 words to express a relation, therefore the constants fb_{max}, b_{max} and ba_{max} are set to 4. Kernels K and $K^{'}$ are computed using the procedure described in Section 3.2.2.

3.3 A Dependency-Path Kernel for Relation Extraction

The pattern examples from Section 3.2.1 show the two entity mentions, together with the set of words that are relevant for their relationship. A closer analysis of

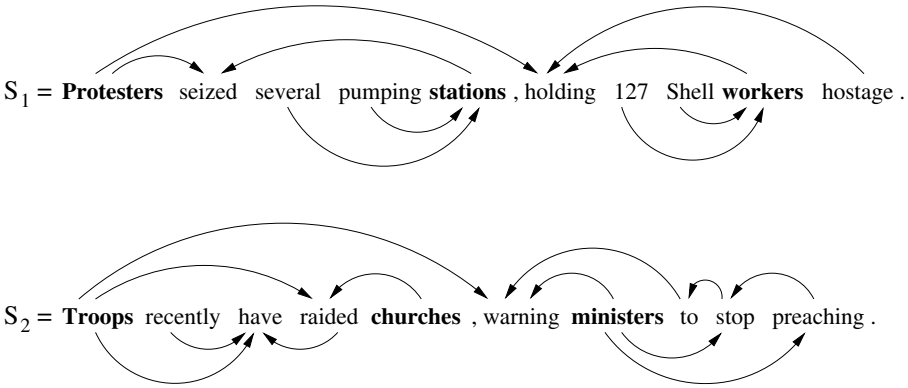


Fig. 3.4. Sentences as dependency graphs.

these examples reveals that all relevant words form a shortest path between the two entities in a graph structure where edges correspond to relations between a word (head) and its dependents. For example, Figure 3.4 shows the full dependency graphs for two sentences from the ACE (Automated Content Extraction) newspaper corpus [12], in which words are represented as nodes and word-word dependencies are represented as directed edges. A subset of these word-word dependencies capture the predicate-argument relations present in the sentence. Arguments are connected to their target predicates either directly through an arc pointing to the predicate ('troops → raided'), or indirectly through a preposition or infinitive particle ('warning ← to ← stop'). Other types of word-word dependencies account for modifier-head relationships present in adjective-noun compounds ('several → stations'), noun-noun compounds ('pumping → stations'), or adverb-verb constructions ('recently → raided').

Word-word dependencies are typically categorized in two classes as follows:

- [**Local Dependencies**] These correspond to local predicate-argument (or head-modifier) constructions such as 'troops → raided', or 'pumping → stations' in Figure 3.4.
- [**Non-local Dependencies**] Long-distance dependencies arise due to various linguistic constructions such as coordination, extraction, raising and control. In Figure 3.4, among non-local dependencies are 'troops → warning', or 'ministers → preaching'.

A Context Free Grammar (CFG) parser can be used to extract local dependencies, which for each sentence form a dependency tree. Mildly context sensitive formalisms such as Combinatory Categorial Grammar (CCG) [13] model word-word dependencies more directly and can be used to extract both local and long-distance dependencies, giving rise to a directed acyclic graph, as illustrated in Figure 3.4.

3.3.1 The Shortest Path Hypothesis

If e_1 and e_2 are two entities mentioned in the same sentence such that they are observed to be in a relationship R, then the contribution of the sentence dependency

Table 3.1. Shortest Path representation of relations.

Relation Instance	Shortest Path in Undirected Dependency Graph
S_1:protesters AT stations	**protesters** → seized ← **stations**
S_1:workers AT stations	**workers** → holding ← protesters → seized ← **stations**
S_2:troops AT churches	**troops** → raided ← **churches**
S_2:ministers AT churches	**ministers** → warning ← troops → raided ← **churches**

graph to establishing the relationship $R(e_1, e_2)$ is almost exclusively concentrated in the shortest path between e_1 and e_2 in the undirected version of the dependency graph.

If entities e_1 and e_2 are arguments of the same predicate, then the shortest path between them will pass through the predicate, which may be connected directly to the two entities, or indirectly through prepositions. If e_1 and e_2 belong to different predicate-argument structures that share a common argument, then the shortest path will pass through this argument. This is the case with the shortest path between 'stations' and 'workers' in Figure 3.4, passing through 'protesters,' which is an argument common to both predicates 'holding' and 'seized'. In Table 3.1, we show the paths corresponding to the four relation instances encoded in the ACE corpus for the two sentences from Figure 3.4. All these paths support the LOCATED relationship. For the first path, it is reasonable to infer that if a PERSON entity (e.g., 'protesters') is doing some action (e.g., 'seized') to a FACILITY entity (e.g., 'station'), then the PERSON entity is LOCATED at that FACILITY entity. The second path captures the fact that the same PERSON entity (e.g., 'protesters') is doing two actions (e.g., 'holding' and 'seized') , one action to a PERSON entity (e.g., 'workers'), and the other action to a FACILITY entity (e.g., 'station'). A reasonable inference in this case is that the 'workers' are LOCATED at the 'station'.

In Figure 3.5, we show three more examples of the LOCATED (AT) relationship as dependency paths created from one or two predicate-argument structures. The second example is an interesting case, as it illustrates how annotation decisions are accommodated in our approach. Using a reasoning similar with that from the previous paragraph, it is reasonable to infer that 'troops' are LOCATED in 'vans,' and that 'vans' are LOCATED in 'city'. However, because 'vans' is not an ACE markable, it cannot participate in an annotated relationship. Therefore, 'troops' is annotated as being LOCATED in 'city,' which makes sense due to the transitivity of the relation LOCATED. In our approach, this leads to shortest paths that pass through two or more predicate-argument structures.

The last relation example is a case where there exist multiple shortest paths in the dependency graph between the same two entities – there are actually two different paths, with each path replicated into three similar paths due to coordination. Our current approach considers only one of the shortest paths, nevertheless it seems reasonable to investigate using all of them as multiple sources of evidence for relation extraction.

There may be cases where e_1 and e_2 belong to predicate-argument structures that have no argument in common. However, because the dependency graph is always connected, we are guaranteed to find a shortest path between the two entities. In general, we shall find a shortest sequence of predicate-argument structures with

target predicates $P_1, P_2, ..., P_n$ such that e_1 is an argument of P_1, e_2 is an argument of P_n, and any two consecutive predicates P_i and P_{i+1} share a common argument (where by "argument" we mean both arguments and complements).

(1) He had no regrets for **his** actions in **Brcko**.

his → actions ← in ← **Brcko**

(2) U.S. **troops** today acted for the first time to capture an alleged Bosnian war criminal, rushing from unmarked vans parked in the northern Serb-dominated **city** of Bijeljina.

troops → rushing ← from ← vans → parked ← in ← **city**

(3) Jelisic created an atmosphere of terror at the **camp** by killing, abusing and threatening the **detainees**.

detainees → killing ← Jelisic → created ← at ← **camp**
detainees → abusing ← Jelisic → created ← at ← **camp**
detainees → threatning ← Jelisic → created ← at ← **camp**
detainees → killing → by → created ← at ← **camp**
detainees → abusing → by → created ← at ← **camp**
detainees → threatening → by → created ← at ← **camp**

Fig. 3.5. Relation examples.

3.3.2 Learning with Dependency Paths

The shortest path between two entities in a dependency graph offers a very condensed representation of the information needed to assess their relationship. A dependency path is represented as a sequence of words interspersed with arrows that indicate the orientation of each dependency, as illustrated in Table 3.1. These paths, however, are completely lexicalized and consequently their performance will be limited by data sparsity. The solution is to allow paths to use both words and their word classes, similar with the approach taken for the subsequence patterns in Section 3.2.1.

The set of features can then be defined as a Cartesian product over words and word classes, as illustrated in Figure 3.6 for the dependency path between 'protesters' and 'station' in sentence S_1. In this representation, sparse or contiguous subsequences of nodes along the lexicalized dependency path (i.e., path fragments) are included as features simply by replacing the rest of the nodes with their corresponding generalizations.

Examples of features generated by Figure 3.6 are "protesters → seized ← stations," "Noun → Verb ← Noun," "PERSON → seized ← FACILITY," or "PERSON → Verb ← FACILITY." The total number of features generated by this dependency path is $4 \times 1 \times 3 \times 1 \times 4$.

$$\begin{bmatrix} \text{protesters} \\ \text{NNS} \\ \text{Noun} \\ \text{PERSON} \end{bmatrix} \times [\rightarrow] \times \begin{bmatrix} \text{seized} \\ \text{VBD} \\ \text{Verb} \end{bmatrix} \times [\leftarrow] \times \begin{bmatrix} \text{stations} \\ \text{NNS} \\ \text{Noun} \\ \text{FACILITY} \end{bmatrix}$$

Fig. 3.6. Feature generation from dependency path.

For verbs and nouns (and their respective word classes) occurring along a dependency path we also use an additional suffix '(-)' to indicate a negative polarity item. In the case of verbs, this suffix is used when the verb (or an attached auxiliary) is modified by a negative polarity adverb such as 'not' or 'never.' Nouns get the negative suffix whenever they are modified by negative determiners such as 'no,' 'neither' or 'nor.' For example, the phrase "He never went to Paris" is associated with the dependency path "He $\rightarrow$ went(-) $\leftarrow$ to $\leftarrow$ Paris."

As in Section 3.2, we use kernel SVMs in order to avoid working explicitly with high-dimensional dependency path feature vectors. Computing the dot-product (i.e., kernel) between two relation examples amounts to calculating the number of common features (i.e., paths) between the two examples. If $\boldsymbol{x} = x_1 x_2 ... x_m$ and $\boldsymbol{y} = y_1 y_2 ... y_n$ are two relation examples, where x_i denotes the set of word classes corresponding to position i (as in Figure 3.6), then the number of common features between $\boldsymbol{x}$ and $\boldsymbol{y}$ is computed as in Equation 3.4.

$$K(\mathbf{x}, \mathbf{y}) = \mathbb{1}(m = n) \cdot \prod_{i=1}^{n} c(x_i, y_i) \tag{3.4}$$

where $c(x_i, y_i) = |x_i \cap y_i|$ is the number of common word classes between x_i and y_i.

This is a simple kernel, whose computation takes $O(n)$ time. If the two paths have different lengths, they correspond to different ways of expressing a relationship – for instance, they may pass through a different number of predicate argument structures. Consequently, the kernel is defined to be 0 in this case. Otherwise, it is the product of the number of common word classes at each position in the two paths. As an example, let us consider two instances of the LOCATED relationship, and their corresponding dependency paths:

1. '**his** actions in **Brcko**' (**his** $\rightarrow$ actions $\leftarrow$ in $\leftarrow$ **Brcko**).
2. '**his** arrival in **Beijing**' (**his** $\rightarrow$ arrival $\leftarrow$ in $\leftarrow$ **Beijing**).

Their representation as a sequence of sets of word classes is given by:

1. $\boldsymbol{x} = [x_1\ x_2\ x_3\ x_4\ x_5\ x_6\ x_7]$, where $x_1 = \{$his, PRP, PERSON$\}$, $x_2 = \{\rightarrow\}$, $x_3 = \{$actions, NNS, Noun$\}$, $x_4 = \{\leftarrow\}$, $x_5 = \{$in, IN$\}$, $x_6 = \{\leftarrow\}$, $x_7 = \{$Brcko, NNP, Noun, LOCATION$\}$
2. $\boldsymbol{y} = [y_1\ y_2\ y_3\ y_4\ y_5\ y_6\ y_7]$, where $y_1 = \{$his, PRP, PERSON$\}$, $y_2 = \{\rightarrow\}$, $y_3 = \{$arrival, NN, Noun$\}$, $y_4 = \{\leftarrow\}$, $y_5 = \{$in, IN$\}$, $y_6 = \{\leftarrow\}$, $y_7 = \{$Beijing, NNP, Noun, LOCATION$\}$

Based on the formula from Equation 3.4, the kernel is computed as $K(\boldsymbol{x}, \boldsymbol{y}) = 3 \times 1 \times 1 \times 1 \times 2 \times 1 \times 3 = 18$.

3.4 Experimental Evaluation

The two relation kernels described above are evaluated on the task of extracting relations from two corpora with different types of narrative, which are described in more detail in the following sections. In both cases, we assume that the entities and their labels are known. All preprocessing steps – sentence segmentation, tokenization, POS tagging, and chunking – were performed using the OpenNLP[1] package. If a sentence contains n entities ($n \geq 2$), it is replicated into $\binom{n}{2}$ sentences, each containing only two entities. If the two entities are known to be in a relationship, then the replicated sentence is added to the set of corresponding positive sentences, otherwise it is added to the set of negative sentences. During testing, a sentence having n entities ($n \geq 2$) is again replicated into $\binom{n}{2}$ sentences in a similar way.

The dependency graph that is input to the shortest path dependecy kernel is obtained from two different parsers:

- The CCG parser introduced in [14][2] outputs a list of functor-argument dependencies, from which head-modifier dependencies are obtained using a straightforward procedure (for more details, see [15]).
- Head-modifier dependencies can be easily extracted from the full parse output of Collins' CFG parser [16], in which every non-terminal node is annotated with head information.

The relation kernels are used in conjunction with SVM learning in order to find a decision hyperplane that best separates the positive examples from negative examples. We modified the LibSVM[3] package by plugging in the kernels described above. The factor λ in the subsequence kernel is set to 0.75. The performance is measured using *precision* (percentage of correctly extracted relations out of the total number of relations extracted), *recall* (percentage of correctly extracted relations out of the total number of relations annotated in the corpus), and *F-measure* (the harmonic mean of *precision* and *recall*).

3.4.1 Interaction Extraction from AIMed

We did comparative experiments on the AIMed corpus, which has been previously used for training the protein interaction extraction systems in [9]. It consists of 225 Medline abstracts, of which 200 are known to describe interactions between human proteins, while the other 25 do not refer to any interaction. There are 4084 protein references and around 1000 tagged interactions in this dataset.

The following systems are evaluated on the task of retrieving protein interactions from AIMed (assuming gold standard proteins):

• **[Manual]**: We report the performance of the rule-based system of [7, 8].

• **[ELCS]**: We report the 10-fold cross-validated results from [9] as a Precision-Recall (PR) graph.

• **[SSK]**: The subseqeuence kernel is trained and tested on the same splits as ELCS. In order to have a fair comparison with the other two systems, which use only lexical information, we do not use any word classes here.

[1] URL: http://opennlp.sourceforge.net

[2] URL:http://www.ircs.upenn.edu/~juliahr/Parser/

[3] URL:http://www.csie.ntu.edu.tw/~cjlin/libsvm/

• **[SPK]**: This is the shortest path dependency kernel, using the head-modifier dependencies extracted by Collins' syntactic parser. The kernel is trained and tested on the same 10 splits as ELCS and SSK.

The Precision-Recall curves that show the trade-off between these metrics are obtained by varying a threshold on the minimum acceptable extraction confidence, based on the probability estimates from LibSVM. The results, summarized in Figure 3.7, show that the subsequence kernel outperforms the other three systems, with a substantial gain. The syntactic parser, which is originally trained on a newspaper corpus, builds less accurate dependency structures for the biomedical text. This is reflected in a significantly reduced accuracy for the dependency kernel.

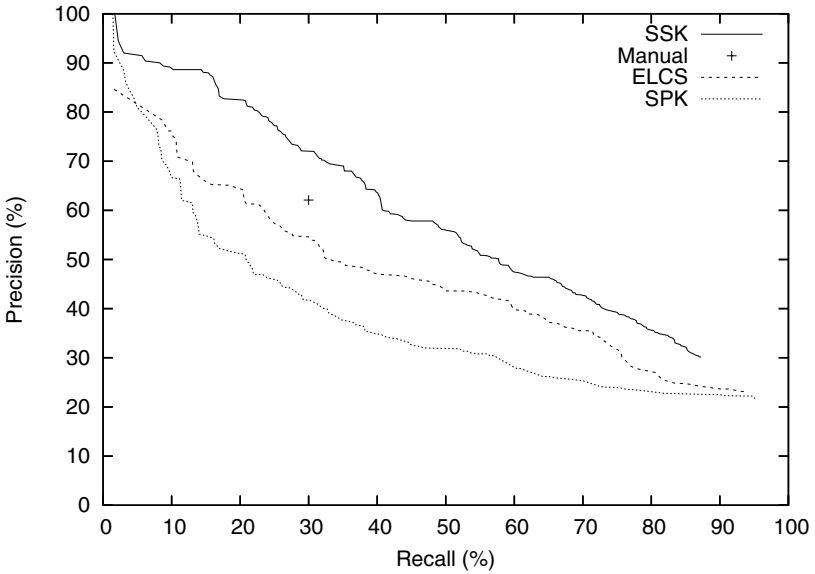


Fig. 3.7. Precision-Recall curves for protein interaction extractors.

3.4.2 Relation Extraction from ACE

The two kernels are also evaluated on the task of extracting top-level relations from the ACE corpus [12], the version used for the September 2002 evaluation. The training part of this dataset consists of 422 documents, with a separate set of 97 documents reserved for testing. This version of the ACE corpus contains three types of annotations: coreference, named entities and relations. There are five types of entities – Person, Organization, Facility, Location, and Geo-Political Entity – which can participate in five general, top-level relations: Role, Part, Located, Near, and Social. In total, there are 7,646 intra-sentential relations, of which 6,156 are in the training data and 1,490 in the test data.

A recent approach to extracting relations is described in [17]. The authors use a generalized version of the tree kernel from [18] to compute a kernel over relation examples, where a relation example consists of the smallest dependency tree containing the two entities of the relation. Precision and recall values are reported for the task of extracting the five top-level relations in the ACE corpus under two different scenarios:

– **[S1]** This is the classic setting: one multi-class SVM is learned to discriminate among the five top-level classes, plus one more class for the no-relation cases.

– **[S2]** One binary SVM is trained for *relation detection*, meaning that all positive relation instances are combined into one class. The thresholded output of this binary classifier is used as training data for a second multi-class SVM, trained for *relation classification*.

The subsequence kernel (SSK) is trained under the first scenario, to recognize the same five top-level relation types. While for protein interaction extraction only the lexicalized version of the kernel was used, here we utilize more features, corresponding to the following feature spaces: Σ_1 is the word vocabulary, Σ_2 is the set of POS tags, Σ_3 is the set of generic POS tags, and Σ_4 contains the five entity types. Chunking information is used as follows: all (sparse) subsequences are created exclusively from the chunk heads, where a head is defined as the last word in a chunk. The same criterion is used for computing the length of a subsequence – all words other than head words are ignored. This is based on the observation that in general words other than the chunk head do not contribute to establishing a relationship between two entities outside of that chunk. One exception is when both entities in the example sentence are contained in the same chunk. This happens very often due to noun-noun ('U.S. troops') or adjective-noun ('Serbian general') compounds. In these cases, the chunk is allowed to contribute both entity heads.

The shortest-path dependency kernel (SPK) is trained under both scenarios. The dependencies are extracted using either Hockenmaier's CCG parser (SPK-CCG) [14], or Collins' CFG parser (SPK-CFG) [16].

Table 3.2 summarizes the performance of the two relation kernels on the ACE corpus. For comparison, we also show the results presented in [17] for their best performing kernel K4 (a sum between a bag-of-words kernel and a tree dependency kernel) under both scenarios.

Table 3.2. Extraction Performance on ACE.

(Scenario) Method	Precision	Recall	F-measure
(S1) K4	70.3	26.3	38.0
(S1) SSK	73.9	35.2	47.7
(S1) SPK-CCG	67.5	37.2	48.0
(S1) SPK-CFG	**71.1**	**39.2**	**50.5**
(S2) K4	67.1	35.0	45.8
(S2) SPK-CCG	63.7	41.4	50.2
(S2) SPK-CFG	**65.5**	**43.8**	**52.5**

The shortest-path dependency kernels outperform the dependency kernel from [17] in both scenarios, with a more substantial gain for SP-CFG. An error analysis revealed that Collins' parser was better at capturing local dependencies, hence the increased accuracy of SP-CFG. Another advantage of shortest-path dependency kernels is that their training and testing are very fast – this is due to representing the sentence as a chain of dependencies on which a fast kernel can be computed. All of the four SP kernels from Table 3.2 take between 2 and 3 hours to train and test on a 2.6GHz Pentium IV machine.

As expected, the newspaper articles from ACE are less prone to parsing errors than the biomedical articles from AIMed. Consequently, the extracted dependency structures are more accurate, leading to an improved accuracy for the dependency kernel.

To avoid numerical problems, the dependency paths are constrained to pass through at most 10 words (as observed in the training data) by setting the kernel to 0 for longer paths. The alternative solution of normalizing the kernel leads to a slight decrease in accuracy. The fact that longer paths have larger kernel scores in the unnormalized version does not pose a problem because, by definition, paths of different lengths correspond to disjoint sets of features. Consequently, the SVM algorithm will induce lower weights for features occurring in longer paths, resulting in a linear separator that works irrespective of the size of the dependency paths.

3.5 Future Work

There are cases when words that do not belong to the shortest dependency path do influence the extraction decision. In Section 3.3.2, we showed how negative polarity items are integrated in the model through annotations of words along the dependency paths. Modality is another phenomenon that is influencing relation extraction, and we plan to incorporate it using the same annotation approach.

The two relation extraction methods are very similar: the subsequence patterns in one kernel correspond to dependency paths in the second kernel. More exactly, pairs of words from a subsequence pattern correspond to pairs of consecutive words (i.e., edges) on the dependency path. The lack of dependency information in the subsequence kernel leads to allowing gaps between words, with the corresponding exponential penalty factor λ. Given the observed similarity between the two methods, it seems reasonable to use them both in an integrated model. This model would use high-confidence head-modifier dependencies, falling back on pairs of words with gaps, when the dependency information is unreliable.

3.6 Conclusion

Mining knowledge from text documents can benefit from using the structured information that comes from entity recognition and relation extraction. However, accurately extracting relationships between relevant entities is dependent on the granularity and reliability of the required linguistic analysis. In this chapter, we presented two relation extraction kernels that differ in terms of the amount of linguistic information they use. Experimental evaluations on two corpora with different types of discourse show that they compare favorably to previous extraction approaches.

3.7 Acknowledgment

This work was supported by grants IIS-0117308 and IIS-0325116 from the NSF. We would like to thank Arun Ramani and Edward Marcotte for their help in preparing the AIMed corpus.

References

1. R. J. Mooney, R. C. Bunescu, Mining knowledge from text using information extraction, SIGKDD Explorations (special issue on Text Mining and Natural Language Processing) 7 (1) (2005) 3–10.
2. H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, C. Watkins, Text classification using string kernels, Journal of Machine Learning Research 2 (2002) 419–444.
3. C. D. Fellbaum, WordNet: An Electronic Lexical Database, MIT Press, Cambridge, MA, 1998.
4. L. R. Rabiner, A tutorial on hidden Markov models and selected applications in speech recognition, Proceedings of the IEEE 77 (2) (1989) 257–286.
5. A. McCallum, D. Freitag, F. Pereira, Maximum entropy Markov models for information extraction and segmentation, in: Proceedings of the Seventeenth International Conference on Machine Learning (ICML-2000), Stanford, CA, 2000.
6. J. Lafferty, A. McCallum, F. Pereira, Conditional random fields: Probabilistic models for segmenting and labeling sequence data, in: Proceedings of 18th International Conference on Machine Learning (ICML-2001), Williamstown, MA, 2001, pp. 282–289.
7. C. Blaschke, A. Valencia, Can bibliographic pointers for known biological data be found automatically? protein interactions as a case study, Comparative and Functional Genomics 2 (2001) 196–206.
8. C. Blaschke, A. Valencia, The frame-based module of the Suiseki information extraction system, IEEE Intelligent Systems 17 (2002) 14–20.
9. R. Bunescu, R. Ge, R. J. Kate, E. M. Marcotte, R. J. Mooney, A. K. Ramani, Y. W. Wong, Comparative experiments on learning information extractors for proteins and their interactions, Artificial Intelligence in Medicine (special issue on Summarization and Information Extraction from Medical Documents) 33 (2) (2005) 139–155.
10. V. N. Vapnik, Statistical Learning Theory, John Wiley & Sons, New York, 1998.
11. N. Cristianini, J. Shawe-Taylor, An Introduction to Support Vector Machines and Other Kernel-based Learning Methods, Cambridge University Press, 2000.
12. National Institute of Standards and Technology, ACE – Automatic Content Extraction, http://www.nist.gov/speech/tests/ace (2000).
13. M. Steedman, The Syntactic Process, MIT Press, Cambridge, MA, 2000.
14. J. Hockenmaier, M. Steedman, Generative models for statistical parsing with combinatory categorial grammar, in: Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL-2002), Philadelphia, PA, 2002, pp. 335–342.
15. R. C. Bunescu, R. J. Mooney, A shortest path dependency kernel for relation extraction, in: Proceedings of the Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP-05), Vancouver, BC, 2005, pp. 724–731.

16. M. J. Collins, Three generative, lexicalised models for statistical parsing, in: Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL-97), 1997, pp. 16–23.
17. A. Culotta, J. Sorensen, Dependency tree kernels for relation extraction, in: Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04), Barcelona, Spain, 2004, pp. 423–429.
18. D. Zelenko, C. Aone, A. Richardella, Kernel methods for relation extraction, Journal of Machine Learning Research 3 (2003) 1083–1106.

4

Mining Diagnostic Text Reports by Learning to Annotate Knowledge Roles

Eni Mustafaraj, Martin Hoof, and Bernd Freisleben

4.1 Introduction

Several tasks approached by using text mining techniques, like text categorization, document clustering, or information retrieval, operate on the document level, making use of the so-called bag-of-words model. Other tasks, like document summarization, information extraction, or question answering, have to operate on the sentence level, in order to fulfill their specific requirements. While both groups of text mining tasks are typically affected by the problem of data sparsity, this is more accentuated for the latter group of tasks. Thus, while the tasks of the first group can be tackled by statistical and machine learning methods based on a bag-of-words approach alone, the tasks of the second group need natural language processing (NLP) at the sentence or paragraph level in order to produce more informative features.

Another issue common to all previously mentioned tasks is the availability of labeled data for training. Usually, for documents in real world text mining projects, training data do not exist or are expensive to acquire. In order to still satisfy the text mining goals while making use of a small contingent of labeled data, several approaches in machine learning have been developed and tested: different types of active learning [16], bootstrapping [13], or a combination of labeled and unlabeled data [1]. Thus, the issue of the lack of labeled data turns into the issue of selecting an appropriate machine learning approach.

The nature of the text mining task as well as the quantity and quality of available text data are other issues that need to be considered. While some text mining approaches can cope with data noise by leveraging the redundancy and the large quantity of available documents (for example, information retrieval on the Web), for other tasks (typically those restricted within a domain) the collection of documents might not possess such qualities. Therefore, more care is required for preparing such documents for the text mining task.

The previous observations suggest that performing a text mining task on new and unknown data requires handling all of the above mentioned issues, by combining and adopting different research approaches. In this chapter, we present an approach to extracting knowledge from text documents containing diagnostic problem solving situations in a technical domain (i.e., electrical engineering). In the proposed approach, we have combined techniques from several areas, including NLP, knowledge

engineering, and machine learning to implement a learning framework for annotating cases with knowledge roles. The ultimate goal of the approach is to discover interesting problem solving situations (hereafter simply referred to as cases) that can be used by an experience management system to support new engineers during their working activities. However, as an immediate benefit, the annotations facilitate the retrieval of cases on demand, allow the collection of empirical domain knowledge, and can be formalized with the help of an ontology to also permit reasoning. The experimental results presented in the chapter are based on a collection of 500 Microsoft Word documents written in German, amounting to about one million words. Several processing steps were required to achieve the goal of case annotation. In particular, we had to (a) transform the documents into an XML format, (b) extract paragraphs belonging to cases, (c) perform part-of-speech tagging, (d) perform syntactical parsing, (e) transform the results into XML representation for manual annotation, (f) construct features for the learning algorithm, and (g) implement an active learning strategy. Experimental results demonstrate the feasibility of the learning approach and a high quality of the resulting annotations.

The chapter is organized as follows. In Section 4.2 we describe our domain of interest, the related collection of documents, and how knowledge roles can be used to annotate text. In Section 4.3 we consider work in natural language processing, especially frame semantics and semantic role labeling, emphasizing parallels to our task and identifying how resources and tools from these domains can be applied to perform annotation. Section 4.4 describes in detail all the preparatory steps for the process of learning to annotate cases. Section 4.5 evaluates the results of learning. Section 4.6 concludes the chapter and outlines areas of future work.

4.2 Domain Knowledge and Knowledge Roles

4.2.1 Domain Knowledge

Our domain of interest is predictive maintenance in the field of power engineering, more specifically, the maintenance of insulation systems of high-voltage rotating electrical machines. Since in many domains it is prohibitive to allow faults that could result in a breakdown of the system, components of the system are periodically or continuously monitored to look for changes in the expected behavior, in order to undertake predictive maintenance actions when necessary. Usually, the findings related to the predictive maintenance process are documented in several forms: the measured values in a relational database; the evaluations of measurements/tests in diagnostic reports written in natural language; or the recognized symptoms in photographs. The focus of the work described here are the textual diagnostic reports.

In the domain of predictive maintenance, two parties are involved: the service provider (the company that has the know-how to perform diagnostic procedures and recommend predictive maintenance actions) and the customer (the operator of the machine). As part of their business agreement, the service provider submits to the customer an *official diagnostic report*. Such a report follows a predefined structure template and is written in syntactically correct and parsimonious language. In our case, the language is German.

A report is organized into many sections: summary, reason for the inspection, data of the inspected machine, list of performed tests and measurements, evaluations

of measurement and test results, overall assessment and recommendations, as well as several attachments with graphical plots of numerical measurements or photographs of damaged parts.

From a diagnostic point of view, the most important information is found in the evaluations of the measurements and tests performed. As a demonstration, consider the two excerpts in Figure 4.1 (originating from English documents for non-German speaking customers).

At $1.9U_N (= 30kV)$, an insulation breakdown occurred on the upper bar of the slot $N^\circ 18$, at the slot exit on the NDE side. The breakdown indicates that the bar insulation is seriously weakened. This may be caused by intense discharges due to a malfunction of the slot anti-corona protection.	The measured bypass currents are in a relatively high range indicating a certain surface conductivity. This is due to the fact that the motor was stored in cold area before it was moved to the high voltage laboratory where the temperature and humidity was much higher so that a certain degree of condensation could occur on the surface of the winding.

Fig. 4.1. Excerpts from two evaluations of isolation current measurements.

As it is often the case with diagnosis, while the quantities that are measured or the components that are inspected are the same, the findings depend on a series of contextual factors, and the reasons for these findings could be quite unique (as the examples of Figure 4.1 demonstrate). Usually, human experts need many years of field experience to gain a degree of expertise that allows them to handle any situation. The goal of our project is to mine the text documents for relevant pieces of knowledge acquired during diagnostic problem solving situations.

4.2.2 Domain Concepts

In some text mining applications, such as text categorization or information retrieval, the goal is often to discover terms specific to the domain that could be used as indices for organizing or retrieving information. Indeed, the excerpts of Figure 4.1 contain several of such domain-specific terms: *insulation*, *discharge*, *slot anti-corona protection*, *conductivity*, or *winding*. Still, using these terms as indices or keywords for representing the documents does not contribute to the purpose of our intended application, which is to find knowledge that supports diagnostic problem solving. To exemplify, consider the sentences in Figure 4.2:

1) The calculated **insulating resistance** values lay in the safe operating area.
2) Compared to the last examination, lower values for the **insulating resistance** were ascertained, due to dirtiness at the surface.

Fig. 4.2. Two sentences with the same domain concept shown in boldface.

In both sentences, the domain concept *insulating resistance* is found, but from a diagnostic point of view only the second sentence is interesting, because it describes

a possible cause for lower values. Thus, more than domain concepts are needed to capture the knowledge expressed in the documents. Our solution to this problem is to label the text with semantic annotations expressed in terms of knowledge roles, which are introduced in the following subsection.

4.2.3 Knowledge Roles

Knowledge roles are a concept introduced in CommonKADS [28], a knowledge engineering methodology for implementing knowledge-based systems. More specifically, knowledge roles are abstract names that refer to the role a domain concept plays when reasoning about a knowledge task. Such tasks are, for example, diagnosis, assessment, monitoring, or planning. Although these tasks are found in many domains, their description in CommonKADS is domain-independent. Thus, when describing a diagnosis task, knowledge roles like `finding`, `symptom`, `fault`, `parameter`, or `hypothesis` would be used.

Indeed, if we consider again the sentences in Figure 4.2, it is reasonable to represent the second sentence with knowledge roles as shown in Figure 4.3:

Knowledge Role	**Text Phrase**
Observed Object:	insulating resistance
Symptom:	lower values
Cause:	dirtiness at the surface

Fig. 4.3. Knowledge roles for sentence 2 of Figure 4.2.

Such a representation can have several advantages. Given a certain value of an *Observed Object*, a list of *Symptoms* that should be checked during the diagnosis could be retrieved. Or, given a certain *Symptom*, possible *Causes* for it could be listed, and so forth.

Understandably, we are interested in performing the text annotation with knowledge roles automatically. To achieve this goal, we draw on research in natural language understanding as described in Section 4.3.

It might be argued that one could simply use a combination of keywords to retrieve the information. For example, for sentences like that in Figure 4.2, one might write a query as below:

[low | small | high | large] && [value] && [insulating resistance]

for retrieving symptoms. Or one can search for:

[due to] | [caused by] | [as a result of] ...

to retrieve sentences containing causes. While this approach may be appealing and in some occasions even successful, there are several reasons why it could not be applied in our application:

- A large number of words (adjectives, nouns, adverbs, or verbs) can be used to describe changes (considered as symptoms in our domain), and no one can know beforehand which of them is used in the text.

- While verbs are very important for capturing the meaning of a sentence, they also abound in numbers. For example, to express an observation, any of the following verbs can be used: *observe*, *detect*, *show*, *exhibit*, *recognize*, *determine*, *result in*, *indicate*, etc. Furthermore, adverbs and negations can change their meaning and therefore need to be considered. Thus, instead of using verbs as keywords, we use them to bootstrap the annotating process, and incorporate them within semantic frames, like the frame *Observation* for the group above.
- Often, meaning emerges from the relation between different words, instead of the words separately, and this is exactly what we encountered in the diagnostic cases.

The knowledge roles used for annotating cases are abstract constructs in knowledge engineering, defined independently of any natural language constructs. Thus, a contribution of this work lies in trying to bridge the gap between knowledge roles and the natural language constructs whose meaning they capture. For this purpose, frame semantics, as described in the next section, is an ideal place to start.

4.3 Frame Semantics and Semantic Role Labeling

4.3.1 Frame Semantics

In frame semantics theory [12], a frame is a "script-like conceptual structure that describes a particular type of situation, object, or event and the participants involved in it" [24]. Based on this theory, the Berkeley FrameNet Project[1] is creating an online lexical resource for the English language by annotating text from the 100 million words British National Corpus.

The structure of a frame contains lexical units (pairs of a word with its meaning), frame elements (semantic roles played by different syntactic dependents), as well as annotated sentences for all lexical units that evoke the frame. An example of a frame with its related components is shown in Figure 4.4.

Annotation of text with frames and roles in FrameNet has been performed manually by trained linguists. An effort to handle this task automatically is being carried out by research in semantic role labeling, as described in the next subsection.

4.3.2 Semantic Role Labeling

Automatic labeling of semantic roles was introduced in [14]. In this work, after acknowledging the success of information extraction systems that try to fill in domain-specific frame-and-slot templates (see Section 4.3.4), the need for semantic frames that can capture the meaning of text independently of the domain was expressed. The authors envision that the semantic interpretation of text in terms of frames and roles would contribute to many applications, like question answering, information extraction, semantic dialogue systems, as well as statistical machine translation or automatic text summarization, and finally also to text mining.

[1] http://framenet.icsi.berkeley.edu/

After this initial work, research on semantic role labeling (SRL) has grown steadily, and in the years 2004 and 2005 [3, 4] a shared task at the CoNLL[2] was defined, in which several research institutions compared their systems. In the meantime, besides FrameNet, another corpus with manually annotated semantic roles has been prepared, PropNet [21], which differs from FrameNet in the fact that it has general semantic roles not related to semantic frames. PropNet is also the corpus used for training and evaluation of research systems on the SRL shared task. A similar corpus to FrameNet for the German language has been created by the Salsa project [10], and a discussion on the differences and similarities among these three projects is found in [9].

Frame *Evidence*

Definition: The Support, a phenomenon or fact, lends support to a claim or proposed course of action, the Proposition, where the Domain_of_Relevance may also be expressed.

Lexical units: *argue.v*, *argument.n*, *attest.v*, *confirm.v*, *contradict.v*, *corroborate.v*, *demonstrate.v*, *disprove.v*, *evidence.n*, *evidence.v*, *evince.v*, *from.prep*, *imply.v*, *indicate.v*, *mean.v*, *prove.v*, *reveal.v*, *show.v*, *substantiate.v*, *suggest.v*, *testify.v*, *verify.v*

Frame Elements:

Proposition [PRP]	This is a belief, claim, or proposed course of action to which the Support lends validity.
Support [SUP]	Support is a fact that lends epistemic support to a claim, or that provides a reason for a course of action.
...	

Examples:

And a [$_{SUP}$ sample tested] **REVEALED** [$_{PRP}$ some inflammation].
It says that [$_{SUP}$ rotation of partners] does not **DEMONSTRATE** [$_{PRP}$ independence].

Fig. 4.4. Information on the frame *Evidence* from FrameNet.

SRL is approached as a learning task. For a given target verb in a sentence, the syntactic constituents expressing semantic roles associated to this verb need to be identified and labeled with the right roles. SRL systems usually divide sentences word-by-word or phrase-by-phrase and for each of these instances calculate many features creating a feature vector. The feature vectors are then fed to supervised classifiers, such as support vector machines, maximum entropy, or memory-based learners. While adapting such classifiers to perform better on this task could bring some improvement, better results can be achieved by constructing informative features for learning. A thorough discussion of different features used for SRL can be found in [14, 22].

4.3.3 Frames and Roles for Annotating Cases

On the one hand, in knowledge engineering there are knowledge tasks and knowledge roles to represent knowledge; on the other hand, in natural language understanding there are semantic frames and semantic roles to represent meaning. When knowledge

[2] Conference of Natural Language Learning

related to a knowledge task (like diagnosis) is represented by natural language, it is reasonable to expect that some knowledge roles will map to some semantic roles. The question is how to find these mappings, and more importantly, how to label text with these roles?

A knowledge task like diagnosis or monitoring is not equivalent to a semantic frame. The former are more complex and abstract, and can usually be divided into several components, which in turn can be regarded equivalent to semantic frames. By analyzing the textual episodes of diagnostic evaluations, we noticed that they typically contain a list of observations, explanations based on evidence, and suggestions to perform some activities. Thus, we consulted FrameNet for frames like Observation, Change, Evidence, or Activity. Indeed, these frames are all present in FrameNet. For example, Activity is present in 10 subframes, and different meanings of Change are captured in 21 frames. The frame Evidence was shown in Figure 4.4, and besides the two roles of Proposition and Support, it has also roles for Degree, Depictive, Domain_of_Relevance, Manner, Means, and Result. When one carefully reads the definition of the roles Proposition and Support and looks at the examples (Figure 4.4), one can conclude that Proposition is similar to Cause and Support to Symptom in a diagnosis task.

The problem is to determine which frames to look for, given that there are currently more than six hundred frames in FrameNet. The key are the lexical units related to each frame, usually verbs. Starting with the verbs, one gets to the frames and then to the associated roles. This is also the approach we follow. We initially look for the most frequent verbs in our corpus, and by consulting several sources (since the verbs are in German), such as [15], VerbNet,[3] and FrameNet, we connect every verb with a frame, and try to map between semantic roles in a frame and knowledge roles we are interested in. One could also use the roles of FrameNet, but they are linguistically biased, and as such are not understandable by domain users that will annotate training instances for learning (a domain user would directly know to annotate *Cause*, but finds *Proposition* somehow confusing.)

In this work, FrameNet was only used as a lexical resource for consultation, that is, to find out which frames are evoked by certain lexical units, and what the related semantic roles are. Since the language of our corpus is German, we cannot make any statements about how useful the FrameNet frames could be to a learning system based on English annotated data corresponding to the defined frames.

Finally, it should be discussed why such an approach to annotating text cases with frames and roles could be beneficial to text mining. For the purpose of this discussion, consider some facts from the introduced domain corpus. During the evaluation of the learning approach, we manually annotated a subcorpus of unique sentences describing one specific measurement (high-voltage isolation current). In the 585 annotated sentences, the frame Evidence was found 152 times, 84 times evoked by the verb *zurückführen* (trace back to), 40 times by the verb *hindeuten* (point to), and 28 times by 9 other verbs. Analyzing the text annotated with the role *Cause* in the sentences with *zurückführen*, 27 different phrases expressing causes of anomalies pointed to by the symptoms were found. A few of these expressions appeared frequently, some of them occasionally, some others rarely. In Table 4.1, some of these expressions are shown.

[3] http://www.cis.upenn.edu/~bsnyder3/cgi-bin/search.cgi

Table 4.1. Some phrases annotated with the role *Cause*.

German Phrase	English Translation	Frequency
Verschmutzungseinflüsse	influences of pollution	10
leitende Verschmutzungen	conducting pollutions	8
Ionisation in Klemmenbereich	ionization in the terminal area	3
äussere Entladungen	external discharges	1

If for every sentence with the frame *Evidence* the text annotated with *Symptom* and *Cause* is extracted, this text can then be processed further with other text mining techniques for deriving domain knowledge, which is not directly available in any of the analyzed texts. For example, one could get answers to questions like: which are the most frequent symptoms and what causes can explain them; what problems (i.e., causes) do appear frequently in a specific type of machine, etc. Thus, such an annotation with frames and roles preprocesses text by generating very informative data for text mining, and it can also be used in the original form for information retrieval. Still, such an approach makes sense in those cases when text contains descriptions of repetitive tasks, which are then expressed by a small number of underlying semantic frames. Since data and text mining try to extract knowledge from data of the same nature in the same domain, we find that annotation of text with knowledge roles could be a valuable approach.

Before explaining in detail the process of learning to automatically annotate text with knowledge roles (based on the SRL task) in Section 4.4, we briefly discuss the related field of information extraction.

4.3.4 Information Extraction

Information extraction (IE), often regarded as a restricted form of natural language understanding, predates research in text mining, although today, IE is seen as one of the techniques contributing to text mining [30]. Actually, the purpose of IE is very similar to what we are trying to achieve with role annotation. In IE it is usually known in advance what information is needed, and part of text is extracted to fill in slots of a predefined template. An example, found in [20], is the job posting template, where, from job posting announcements in Usenet, text to fill slots like: title, state, city, language, platform, etc. is extracted and stored in a database for simpler querying and retrieval.

Usually, methods used by IE have been based on shallow NLP techniques, trying to extract from a corpus different types of syntactic rules that match syntactic roles to semantic categories, as for example in [23].

With the advances in NLP and machine learning research, IE methods have also become more sophisticated. Actually, SRL can also be seen as a technology for performing information extraction, in those cases when text is syntactically and semantically more demanding and expressive. All these technologies are intended to be used for extracting knowledge from text, despite their differences in implementation or scope.

4.4 Learning to Annotate Cases with Knowledge Roles

To perform the task of learning to annotate cases with knowledge roles, we implemented a software framework, as shown in Figure 4.5. Only the preparation of documents (described in Section 4.4.1) is performed outside of this framework. In the remainder of the section, every component of the framework is presented in detail.

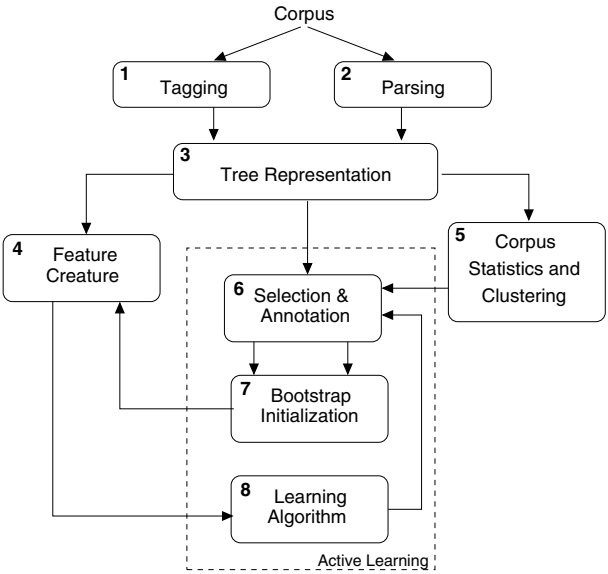


Fig. 4.5. The Learning Framework Architecture.

4.4.1 Document Preparation

In Section 4.2.1 it was mentioned that our documents are official diagnostic reports hierarchically structured in several sections and subsections, written by using MS® Word. Actually, extracting text from such documents, while preserving the content structure, is a difficult task. In completing it we were fortunate twice. First, with MS® Office 2003 the XML based format WordML was introduced that permits storing MS® Word documents directly in XML. Second, the documents were originally created using a MS® Word document template, so that the majority of them had the same structure. Still, many problems needed to be handled. MS® Word mixes formatting instructions with content very heavily and this is reflected also in its XML format. In addition, information about spelling, versioning, hidden template elements, and so on are also stored. Thus, one needs to explore the XML output of the documents to find out how to distinguish text and content structure from unimportant information. Such a process will always be a heuristic one, depending on the nature of the documents. We wrote a program that reads the XML document tree,

and for each section with a specified label (from the document template) it extracts the pure text and stores it in a new XML document, as the excerpt in Figure 4.6 shows.

```
<section title="Measurements">
  <subsection title="Stator_Winding">
    <measurement title="Visual_Control">
      <submeasurement title="Overhang_Support">
        <evaluation>
          Die Wickelkopfabsttzung AS und NS befand sich in einem ...
        </evaluation>
        <action>Keine</action>
      </submeasurement>
...
```

Fig. 4.6. Excerpt of the XML representation of the documents.

Based on such an XML representation, we create subcorpora of text containing measurement evaluations of the same type, stored as paragraphs of one to many sentences.

4.4.2 Tagging

The part-of-speech (POS) tagger (TreeTagger[4]) that we used [26] is a probabilistic tagger with parameter files for tagging several languages: German, English, French, or Italian. For some small problems we encountered, the author of the tool was very cooperative in providing fixes. Nevertheless, our primary interest in using the tagger was not the POS tagging itself (the parser, as is it shown in Section 4.4.3, performs tagging and parsing), but getting stem information (since the German language has a very rich morphology) and dividing the paragraphs in sentences (since the sentence is the unit of operation for the next processing steps).

The tag set used for tagging German is slightly different from that of English.[5] Figure 4.7 shows the output of the tagger for a short sentence.[6]

As indicated in Figure 4.7, to create sentences it suffices to find the lines containing: `". \$. ."` (one sentence contains all the words between two such lines). In general, this is a very good heuristic, but its accuracy depends on the nature of the text. For example, while the tagger correctly tagged abbreviations found in its list of abbreviations (and the list of abbreviations can be customized by adding abbreviations common to the domain of the text), it got confused when the same abbreviations were found inside parentheses, as the examples in Figure 4.8 for the word 'ca.' (circa) show.

If such phenomena occur often, they become a problem for the further correct processing of sentences, although one becomes aware of such problems only in the

[4] http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger
[5] http://www.ims.uni-stuttgart.de/projekte/corplex/TagSets/stts-table.html
[6] Translation: A generally good external winding condition is present.

Es	PPER	es
liegt	VVFIN	liegen
insgesamt	ADV	insgesamt
ein	ART	ein
guter	ADJA	gut
äusserer	ADJA	äuβer
Wicklungszustand	NN	<unknown>
vor	PTKVZ	vor
.	$.	.

Fig. 4.7. A German sentence tagged with POS-tags by TreeTagger.

course of the work. A possible solution in such cases is to use heuristics to replace erroneous tags with correct ones for the types of identified errors.

an	APR	an
ca.	ADV	ca.
50	CARD	50
%	NN	%

(	$(	(
ca	NE	<unknown>
.	$.	.
20	CARD	20

Fig. 4.8. Correct and erroneous tagging for the word 'ca.'

The more problematic issue is that of words marked with the stem <unknown>. Actually, their POS is usually correctly induced, but we are specifically interested in the stem information. The two reasons for an <unknown> label are a) the word has been misspelled and b) the word is domain specific, and as such not seen during the training of the tagger. On the positive side, selecting the words with the <unknown> label directly creates the list of domain specific words, useful in creating a domain lexicon.

A handy solution for correcting spelling errors is to use a string similarity function, available in many programming language libraries. For example, the Python language has the function "get_close_matches" in its "difflib" library. An advantage of such a function is having as a parameter the degree of similarity between strings. By setting this value very high (between 0 and 1) one is sure to get really similar matches if any at all.

Before trying to solve the problem of providing stems for words with the <unknown> label, one should determine whether the stemming information substantially contributes to the further processing of text. Since we could not know that in advance, we manually provided stems for all words labeled as <unknown>. Then, during the learning process we performed a set of experiments, where: a) no stem information at all was used and b) all words had stem information (tagger + manually created list of stems). Table 4.2 summarizes the recall and precision of the learning task in each experiment.

These results show approximately 1% improvement in recall and precision when stems instead of original words are used. We can say that at least for the learning task of annotating text with knowledge roles stem information is not necessarily important, but this could also be due to the fact that a large number of other features (see Section 4.4.5) besides words are used for learning.

Table 4.2. Results of experiments for the contribution of stem information on learning.

Experiment	Recall	Precision
a) no stems (only words)	90.38	92.32
b) only stems	91.29	93.40

Still, the reason for having a list of stems was not in avoiding more data due to word inflections, but in capturing the word composition, a phenomenon typical for the German language. For example, all the words in the first row of Table 4.3 are compound words that belong to the same semantic category identified by their last word 'wert' (value), i.e., they all denote values of different measured quantities, and as such have a similar meaning. This similarity cannot be induced if one compares the words in the original form, something possible by comparing the word representations of the second row.

Table 4.3. Original words (first row), words composed of stems (second row).

Ableitstrom*werte*, Gesamtstrom*werte*, Isolationswiderstands*werte*, Isolationsstrom*werte*, Kapazitäts*werte*, Ladestrom*werte*, Strom*wert*en, Verlustfaktoranfangs*wert*, etc.
Ableit-Strom-Wert, Gesamt-Strom-Wert, Isolation-Widerstand-Wert, Isolation-Strom-Wert, Kapazität-Wert, Lade-Strom-Wert, Strom-Wert, Verlustfaktor-Anfang-Wert, etc.

Unfortunately, there are only a few tools available for morphological analysis of German words. We tried Morphy [17], which is publicly available, but it was not able to analyze any of our domain-specific words. Therefore, we had to perform this task by hand.

4.4.3 Parsing

Syntactical parsing is one of the most important steps in the learning framework, since the produced parse trees serve as input for the creation of features used for learning. Since we are interested in getting qualitative parsing results, we experimented with three different parsers: the Stanford parser (Klein 2005), the BitPar parser [27, 25], and the Sleepy parser [7]. What these parsers have in common is that they all are based on unlexicalized probabilistic context free grammars (PCFG) [18], trained on the same corpus of German, Negra[7] (or its superset Tiger[8]), and their source code is publicly available. Still, they do differ in the degree they model some structural aspects of the German language, their annotation schemas, and the infor-

[7] http://www.coli.uni-saarland.de/projects/sfb378/negra-corpus/
[8] http://www.ims.uni-stuttgart.de/projekte/TIGER/TIGERCorpus/

mation included in the output. Figure 4.9 shows the output of the same sentence[9] parsed by each parser, and in the following, we discuss each of them.

Stanford Parser

```
(ROOT
 (NUR
  (S
   (PP (APPR Auf) (CARD NS))
   (VAFIN wurden)
   (VP
    (AP (ADV ca.)
     (NM (CARD 5)
      (CARD gerissene)
      (CARD Keilsicherungsbandagen)))
    (VVPP festgestellt)))
  ($..)))
```

BitPar Parser

```
(utt:
 (S.fin:
  (PP: (APPR: Auf)
   (NN: NS))
  (VWFIN: wurden)
  (AP: (AVP-MAD: (ADV-MAD: ca.))
   (CARD: 5))
  (NP.nom: (AP: (ADJA%: gerissene))
   (NN.nom: Keilsicherungsbandagen))
  (VVPP%: festgestellt)))
 (\$.: .))
```

Sleepy Parser

```
(TOP
 (S
  (PP-MO (APPR-AD Auf)
   (NE-NK NS) )
  (VAFIN-HD wurden)
  (NP-SB
   (ADV-MO ca.) (CARD-NK 5)
   (ADJA-NK gerissene)
   (NN-NK  Keilsicherungsbandagen))
  (VP-OC (VVPP-HD festgestellt)))
 ($. .))
```

Auf	NS wurden	ca. 5 gerissene	Keilsicherungsbandagen festgestellt.
On	NS were	ca. 5 torn	wedge's safety bands detected.

Fig. 4.9. Parsing output of the same sentence from the three parsers

Stanford Parser - The Stanford parser is an ambitious project that tackles the task of generating parse trees from unlabeled data independently of the language. For the moment, the parser is distributed with parameter files for parsing English, German, and Chinese. We tested the parser on our data and noticed that

[9] English translation: "On NS were detected circa 5 torn wedge's safety bands."

the POS tags were often erroneously induced (in the sentence with only 8 words of Figure 4.9 there are 3 such errors—CARD tags for 2 nouns and 1 adjective), which then resulted in erroneous parse trees. But, in those cases when the tagging was performed correctly, the parse trees were also correct. Still, the parser could not parse long sentences, perhaps due to the fact that it was trained in the part of the Negra corpus with sentences having up to 10 words. Trying the parser with long English sentences instead, produced excellent results. We concluded that at this phase of implementation, the Stanford parser could not be used with our corpus of German sentences that contain an average of up to 18 words per sentence.

BitPar Parser - This parser is composed of two parts, the parser itself [27] and the parameter files (chart rules, lexicon, etc.) from [25]. Published experimental results claim robust performance, due to the use of sophisticated annotation and transformation schemata for modeling grammars. Another advantage of the parser is that its lexicon can be extended very easily with triples of domain-dependent words, their tags, their frequency counts in a corpus, thus avoiding the tagging errors typical for unlexicalised parsers. These tagging errors damage the parse results, as can be seen from the results of the Stanford parser. Our critique for the described BitPar is that it usually produces trees with more nodes than the other parsers and the annotation of nodes contains specialized linguistic information, not very appropriate for creating features for learning.

Sleepy Parser - This parser has been specifically tuned for the German language, and while it is a statistical parser like the others, it uses different annotation schemas and incorporates grammatical functions (SB–subject, OC–clausal object, MO–modifier, HD–head, etc.) or long-distance dependencies between terms. In constrast to the two other parsers, it also has a highly tuned suffix analyzer for guessing POS tags [8], which contributes to more accurate tagging results than the other parsers, although some domain-dependent words are not always correctly tagged. Erroneous parsing is also encountered for very long sentences.

Choosing a Parser

All the tested parsers make errors during parsing. In the end, the criteria upon which we based our choice of the parser were speed and output information. Sleepy was the fastest and had the most informative output (it prints the log value expressing the likelihood of parsing, and it labels the majority of nodes with their grammatical function). Actually, choosing a parser upon these criteria instead of the accuracy of parsing could be regarded as inappropriate. Our justification is that a metric to measure the accuracy of parsing on new data does not exist. These parsers have all been trained on the same corpus, and at least the two German parsers tuned up to the point where their results are almost the same. Thus, *a priori* their expected accuracy in a new corpus should be equal, and accuracy is not a criterion for choosing one over the other. Given the difficulty of evaluating the accuracy of the parse trees and their presumed similarity, we based the choice of parser on the qualities that contributed most to our task, namely speed and informative output.

4.4.4 Tree Representation

The bracketed parse tree and the stem information of tagging serve as input for the step of creating a tree data structure. The tree is composed of terminals (leaf nodes) and non-terminals (internal nodes), all of them known as constituents of the tree. For export purposes as well as for performing exploration or annotation of the corpus, the tree data structures are stored in XML format, according to a schema defined in the TigerSearch[10] tool. The created tree, when visualized in TigerSearch, looks like the one shown in Figure. 4.10.[11] The terminals are labeled with their POS tags and also contain the corresponding words and stems; the inside nodes are labeled with their phrase types (NP, PP, etc.); and the branches have labels, too, corresponding to the grammatical functions of the nodes. The XML representation of a portion of the tree is shown in Figure 4.11.

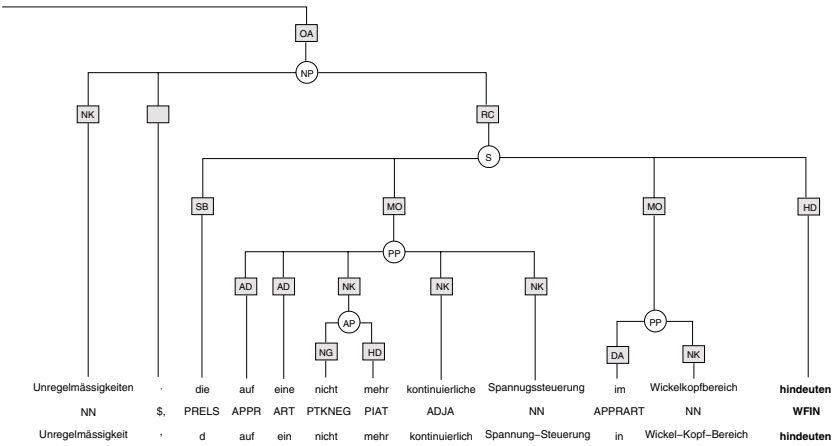


Fig. 4.10. Representation of a parsed tree in the TigerSearch tool. Due to space reasons, only a branch of the tree is shown.

4.4.5 Feature Creation

Features are created from the parse tree of a sentence. A feature vector is created for every constituent of the tree, containing some features unique to the constituent, some features common to all constituents of the sentence, and some others calculated with respect to the target constituent (the predicate verb).

A detailed linguistic description of possible features used by different research systems for the SRL task is found in [22]. In this subsection, we only list the features used in our system and give example values for the leaf node `Spannungssteuerung` of the parse tree in Figure 4.10.

[10] http://www.ims.uni-stuttgart.de/projekte/TIGER/TIGERSearch/

[11] English translation: "... irregularities, which point to a not anymore continuous steering of voltage in the area of the winding head."

```
...
  <t lemma="Spannung-Steuerung" word="Spannungssteuerung" pos="NN"
       id="sentences._108_28" />
  <t lemma="in" word="im" pos="APPRART"
       id="sentences._108_29" />
  <t lemma="Wickel-Kopf-Bereich" word="Wickelkopfbereich" pos="NN"
       id="sentences._108_30" />
  <t lemma="hindeuten" word="hindeuten" pos="VVFIN" id="sentences._108_31" />
 </terminals>
 <nonterminals>
  <nt id="sentences._108_500" cat="PP">
   <edge idref="sentences._108_3" label="NK" />
   <edge idref="sentences._108_2" label="DA" />
   <edge idref="sentences._108_1" label="DA" />
  </nt>
 ...
```

Fig. 4.11. XML representation of a portion of the parse tree from Figure 4.10.

Phrase type **NN**
Grammatical function **NK**
Terminal (is the constituent a terminal or non-terminal node?) **1**
Path (path from the target verb to the constituent, denoting u(up) and d(down) for the direction) **uSdPPd**
Grammatical path (like Path, but instead of node labels, branch labels are considered) **uHDdMOdNK**
Path length (number of branches from target to constituent) **3**
Partial path (path to the lowest common ancestor between target and constituent) **uPPuS**
Relative Position (position of the constituent relative to the target) **left**
Parent phrase type (phrase type of the parent node of the constituent) **PP**
Target (lemma of the target word) **hindeuten**
Target POS (part-of-speech of the target) **VVFIN**
Passive (is the target verb passive or active?) **0**
Preposition (the preposition if the constituent is a PP) **none**
Head Word (for rules on head words refer to [5]) **Spannung-Steuerung**
Left sibling phrase type **ADJA**
Left sibling lemma **kontinuierlich**
Right sibling phrase type **none**
Right sibling lemma **none**
Firstword, Firstword POS, Lastword, Lastword POS (in this case, the constituent has only one word, thus, these features get the same values: Spannung-Steuerung and NN. For non-terminal constituents like PP or NP, first word and last word will be different.)
Frame (the frame evoked by the target verb) **Evidence**
Role (this is the class label that the classifier will learn to predict. It will be one of the roles related to the frame or none, for an example refer to Figure 4.12.) **none**

If a sentence has several clauses where each verb evokes a frame, the feature vectors are calculated for each evoked frame separately and all the vectors participate in the learning.

4.4.6 Annotation

To perform the manual annotation, we used the Salsa annotation tool (publicly available) [11]. The Salsa annotation tool reads the XML representation of a parse tree and displays it as shown in Figure 4.12. The user has the opportunity to add frames and roles as well as to attach them to a desired target verb. In the example of Figure 4.12 (the same sentence of Figure 4.10), the target verb *hindeuten* (point to) evokes the frame Evidence, and three of its roles have been assigned to constituents of the tree. Such an assignment can be easily performed using point-and-click. After this

process, an element <frames> is added to the XML representation of the sentence, containing information about the frame. Excerpts of the XML code are shown in Figure 4.13.

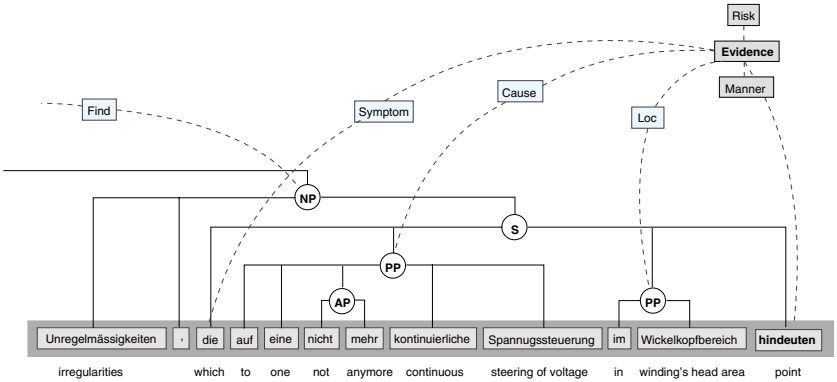


Fig. 4.12. Annotation with roles with the Salsa tool.

```
<frames>
 <frame name="Evidence" id="sentences._108__f1">
  <target><fenode idref="sentences._108_31"/></target>
   <fe name="Symptom" id="sentences._108_f1_e1">
    <fenode idref="sentences._108_22"/>
   </fe>
   <fe name="Cause" id="sentences._108__f1_e2">
    <fenode idref="sentences._108_509"/>
   </fe>
    <fe name="Loc" id="sentences._108__f1_e5">
    <fenode idref="sentences._108_510"/>
   </fe>
...
```

Fig. 4.13. XML Representation of an annotated frame.

4.4.7 Active Learning

Research in IE has indicated that using an active learning approach for acquiring labels from a human annotator has advantages over other approaches of selecting instances for labeling [16]. In our learning framework, we have also implemented an active learning approach. The possibilities for designing an active learning strategy are manifold; the one we have implemented uses a committee-based classification scheme that is steered by corpus statistics. The strategy consists of the following steps:

a) Divide the corpus in clusters of sentences with the same target verb. If a cluster has fewer sentences than a given threshold, group sentences with verbs evoking the same frame into the same cluster.
b) Within each cluster, group the sentences (or clauses) with the same parse sub-tree together.
c) Select sentences from the largest groups of the largest clusters and present them to the user for annotation.
d) Bootstrap initialization: apply the labels assigned by the user to groups of sentences with the same parse sub-tree.
e) Train all the classifiers of the committee on the labeled instances; apply each trained classifier to the unlabeled sentences.
f) Get a pool of instances where the classifiers of the committee disagree and present to the user the instances belonging to sentences from the next largest clusters not yet manually labeled.
g) Repeat steps d)–f) a few times until a desired accuracy of classification is achieved.

In the following, the rationale behind choosing these steps is explained.

Steps a), b), c): In these steps, statistics about the syntactical structure of the corpus are created, with the intention of capturing its underlying distribution, so that representative instances for labeling can be selected.

Step d): This step has been regarded as applicable to our corpus, due to the nature of the text. Our corpus contains repetitive descriptions of the same diagnostic measurements on electrical machines, and often, even the language used has a repetitive nature. Actually, this does not mean that the same words are repeated (although often standard formulations are used, especially in those cases when nothing of value was observed). Rather, the kind of sentences used to describe the task has the same syntactic structure. As an example, consider the sentences shown in Figure 4.14.

[PP Im Nutaustrittsbereich] wurden [NP stärkere Glimmentladungsspuren] festgestellt.
In the area of slot exit stronger signs of corona discharges were detected.

[PP Bei den Endkeilen] wurde [NP ein ausreichender Verkeildruck] festgestellt.
At the terminals' end a sufficient wedging pressure was detected.

[PP An der Schleifringbolzenisolation] wurden [NP mechanische Beschädigungen] festgestellt.
On the insulation of slip rings mechanical damages were detected.

[PP Im Wickelkopfbereich] wurden [NP grossflächige Decklackablätterungen] festgestellt.
In the winding head area extensive chippings of the top coating were detected.

Fig. 4.14. Examples of sentences with the same structure.

What all these sentences have in common is the passive form of the verb *feststellen* (wurden festgestellt), and due to the subcategorization of this verb, the parse tree on the level of phrases is identical for all sentences, as indicated by 4.15. Furthermore, for the frame Observation evoked by the verb, the assigned roles are in all cases: NP—Finding, PP—Observed_Object. Thus, to bootstrap initialization, we assign the same roles to sentences with the same sub-tree as the manually labeled sentences.

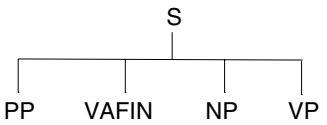


Fig. 4.15. Parse tree of the sentences in Figure 4.14.

Step e): The committee of classifiers consists of a maximum entropy (MaxEnt) classifier from Mallet [19], a Winnow classifier from SNoW [2], and a memory-based learner (MBL) from TiMBL [6]. For the MBL, we selected k=5 as the number of the nearest neighbours. The classification is performed as follows: if at least two classifiers agree on a label, the label is accepted. If there is disagreement, the cluster of labels from the five nearest neighbours is examined. If the cluster is not homogenous (i.e., it contains different labels), the instance is included in the set of instances to be presented to the user for manual labeling.

Step f): If one selects new sentences for manual annotation only based on the output of the committee-based classifier, the risk of selecting outlier sentences is high [29]. Thus, from the instances' set created by the classifier, we select those belonging to large clusters not manually labeled yet.

4.5 Evaluations

To evaluate this active learning approach on the task of annotating text with knowledge roles, we performed a series of experiments that are described in the following. It was explained in Section 4.4.1 that, based on the XML structure of the documents, we created subcorpora with text belonging to different types of diagnostic tests. After such subcorpora have been processed to create sentences, only unique sentences are retained for further processing (repetitive, standard sentences do not bring any new information, they only disturb the learning and therefore are discarded). Then, lists of verbs were created, and by consulting the sources mentioned in Section 4.3.3, verbs were grouped with one of the frames: Observation, Evidence, Activity, and Change. Other verbs that did not belong to any of these frames were not considered for role labeling.

4.5.1 Learning Performance on the Benchmark Datasets

With the aim of exploring the corpus to identify roles for the frames and by using our learning framework, we annotated two different subcorpora and then manually controlled them, to create benchmark datasets for evaluation. Some statistics for the manually annotated subcorpora are summarized in Table 4.4. Then, to evaluate the efficiency of the classification, we performed 10-fold cross-validations on each set, obtaining the results shown in Table 4.5, where recall, precision, and the $F_{\beta=1}$ measure are the standard metrics of information retrieval.

We analyzed some of the classification errors and found that they were due to parsing anomalies, which had forced us in several occasions to split a role among several constituents.

Table 4.4. Statistics for the benchmark datasets.

Subcorpus	Cases No.	Sentences No.	Unique Sentences No.	Annotated Roles No.
Isolation Current	491	1134	585	1862
Wedging System	453	775	602	1751

Table 4.5. Learning results for the benchmark datasets.

Subcorpus	Recall	Precision	$F_{\beta=1}$ measure
Isolation Current	0.913	0.934	0.92
Wedging System	0.838	0.882	0.86

4.5.2 Active Learning versus Uniform Random Selection

In order to evaluate the advantages of active learning, we compared it to the uniform random selection of sentences for manual annotations. Some results for both approaches are summarized in Table 4.6 and Table 4.7. Recall, precision, and $F_{\beta=1}$ measure were calculated after each iteration, in which 10 new sentences manually labeled were added to the training set. The results of active learning ($F_{\beta=1}$ measure) are 5–10 points better than those of random learning. For this experiment, the step d) of the active learning strategy was not applied, since it is very specific to our corpus.

Table 4.6. Random Learning Results.

Sentences No.	Recall	Precision	$F_{\beta=1}$ measure
10	0.508	0.678	0.581
20	0.601	0.801	0.687
30	0.708	0.832	0.765
40	0.749	0.832	0.788

Table 4.7. Active Learning Results.

Sentences No.	Recall	Precision	$F_{\beta=1}$ measure
10	0.616	0.802	0.697
20	0.717	0.896	0.797
30	0.743	0.907	0.817
40	0.803	0.906	0.851

4.5.3 Bootstrapping Based on Other Sets

During the annotation of the two benchmark datasets, we noticed that the two subcorpora, although different in nature (set_1: *Isolation Current* contains evaluations of numerical measurements performed on the three phases of the machine, set_2: *Wedging System* describes visual inspections on the wedging components of the machine) had very often the frame Observation or Change in common, while the frame Evidence appeared almost only in the first set, and the frame Activity almost always in the second. Thus, we tested whether text annotated with the same roles in one set could bootstrap the learning in the second, and the results are summarized in Table 4.8.

Table 4.8. Results for bootstrapping based on other labeled sets

Training File	Testing File	Recall	Precision
Isolation Current	Wedging System	0.765	0.859
Wedging System	Isolation Current	0.642	0.737

We consider these results as very promising, since they hint at the possibility of using previously annotated text from other subcorpora to bootstrap the learning process, something that would alleviate the process of acquiring manual annotations for new text.

4.6 Conclusions

In this chapter, we have presented an approach for extracting knowledge from text documents containing descriptions of knowledge tasks in a technical domain. Knowledge extraction in our approach is based on the annotation of text with knowledge roles (a concept originating in knowledge engineering), which we map to semantic roles found in frame semantics. The framework implemented for this purpose is based on deep NLP and active learning. Experiments have demonstrated a robust learning performance, and the obtained annotations were of high quality. Since our framework is inspired by and founded upon research in semantic role labeling (SRL), the results indicate that SRL could become a highly valuable processing step for text mining tasks.

In future work, we will consider the advantages of representing annotated text by means of knowledge roles and the related frames. Besides the previously explained uses for semantic retrieval of cases and the extraction of empirical domain knowledge facts, such a representation could also permit looking for potentially interesting relations in text and can be exploited to populate application and domain ontologies with lexical items.

4.7 Acknowledgments

The Insulation Competence Center of ALSTOM Ltd. Switzerland kindly permitted the use of the text documents for research purposes. Katrin Erk, Sebastian Pado, Amit Dubey, Sabine Schulte im Walde, Michael Schiehlen, and Helmut Schmid provided their linguistic tools and were an invaluable source of information and support. We are grateful to all of them.

References

1. A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *Proc. of the Workshop on Computational Learning Theory, COLT '98, Madison, WI*, pages 92–100, 1998.
2. A. J. Carlson, C. M. Cumby, N. D. Rizzolo, J. L. Rosen, and D. Roth. SNoW: Sparse Network of Winnow. 2004.
3. X. Carreras and L. Màrquez. Introduction to the coNLL shared task: Semantic role labeling. In *Proc. of 8th Conference of Natural Language Learning*, pages 89–97, Boston, MA, 2004.
4. X. Carreras and L. Màrquez. Introduction to the coNLL-2005 shared task: Semantic role labeling. In *Proc. of 9th Conference of Natural Language Learning*, pages 152–165, Ann Arbor, MI, June 2005.
5. M. Collins. *Head-Driven Statistical Models for Natural Language Parsing.* PhD thesis, University of Pennsylvania, 1999.
6. W. Daelemans, J. Zavrel, K. van der Sloot, and A. van den Bosch. TiMBL: Tilburg Memory Based Learner. 2004.
7. A. Dubey. *Statistical Parsing for German.* PhD thesis, University of Saarland, Germany, 2003.
8. A. Dubey. What to do when lexicalization fails: Parsing German with suffix analysis and smoothing. In *Proc. of 43rd Annual Meeting of ACL, Ann Arbor, MI*, pages 314–321, 2005.
9. M. Ellsworth, K. Erk, P. Kingsbury, and S. Padó. PropBank, SALSA, and FrameNet: How design determines product. In *Proc. of the LREC 2004 Workshop on Building Lexical Resources from Semantically Annotated Corpora, Lisbon, Portugal*, 2004.
10. K. Erk, A. Kowalski, and S. Padó. The Salsa annotation tool-demo description. In *Proc. of the 6th Lorraine-Saarland Workshop, Nancy, France*, pages 111–113, 2003.
11. K. Erk, A. Kowalski, S. Padó, and M. Pinkal. Towards a resource for lexical semantics: A large German corpus with extensive semantic annotation. In *Proc. of 41st Annual Meeting of ACL, Saporo, Japan*, pages 537–544, 2003.
12. C. J. Fillmore. Frame semantics and the nature of language. In *Annals of the New York Academy of Sciences: Conf. on the Origin and Development of Language and Speech*, volume 280, pages 20–32, 1976.
13. R. Ghani and R. Jones. A comparison of efficacy of bootstrapping of algorithms for information extraction. In *Proc. of LREC 2002 Workshop on Linguistic Knowledge Acquisition, Las Palmas, Spain*, 2002.
14. D. Gildea and D. Jurafsky. Automatic labeling of semantic roles. In *Computational Linguistics*, volume 23, pages 245–288, 2002.

15. S. Schulte im Walde. *Experiments on the Automatic Induction of German Semantic Verb Classes*. PhD thesis, Universität Stuttgart, Germany, 2003.
16. R. Jones, R. Ghani, T. Mitchell, and E. Riloff. Active learning for information extraction with multiple view features sets. In *Proc. of Adaptive Text Extraction and Mining, EMCL/PKDD-03, Cavtat-Dubrovnik, Croatia*, pages 26–34, 2003.
17. W. Lezius. Morphy - German morphology, part-of-speech tagging and applications. In *Proc. of 9th Euralex International Congress, Stuttgart, Germany*, pages 619–623, 2000.
18. C. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, Cambridge, MA, 1999.
19. A. K. McCallum. MALLET: A machine learning for language toolkit, 2002.
20. R. J. Mooney and R. Bunescu. Mining knowledge from text using information extraction. *SIGKDD Explor. Newsl.*, 7(1):3–10, 2005.
21. M. Palmer and D. Gildea. The proposition bank: An annotated corpus of semantic roles. In *Computational Linguistics*, volume 31, pages 71–106, 2005.
22. S. Pradhan, K. Hacioglu, V. Kruglery, W. Ward, J. H. Martin, and D. Jurafsky. Support vector learning for semantic argument classification. *Machine Learning Journal*, Kluwer Academic Publishers, 59:1–29, 2005.
23. E. Rillof and M. Schelzenbach. An empirical approach to conceptual frame acquisition. In *Proc. of 6th Workshop on Very Large Corpora, Montreal, Canada*, pages 49–56, 1998.
24. J. Ruppenhofer, M. Ellsworth, M. R. L. Petruck, and C. R. Johnson. *FrameNet: Theory and Practice*. 2005.
25. M. Schiehlen. Annotation strategies for probabilistic parsing in German. In *Proc. of CoLing'04, Geneva, Switzerland*, 2004.
26. H. Schmid. Improvement in part-of-speech tagging with an application to German. In *Proc. of the ACL SIGDAT-Workshop, Dublin, Ireland*, pages 47–50, 1995.
27. H. Schmid. Efficient parsing of highly ambiguous context-free grammars with bit vectors. In *Proc. of CoLing'04, Geneva, Switzerland*, 2004.
28. G. Schreiber, H. Akkermans, A. Anjewierden, R. deHoog, N. Shadbolt, W. VandeVelde, and B. Wielinga. *Knowledge Engineering and Management: The CommonKADS Methodology*. The MIT Press, Cambridge, MA, 2000.
29. M. Tang, X. Luo, and S. Roukos. Active learning for statistical natural language parsing. In *Proc. of the ACL 40th Anniversary Meeting, Philadelphia, PA*, pages 120–127, 2002.
30. S. Weiss, N. Indurkhya, T. Zhang, and F. Damerau. *Text Mining: Predictive Methods for Analyzing Unstructured Information*. Springer, New York, NY, 2004.

5

A Case Study in Natural Language Based Web Search

Giovanni Marchisio, Navdeep Dhillon, Jisheng Liang, Carsten Tusk, Krzysztof Koperski, Thien Nguyen, Dan White, and Lubos Pochman

5.1 Introduction

Is there a public for natural language based search? This study, based on our experience with a Web portal, attempts to address criticisms on the lack of scalability and usability of natural language approaches to search. Our solution is based on InFact®, a natural language search engine that combines the speed of keyword search with the power of natural language processing. InFact performs clause level indexing, and offers a full spectrum of functionality that ranges from Boolean keyword operators to linguistic pattern matching in real time, which include recognition of syntactic roles, such as subject/object and semantic categories, such as people and places. A user of our search can navigate and retrieve information based on an understanding of actions, roles and relationships. In developing InFact, we ported the functionality of a deep text analysis platform to a modern search engine architecture. Our distributed indexing and search services are designed to scale to large document collections and large numbers of users. We tested the operational viability of InFact as a search platform by powering a live search on the Web. Site statistics and user logs demonstrate that a statistically significant segment of the user population is relying on natural language search functionality. Going forward, we will focus on promoting this functionality to an even greater percentage of users through a series of creative interfaces.

Information retrieval on the Web today makes little use of Natural Language Processing (NLP) techniques [1, 3, 11, 15, 18]. The perceived value of improved understanding is greatly outweighed by the practical difficulty of storing complex linguistic annotations in a scalable indexing and search framework. In addition, any champion of natural language techniques must overcome significant hurdles in user interface design, as greater search power often comes at a price of more work in formulating a query and navigating the results. All of these obstacles are compounded by the expected resistance to any technological innovation that has the potential to change or erode established models for advertising and search optimization, which are based on pricing of individual keywords or noun phrases, rather than relationships or more complex linguistic constructs.

Nevertheless, with the increasing amount of high value content made available on the Web and increased user sophistication, we have reasons to believe that a segment

of the user population will eventually welcome tools that understand a lot more than present day keyword search does. Better understanding and increased search power depend on better parameterization of text content in a search engine index. The most universal storage employed today to capture text content is an inverted index. In a typical Web search engine, an inverted index may register presence or frequency or keywords, along with font size or style, and relative location in a Web page. Obviously this model is only a rough approximation to the complexity of human language and has the potential to be superseded by future generation of indexing standards.

InFact relies on a new approach to text parameterization that captures many linguistic attributes ignored by standard inverted indices. Examples are syntactic categories (parts of speech), syntactical roles (such as subject, objects, verbs, prepositional constraints, modifiers, etc.) and semantic categories (such as people, places, monetary amounts, etc.). Correspondingly, at query time, there are explicit or implicit search operators that can match, join or filter results based on this rich assortment of tags to satisfy very precise search requirements.

The goal of our experiment was to demonstrate that, once scalability barriers are overcome, a statistically significant percentage of Web users can be converted from keyword search to natural language based search. InFact has been the search behind the GlobalSecurity.org site (www.globalsecurity.org) for the past six months. According to the Alexa site (www.alexa.com), GlobalSecurity.org has a respectable overall traffic rank (no. 6,751 as of Feb 14, 2006). Users of the site can perform keyword searches, navigate results by action themes, or enter explicit semantic queries. An analysis of query logs demonstrate that all these non-standard information discovery processes based on NLP have become increasingly popular over the first six months of operation.

The remainder of this chapter is organized as follows. Section 5.2 presents an overview of our system, with special emphasis on the linguistic analyses and new search logic. Section 5.3 describes the architecture and deployment of a typical InFact system. Section 5.4 is a study of user patterns and site statistics.

5.2 InFact System Overview

InFact consists of an indexing and a search module. With reference to Figure 5.1, indexing pertains to the processing flow on the bottom of the diagram. InFact models text as a complex multivariate object using a unique combination of deep parsing, linguistic normalization and efficient storage. The storage schema addresses the fundamental difficulty of reducing information contained in parse trees into generalized data structures that can be queried dynamically. In addition, InFact handles the problem of linguistic variation by mapping complex linguistic structures into semantic and syntactic equivalents. This representation supports dynamic relationship and event search, information extraction and pattern matching from large document collections in real time.

5.2.1 Indexing

With reference to Figure 5.1, InFact's Indexing Service performs in order: 1) document processing, 2) clause processing, and 3) linguistic normalization.

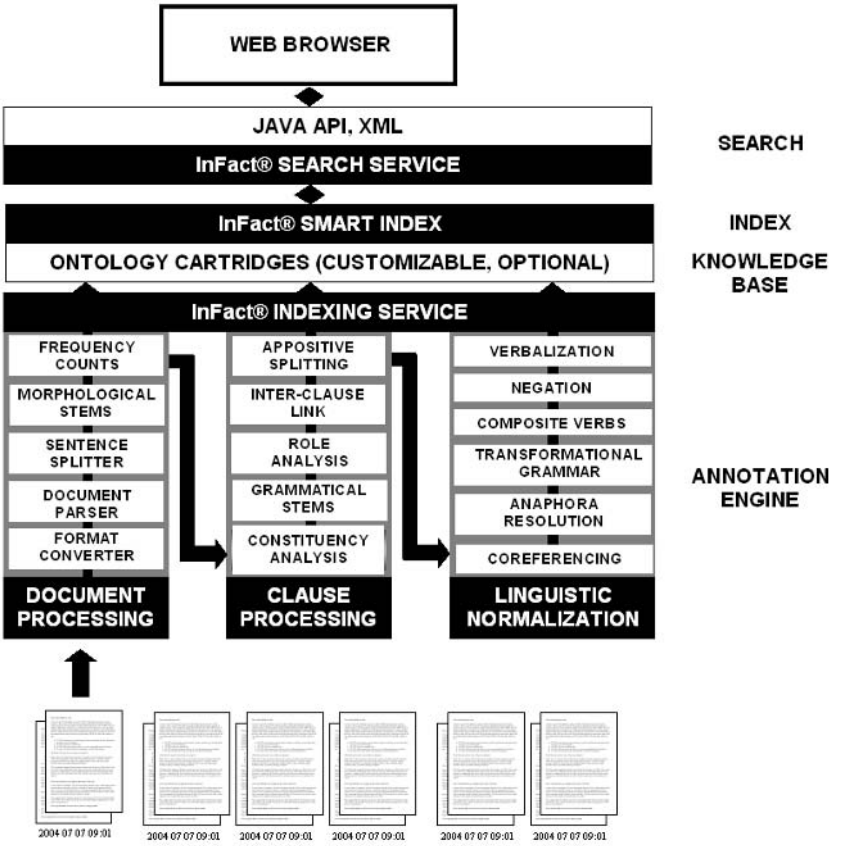


Fig. 5.1. Functional overview of InFact.

Document Processing

The first step in document processing is format conversion, which we handle through our native format converters, or optionally via search export conversion software from Stellant™ (www.stellent.com), which can convert 370 different input file types. Our customized document parsers can process disparate styles and recognized zones within each document. Customized document parsers address the issue that a Web page may not be the basic unit of content, but it may consist of separate sections with an associated set of relationships and metadata. For instance a blog post may contain blocks of text with different dates and topics. The challenge is to automatically recognize variations from a common style template, and segment information in the index to match zones in the source documents, so the relevant section can be displayed in response to a query. Next we apply logic for sentence splitting in preparation for clause processing. Challenges here include the ability to unambiguously recognize sentence delimiters, and recognize regions such as lists or tables that

are unsuitable for deep parsing. Last, we extract morphological stems and compute frequency counts, which are then entered in the index.

Clause Processing

The indexing service takes the output of the sentence splitter and feeds it to a deep linguistic parser. A sentence may consist of multiple clauses. Unlike traditional models that store only term frequency distributions, InFact performs clause level indexing and captures syntactic category and roles for each term, and grammatical constructs, relationships, and inter-clause links that enable it to understand events. One strong differentiator of our approach to information extraction [4, 5, 7, 8, 14, 19] is that we create these indices automatically, without using predefined extraction rules, and we capture all information, not just predefined patterns. Our parser performs a full constituency and dependency analysis, extracting part-of-speech (POS) tags and grammatical roles for all tokens in every clause. In the process, tokens undergo grammatical stemming and an optional, additional level of tagging. For instance, when performing grammatical stemming on verb forms, we normalize to the infinitive, but we may retain temporal tags (e.g., past, present, future), aspect tags (e.g., progressive, perfect), mood/modality tags (e.g., possibility, subjunctive, irrealis, negated, conditional, causal) for later use in search.

Next we capture inter-clause links, through: 1) explicit tagging of conjunctions or pronouns that provide the link between the syntactic structures for two adjacent clauses in the same sentence; and 2) pointing to the list of annotated keywords in the antecedent and following sentence. Note that the second mechanism ensures good recall in those instances where the parser fails to produce a full parse tree for long and convoluted sentences, or information about an event is spread across adjacent sentences. In addition, appositive clauses are recognized, split into separate clauses and cross-referenced to the parent clause.

For instance, the sentence: "Appointed commander of the Continental Army in 1775, George Washington molded a fighting force that eventually won independence from Great Britain" consists of three clauses, each containing a governing verb (appoint, mold, and win). InFact decomposes it into a primary clause ("George Washington molded a fighting force") and two secondary clauses, which are related to the primary clause by an appositive construct ("Appointed commander of the Continental Army in 1775") and a pronoun ("that eventually won independence from Great Britain"), respectively. Each term in each clause is assigned a syntactic category or POS tag (e.g., noun, adjective, etc.) and a grammatical role tag (e.g., subject, object, etc.). InFact then utilizes these linguistic tags to extract relationships that are normalized and stored in an index, as outlined in the next two sections.

Linguistic Normalization

We apply normalization rules at the syntactic, semantic, or even pragmatic level. Our approach to coreferencing and anaphora resolution make use of syntactic agreement and/or binding theory constraints, as well as modeling of referential distance, syntactic position, and head noun [6, 10, 12, 13, 16, 17]. Binding theory places syntactic restrictions on the possible coreference relationships between pronouns and

their antecedents [2]. For instance, when performing pronoun coreferencing, syntactic agreement based on person, gender and number limits our search for a noun phrase linked to a pronoun to a few candidates in the text. In addition, consistency restrictions limit our search to a precise text span (the previous sentence, the preceding text in the current sentence, or the previous and current sentence) depending upon whether the pronoun is personal, possessive, reflective, and what is its person. In the sentence "John works by himself," "himself" must refer to John, whereas in "John bought him a new car," "him" must refer to some other individual mentioned in a previous sentence. In the sentence, ""You have not been sending money," John said in a recent call to his wife from Germany," binding theory constraints limit pronoun resolution to first and second persons within a quotation (e.g., you), and the candidate antecedent to a noun outside the quotation, which fits the grammatical role of object of a verb or argument of a preposition (e.g., wife). Our coreferencing and anaphora resolution models also benefit from preferential weighting based on dependency attributes. The candidate antecedents that appear closer to a pronoun in the text are scored higher (weighting by referential distance). Subject is favored over object, except for accusative pronouns (weighting by syntactic position). A head noun is favored over its modifiers (weighting by head label). In addition, as part of the normalization process, we apply a transformational grammar to map multiple surface structures into an equivalent deep structure. A common example is the normalization of a dependency structure involving a passive verb form into the active, and recognition of the deep subject of such clause. At the more pragmatic level, we apply rules to normalize composite verb expressions, capture explicit and implicit negations, or to verbalize noun or adjectives in cases where they convey action sense in preference to the governing verb of a clause. For instance, the sentences "Bill did not visit Jane," which contains an explicit negation, and "Bill failed to visit Jane," where the negation is rendered by a composite verb expression, are mapped to the same structure.

5.2.2 Storage

The output of a deep parser is a complex augmented tree structure that usually does not lend itself to a tractable indexing schema for cross-document search. Therefore, we have developed a set of rules for converting an augmented tree representation into a scalable data storage structure.

In a dependency tree, every word in the sentence is a modifier of exactly one other word (called its head), except the head word of the sentence, which does not have a head. We use a list of tuples to specify a dependency tree with the following format:

(Label Modifier Root POS Head-label Role Antecedent [Attributes])

where: **Label** is a unique numeric ID; **Modifier** is a term in the sentence; **Root** is the root form (or category) of the modifier; **POS** is its lexical category; **Head-label** is the ID of the term that modifier modifies; **Role** specifies the type of dependency relationship between head and modifier, such as subject, complement, etc; **Antecedent** is the antecedent of the modifier; **Attributes** is the list of semantic attributes that may be associated with the modifier, e.g., person's name, location, time, number, date, etc.

For instance, the parse tree for our Washington example above is shown in Table 5.1.

Table 5.1. The parse tree representation of a sentence.

Label	Modifier	Root	POS	Head Label	Role	Antecedent	Attributes
1	Appointed	Appoint	V				
2	commander		N	1	Obj		Person/title
3	of		Prep	2	Mod		
4	the		Det	5	Det		
5	Continental Army		N	3	Pcomp		Organization/name
6	in		Prep	1	Mod		
7	1775		N	6	Pcomp		Numeric/date
8	George Washington		N	9	Subj		Person/name
9	molded	mold	V				
10	a		Det	12	Det		
11	fighting		A	12	Mod		
12	force		N	9	Obj		
13	that		N	15	Subj	12	
14	eventually		A	15	Mod		
15	won	win	V				
16	independence		N	15	Obj		
17	from		Prep	16	Mod		
18	Great Britain		N	17	Pcomp		Location/country

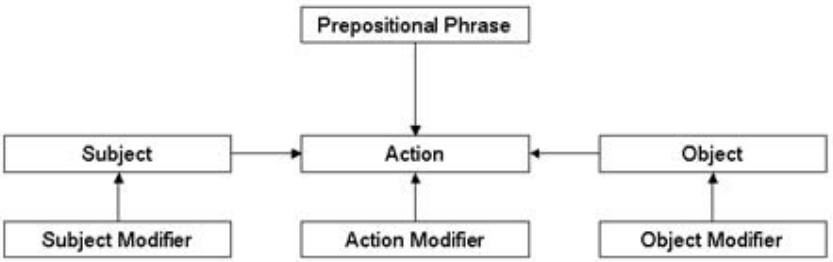


Fig. 5.2. The Subject-Action-Object indexing structure.

The basic idea behind or approach to indexing involves collapsing selected nodes in the parse tree to reduce the overall complexity of the dependency structures.

We model our storage structures after the general notion of subject-action-object triplets, as shown in Figure 5.2. Interlinked subject-action-object triples and their respective modifiers can express most types of syntactic relations between various entities within a sentence.

The index abstraction is presented in Table 5.2, where the additional column *"Dist"* denotes degrees of separations (or distance) between primary *Subject, Verb, Object* and each *Modifier*, and *"Neg"* keeps track of negated actions.

Table 5.2. The index abstraction of a sentence.

Subject	Subject-Modifier	Object	Object-Modifier	Verb	Verb-Modifier	Prep	Pcomp	Dist	Neg
		Washington	George	appoint				1	F
		commander		appoint				1	F
		Army	Continental	appoint				3	F
				appoint		in	1775	2	F
Washington	George	force	fighting	mold				2	F
force	fighting	independence		win				1	F
				win		from	Great Britain	3	F
				win	eventually			1	F

InFact stores the normalized triplets into dedicated index structures that

- are optimized for efficient keyword search
- are optimized for efficient cross-document retrieval of arbitrary classes of relationships or events (see examples in the next section)
- store document metadata and additional ancillary linguistic variables for filtering of search results by metadata constraints (e.g., author, date range), or by linguistic attributes (e.g., retrieve negated actions, search subject modifier field in addition to primary subject in a relationship search)
- (optionally) superimposes annotations and taxonomical dependencies from a custom ontology or knowledge base.

With regard to the last feature, for instance, we may superimpose a *[Country]* entity label on a noun phrase, which is the subject of the verb *"to attack."* The index supports multiple ontologies and entangled multiparent taxonomies.

InFact stores "soft events" instead of fitting textual information into a rigid relational schema that may result in information loss. "Soft events" are data structures that can be recombined to form events and relationships. "Soft events" are pre-indexed to facilitate thematic retrieval by action, subject, and object type. For instance, a sentence like "The president of France visited the capital of Tunisia" contains evidence of 1) a presidential visit to a country's capital and 2) diplomatic relationships between two countries. Our storage strategy maintains both interpretations. In other words, we allow more than one subject or object to be associated with the governing verb of a sentence. The tuples stored in the database are therefore "soft events," as they may encode alternative patterns and relationships found in each sentence. Typically, only one pattern is chosen at search time, in response to a specific user request (i.e., request #1: gather all instances of a president visiting a country; request #2: gather all instances of interactions between any two countries).

5.2.3 Search

Unlike keyword search engines, InFact employs a highly expressive query language (IQL or InFact Query Language) that combines the power of grammatical roles with the flexibility of Boolean operators, and allows users to search for actions, entities, relationships, and events. InFact represents the basic relationship between two entities with an expression of the kind:

Subject Entity > Action > Object Entity,

The arrows in the query refer to the directionality of the action, which could be either uni-directional (as above) or bi-directional. For example,

Entity 1 <> Action <> Entity 2

will retrieve all relationships involving *Entity 1* and *Entity 2*, regardless of their roles as subject or object of the action. Wildcards can be used for any grammatical role. For instance, the query "$*$ > *eat* > *cake*" will retrieve a list of anybody or anything that eats a cake; and a query like "*John* > $*$ > *Jane*" will retrieve a list of all uni-directional relationships between John and Jane. InFact also supports the notion of entity types. For instance, in addition to entering an explicit country name like "Argentina" as Entity 1 or Entity 2 in a relationship query, a user can enter a wildcard for any country name by using the syntax *[Country]*. InFact comes with a generic ontology that includes *[Location]*, *[Person]*, *[Organization]*, *[Numeric]* as the four main branches. Entity types can be organized hierarchically in a taxonomy. IQL renders hierarchical dependencies by means of taxonomy paths. For instance, in *[Entity/Location/Country]* and *[Entity/Location/City]* both *[Country]* and *[City]* nodes have a common parent *[Location]*. Taxonomy path can encode "is-a" relations (as in the above examples), or any other relations defined in a particular ontology (e.g., "part-of" relation). When querying, we can use a taxonomy node in a relationship search, e.g., *[Location]*, and the query will automatically include all subpaths in the taxonomic hierarchy, including *[City]*, *[Location]*, or narrow the search by expanding the path to *[Location/City]*.

With the InFact query language, we can search for:

- Any relationships involving an entity of interest

For example, the query "*George Bush* <> $*$ <> $*$" will retrieve any events involving "*George Bush*" as subject or object

- Relationships between two entities or entity types

For example, the query "*China* <> $*$ <> *Afghan**" will retrieve all relationships between the two countries. Note in this case a wildcard is used in "*Afghan**" to handle different spelling variations of Afghanistan. The query "*Bin Laden* <>$*$<> *[Organization]*" will retrieve any relationships involving "Bin Laden" and an organization.

- Events involving one or more entities or types

For example, the query "*Pope* > *visit* > *[country]*" will return all instances of the Pope visiting a country. In another example, "*[Organization/name]* > *acquire* > *[Organization/name]*" will return all events involving a named company buying another named company.

- Events involving a certain action type

"Action types" are groups of semantically linked actions. For example, query "*[Person] > [Communication] > [Person]*" will retrieve all events involving communication between two people.

InFact's query syntax supports Boolean operators (i.e., AND, OR, NOT). For example, the query:

Clinton NOT Hillary > visit OR travel to > [Location]

is likely to retrieve the travels of *Bill Clinton*, but not *Hillary Clinton*.

We can further constrain actions with modifiers, which can be explicit entities or entity types, e.g., *Paris* or *[location]*. For example, the query

[Organization/Name] > buy > [Organization/Name]ˆ[money]

will only return results where a document mentions a specific monetary amount along with a corporate acquisition. Similarly, the query

Bush <> meet<> Clinton ˆ[location]

will return results restricted to actions that occur in an explicit geographical location.

We can also filter search results by specifying document-level constraints, including:

- Document metadata tags – lists of returned actions, relationships or events are restricted to documents that contain the specified metadata values.
- Boolean keyword expressions – lists of returned actions, relationships or events are restricted to documents that contain the specified Boolean keyword expressions.

For instance, a query like:

[Organization/Name] > buy > [Organization/Name]ˆ[money]; energy NOT oil

will return documents that mentions a corporate acquisition with a specific monetary amount, and also contain the keyword "energy" but do not contain the keyword "oil."

InFact also provides a context operator for inter-clause linking. Suppose for instance, that we want to retrieve all events where a plane crash kills a certain number of passengers. The event could be spread over adjacent sentences, as in: *"The plane crashed shortly after take-off. As many as 224 people were killed."*

In this case, a query like:

** > kill > [numeric] ∼plane crash*

will retrieve all plane crash events, regardless of whether they are contained in a single or multiple, adjacent sentences.

InFact can also support synonyms and query expansion via custom ontologies. In this case, InFact will automatically recognize the equivalence of entities or actions that belong to the same ontology node.

The InFact Query Language rests on a flexible Java Search API. The Java Search API allows us to programmatically concatenate search operators, package and present them to the end user through a simpler interface.

5.3 Architecture and Deployment

We designed both indexing and search as parallel distributed services. Figure 5.3 shows a typical deployment scenario, with an indexing service on the left and a search service on the right. A typical node in each of the diagrams would is a dual processor (e.g., 2.8+GHz Xeon 1U) machine with 4GB of RAM and two 120GB drives.

The Indexing Service (left) processes documents in parallel. Index workers access source documents from external web servers. Multiple index workers can run on each node. Each index worker performs all the "Annotation Engine" analyses described in Figure 5.1. An index manager orchestrates the indexing process across many index workers. The results of all analyses are stored in temporary indices in the index workers. At configurable intervals, the index manager orchestrates the merging of all temporary indices into the partition index components.

A partition index hosts the actual disk based indices used for searching. The contents of a document corpus are broken up into one or more subsets that are each stored in a partition index. The system supports multiple partition indices: the exact number will depend on corpus size, number of queries per second and desired response time. Indices are queried in parallel and are heavily IO bound. Partition indices are attached to the leaf nodes of the Search Service on the right.

In addition to storing results in a temporary index, index workers can also store the raw results of parsing in a Database Management System (DBMS). The database is used almost exclusively to restore a partition index in the event of index corruption. Data storage requirements on the DBMS range between 0.5 and 6x corpus size depending on which recovery options for the InFact system are enabled. Once a document has been indexed and merged into a partition index it is available for searching.

In a typical search deployment, queries are sent from a client application; the client application may be a Web browser or a custom application built using the Search API. Requests arrive over HTTP and are passed through a Web Server to the Search Service layer and on to the top searcher of a searcher tree. Searchers are responsible for searching one or more partition index. Multiple searchers are supported and can be stacked in a hierarchical tree configuration to enable searching large data sets. The top level searcher routes ontology related requests to one or more ontology searchers, which can run on a single node. Search requests are passed to child searchers, which then pass the request down to one or more partition indices. The partition index performs the actual search against the index, and the result passes up the tree until it arrives back at the client for display to the user.

If a particular segment of data located in a partition index is very popular and becomes a search bottleneck, it may be cloned; the parent searcher will load balance across two or three partition indices. In addition, if ontology searches become a bottleneck, more ontology searchers may be added. If a searcher becomes a bottleneck, more searchers can be added. The search service and Web server tier may be replicated, as well, if a load balancer is used.

The example in Figure 5.3 is an example of a large-scale deployment. In the GlobalSecurity.org portal, we currently need only four nodes to support a user community of 100,000 against a corpus of several GB of international news articles, which are updated on a daily basis.

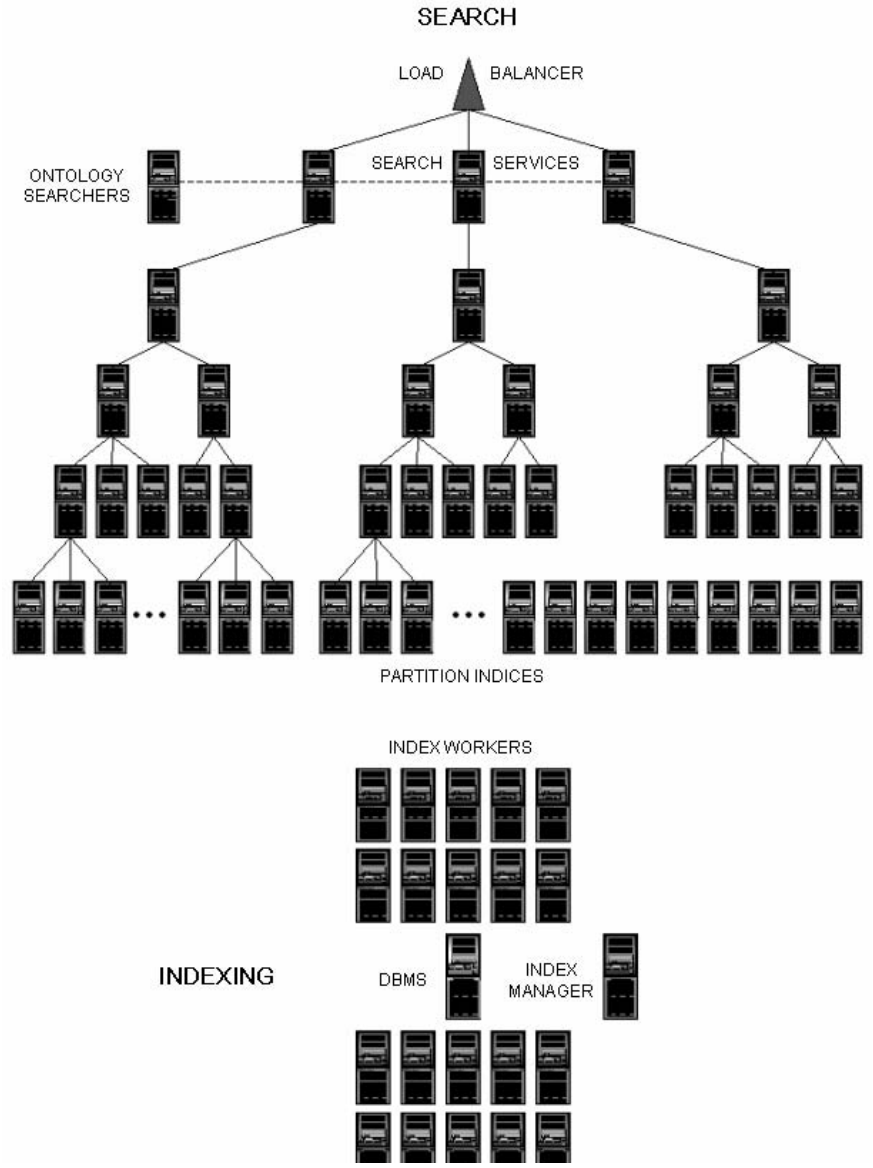


Fig. 5.3. Architectural overview of InFact.

5.4 The GlobalSecurity.org Experience

5.4.1 Site Background

InFact started powering the GlobalSecurity.org Web site on June 22, 2005. Based in Alexandria, VA, and "launched in 2000, GlobalSecurity.org is the most compre-

hensive and authoritative online destination for those in need of both reliable background information and breaking news ... GlobalSecurity.org's unique positioning enables it to reach both a targeted and large diversified audience. The content of the website is updated hourly, as events around the world develop, providing in-depth coverage of complicated issues. The breadth and depth of information on the site ensures a loyal repeat audience. This is supplemented by GlobalSecurity.org's unique visibility in the mass media, which drives additional growth" [9]. The director of GlobalSecurity.org, John Pike, regularly provides commentary and analysis on space and security issues to PBS, CNN, MSNBC, Fox, ABC, CBS, NBC, BBC, NPR, and numerous print and online publications. In powering this site, InFact serves the information search needs of a well-established user community of 100,000, consisting of news reporters, concerned citizens, subject matter experts, senior leaders, and junior staff and interns.

5.4.2 Operational Considerations

When preparing the GlobalSecurity.org deployment, one of our prime concerns was the response time of the system. For this reason, we kept the data size of the partition indices small enough so that most operations occur in memory and disk access is minimal. We split the GlobalSecurity.org data across two index chunks, each containing roughly 14 GB of data in each partition index. Another concern was having sufficient capacity to handle the user load. To account for future user traffic, we specified the deployment for 2-3 times the maximum expected load of about 11,000 queries per day. This left us with two cloned partition indices per index chunk. In addition, we wanted a hot back up of the entire site, in case of any hardware failures, and to support us each time we are rolling out new features.

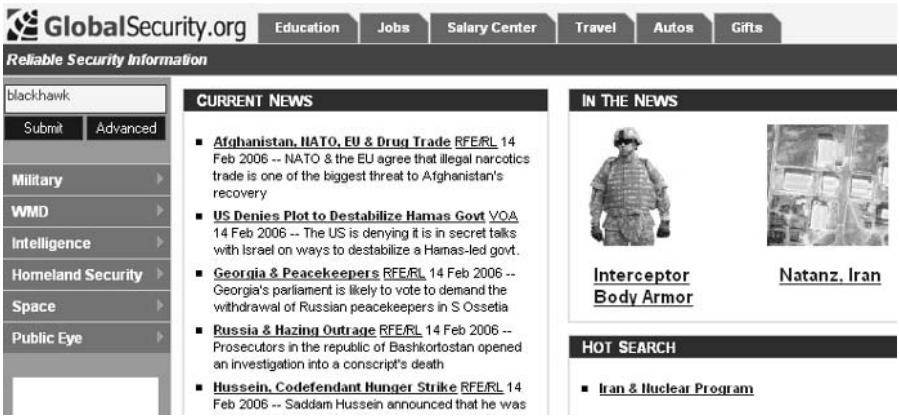


Fig. 5.4. The GlobalSecurity.org home page.

Another area of concern was the distribution of query types. Our system has significantly varying average response time and throughput (measured in queries/minute) depending on the type of queries being executed. We assumed that users

would take some time to migrate from keyword queries to fact queries. Therefore, we selected a very conservative ratio of 50/50 fact-to-keyword query types with a view to adding more hardware if needed. After automatically generating millions of query files, we heavily loaded the system with the queries to simulate heavy traffic using JMeter, a multi-threaded client web user simulation application from the Apache Jakarta organization. Based on these simulations, we deployed with only four nodes.

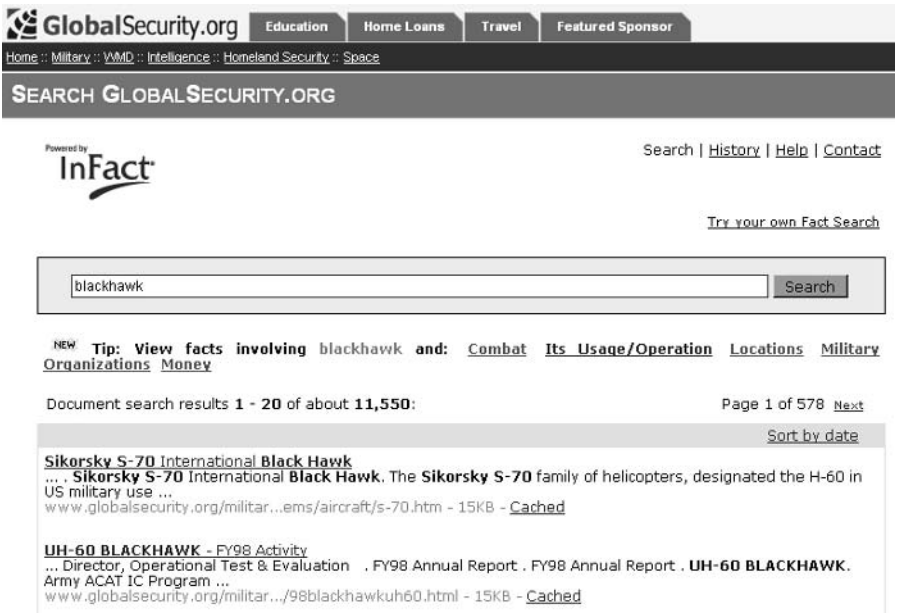


Fig. 5.5. Keyword search result and automatic tip generation with InFact in response to the keyword query "blackhawk."

5.4.3 Usability Considerations

In deploying InFact on the GlobalSecurity.org site, our goal was to serve the information needs of a wide community of users, the majority of which are accustomed to straightforward keyword search. Therefore, on this site, by default, InFact acts as a keyword search engine. However, we also started experimenting with ways to progressively migrate users away from keyword search and towards natural language search or "fact search." With reference to Figure 5.4, users approaching the site can enter InFact queries from the search box in the upper left, or click on the Hot Search link. The latter executes a predefined fact search, which is particularly popular over an extended time period (days or even weeks). The Hot Search is controlled by GlobalSecurity.org staff, and is outside the control of the general user. However, once in the InFact search page (Figure 5.5), the user can execute fact searches explicitly by using the IQL syntax. The IQL syntax is fully documented in the InFact Help page.

Alternatively, by clicking on the "Try your own Fact Search" link on the upper right of the InFact Search page, the user is introduced to a Custom Query Generator (Figure 5.6), which produces the query of Figure 5.7.

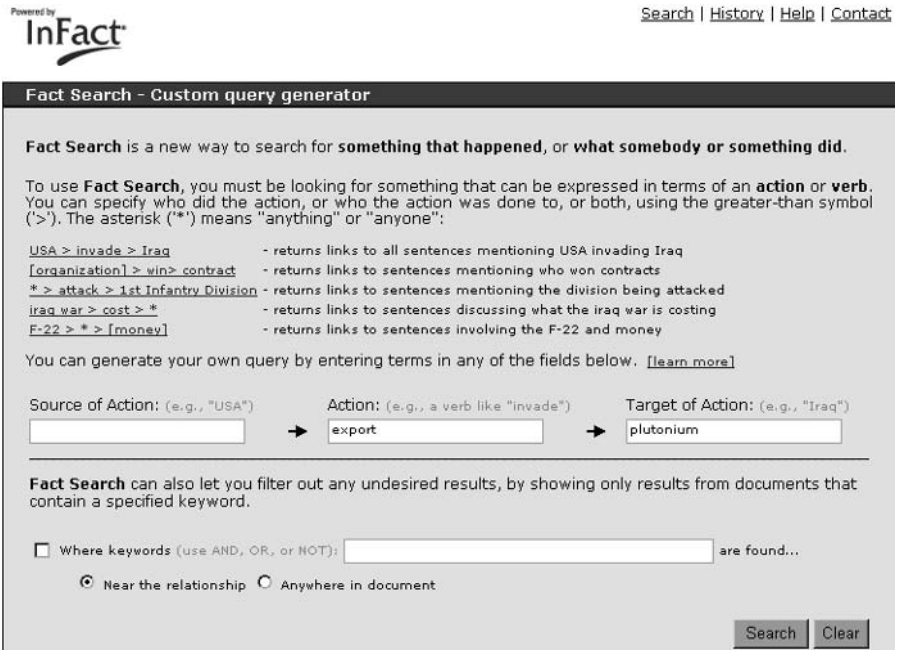


Fig. 5.6. Fact search with the InFact Custom Query Generator: the user is looking for facts that involve the export of plutonium.

The most interesting device we employed is guided fact navigation in response to a keyword entry. We call this process "tip generation." In this scenario, we capture keywords entered by a user and try to understand whether these are names of people, places, organization, military units, vehicles, etc. When executing a keyword search, the InFact system can recommend several fact searches which may be of interest to a user based on the keywords entered. These recommendations are presented to the user as a guided navigation menu consisting of links. In the example of Figure 5.5, the user is performing a keyword search for "blackhawk." The user sees a series of links presented at the top of the result set. They read: "Tip: View facts involving blackhawk and: Combat, Its Usage/Operation, Locations, Military Organizations, Money." Each of these links when clicked in turn executes a fact search. For instance, clicking on Military Organizations will generate the list of facts or relationships of Figure 5.8, which gives an overview of all military units that have used the blackhawk helicopter; clicking on Money will generate the list of facts or relationships of Figure 5.9, which gives an overview of procurement and maintenance costs, as well as government spending for this vehicle. The relationships are returned in a table display where each row is an event, and columns identify the three basic semantic

Fig. 5.7. Fact search with the InFact Custom Query Generator: InFact translates the query of Figure 5.6 into the InFact Query Language (IQL) and returns a list of results. IQL operators are fully documented in the Help page.

roles of source (or subject), action (or verb), and target (or object). Note that relationships, by default, are sorted by relevance to a query, but can also be resorted by date, action frequency, or alphabetically by source, action or target. Each of the relationships or facts in the table is in turn hyperlinked to the exact location in the source document where it was found, so the user can quickly validate the findings and explore its context (Figure 5.10).

Usage logs were the primary driver for this customization effort. The personnel at GlobalSecurity.org were very helpful and provided us with many months of user traffic Web logs. We wrote some simple scripts to analyze the logs. For example, we studied the 500 most popular key word searches performed on the site ranked in order of popularity. Next, we began looking for entity types that would be helpful to the most number of users. We found a lot of user interest in weapons, terrorists, and US officials, amongst other things. We then set about creating ontologies for each of these areas. New custom ontologies can easily be mapped into the internal InFact ontology XML format.

5.4.4 Analysis of Query Logs

We wish to quantify the relative popularity of natural language (Fact) search versus keyword search. In addition, we wish to compare the relative success of alternative strategies we adopted to overcome usability issues. This study of log data reflect

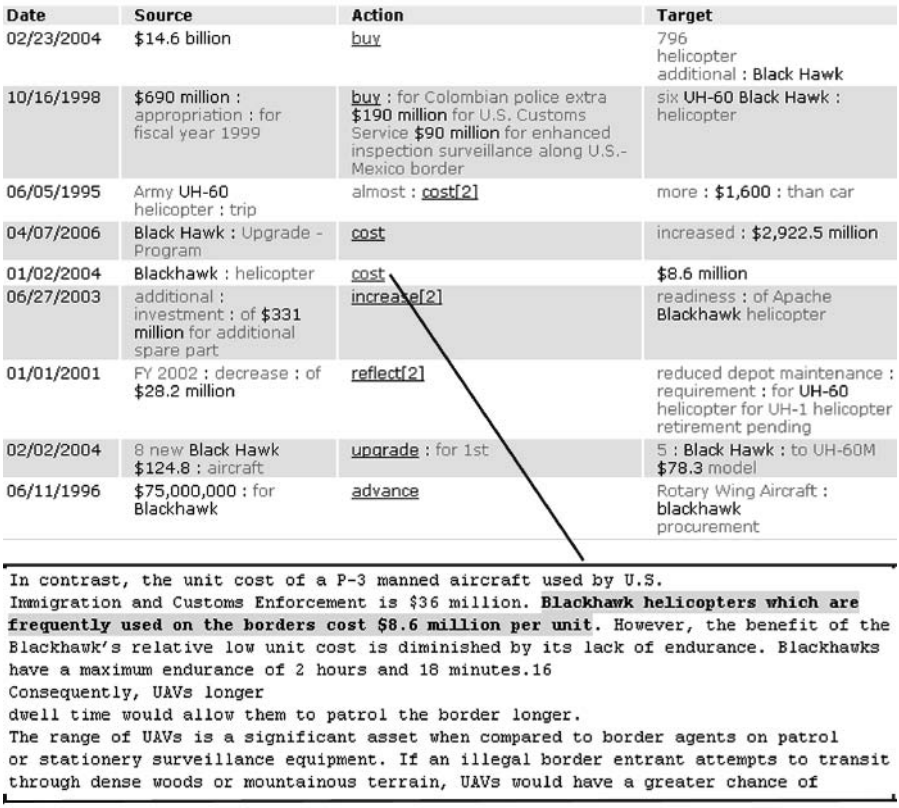
Chinook All : BLACK HAWK	include	160th Special Operations Aviation Regiment 101st Airborne Division 10th Mountain Division 82nd Airborne Division
Marine Corps	inspect : across airfield	rigging : on Blackhawk
U.S. Army	install	UH-60 : in future ALQ-211 : on CH-47
U.S. Army	introduce : for example	fly-by-wire : capability : to Army Apache Black Hawk fleet
air force	investigate	Black Hawk fratricide : incident
Troops : from 2nd Battalion 505th Parachute Infantry Regiment 82nd Airborne Division	kick off	Operation Desert Lion : with air assault from Chinook Black Hawk helicopter
Black Hawk : helicopter	land : As part of Operation Falcon Sweep	Shakaria Soldier : of 2nd Battalion 502nd Infantry Regiment
40 : CH-47 UH-60 AH-1	lift	1st Brigade : into
211th Aviation Group	maintain	UH-60 Blackhawk AH-64 Apache
air force : contractor	maintain	Blackhawk : helicopter Army : Apache

Fig. 5.8. Tip Navigation with InFact: facts involving the "blackhawk" helicopter and military organizations.

Date	Source	Action	Target
02/23/2004	$14.6 billion	buy	796 helicopter additional : Black Hawk
10/16/1998	$690 million : appropriation : for fiscal year 1999	buy : for Colombian police extra $190 million for U.S. Customs Service $90 million for enhanced inspection surveillance along U.S.-Mexico border	six UH-60 Black Hawk : helicopter
06/05/1995	Army UH-60 helicopter : trip	almost : cost[2]	more : $1,600 : than car
04/07/2006	Black Hawk : Upgrade - Program	cost	increased : $2,922.5 million
01/02/2004	Blackhawk : helicopter	cost	$8.6 million
06/27/2003	additional : investment : of $331 million for additional spare part	increase[2]	readiness : of Apache Blackhawk helicopter
01/01/2001	FY 2002 : decrease : of $28.2 million	reflect[2]	reduced depot maintenance : requirement : for UH-60 helicopter for UH-1 helicopter retirement pending
02/02/2004	8 new Black Hawk $124.8 : aircraft	upgrade : for 1st	5 : Black Hawk : to UH-60M $78.3 model
06/11/1996	$75,000,000 : for Blackhawk	advance	Rotary Wing Aircraft : blackhawk procurement

Fig. 5.9. Tip Navigation with InFact: facts involving the "blackhawk" helicopter and money.

four ways users submit a natural language query to InFact: 1) they click on the Hot Search link; 2) they click on a keyword tip; 3) they click on an example in the Query Generator or Help page; 4) they attempt to type an explicit relationship or fact search using the IQL syntax.



Date	Source	Action	Target
02/23/2004	$14.6 billion	buy	796 helicopter additional : Black Hawk
10/16/1998	$690 million : appropriation : for fiscal year 1999	buy : for Colombian police extra $190 million for U.S. Customs Service $90 million for enhanced inspection surveillance along U.S.-Mexico border	six UH-60 Black Hawk : helicopter
06/05/1995	Army UH-60 helicopter : trip	almost : cost[2]	more : $1,600 : than car
04/07/2006	Black Hawk : Upgrade - Program	cost	increased : $2,922.5 million
01/02/2004	Blackhawk : helicopter	cost	$8.6 million
06/27/2003	additional : investment : of $331 million for additional spare part	increase[2]	readiness : of Apache Blackhawk helicopter
01/01/2001	FY 2002 : decrease : of $28.2 million	reflect[2]	reduced depot maintenance : requirement : for UH-60 helicopter for UH-1 helicopter retirement pending
02/02/2004	8 new Black Hawk $124.8 : aircraft	upgrade : for 1st	5 : Black Hawk : to UH-60M $78.3 model
06/11/1996	$75,000,000 : for Blackhawk	advance	Rotary Wing Aircraft : blackhawk procurement

In contrast, the unit cost of a P-3 manned aircraft used by U.S. Immigration and Customs Enforcement is $36 million. **Blackhawk helicopters which are frequently used on the borders cost $8.6 million per unit.** However, the benefit of the Blackhawk's relative low unit cost is diminished by its lack of endurance. Blackhawks have a maximum endurance of 2 hours and 18 minutes.16
Consequently, UAVs longer
dwell time would allow them to patrol the border longer.
The range of UAVs is a significant asset when compared to border agents on patrol or stationery surveillance equipment. If an illegal border entrant attempts to transit through dense woods or mountainous terrain, UAVs would have a greater chance of

Fig. 5.10. Tip Navigation with InFact: each fact is hyperlinked to the exact location where it was found in the source document.

At the time of this writing, an average of 36% of advanced search users click on the hot search link "Iran and Nuclear program," which executes a predefined search like *"Iran* $> * \sim$ *nuclear."* However, it is difficult to assess what the user experience is like because in 80% of cases the user performs non-search-related tasks, and therefore we don't know how long they spent looking at the results. Note that users clicking on this link may not realize that they are going to a search engine page, since the link title is ambiguous. The results of this search are quite good, and still relevant. The hot search is an important entry point into our search site, as 36% of all the fact search queries executed came from this source. It seems likely that adding more of these hot search links or otherwise accentuating them on the page would significantly increase user exposure to natural language based queries.

Our analysis of query logs shows that keyword tips are the most effective way to migrate users to Fact Search. Users who click on tips frequently follow up with

queries of their own. Tip clickers also write better queries, probably because, after seeing the column display, they have a much better sense of how queries can be composed. Keyword tip clickers typically find the results engaging enough to spend an average of 1.5 minutes studying the results: 37% of users go back and click on more than one tip. Even better, 87% follow up by clicking on the "Try your own Fact Search" link and try their own query. All of the queries attempted are queries; 90% produce results; our follow up analysis suggests that for two thirds of these queries the results are relevant to the users search goals. In other words, users who click on the tips are extremely likely not only to try their own fact search, but also to pay enough attention to the format to write both valid and useful queries.

Examples in the Help File or Query Generator are largely ineffective at getting users to try Fact Search. Because the results returned by the examples usually do not necessarily relate to what the user wishes to search on, the column display is more of a distraction than an enticement to try Fact Search. However, those who go on to try Fact Search, after clicking on an example, have a better chance of writing good queries. Example link clickers are less likely to experiment with Fact Search or invest time learning how it works. Seventy-two percent of users end their session after clicking on one or more examples, not even returning to perform the keyword search that presumably brought them to the site in the first place. Of the 28% who did not leave the site after clicking an example, two thirds went on to try a Fact Search. Only 6% of users click on examples after having tried a Fact Search query on their own. Analysis of this user group suggests that examples have a place in the UI, but are not sufficiently compelling to motivate users to try Fact Search alone. However, this evidence does lend support to the hypothesis that users who see the column display are more likely to create valid queries: 60% of the users who click on examples and go on to write their own queries write valid queries and get results, which is still a much higher percentage than for users who blindly try to create queries.

About 75% of users who try Fact Search directly by using the IQL syntax, and without seeing the column display first fail to get results. Forty-five percent of users write invalid queries where nouns are inserted in the action field (the most common error). Another common error is specifying too much information or attaching prepositions to noun phrases. We can detect some of these errors automatically, and we plan to provide automatic guidance to users going forward. About 20% of query creators get impressive results. Most successful users get their queries right on the first shot, and, in general, seem unwilling to invest much time experimenting. Successful users are most likely expert analysts. In reproducing their searches and inspecting their results, we estimate that they have a positive impression of Fact search. In 75% of cases the results of Fact Search take direct the user quickly to the relevant parts of relevant documents, providing a deeper overview and faster navigation of content. However, in 25% of cases, expert users also write queries that return no results. Reasons for this include specifying too much information or including modifiers or prepositional terms in the verb field such as: "cyber attack," "led by," and "go to." In many cases users would be successful by just entering the verb. In some cases, users get lots of fact search results, but lack the experience to refine their query, so they simply go back to keyword search. We should try to communicate how queries can be modified further if there are too many results, perhaps by adding an ontology tag, or a context operator to the query syntax. For instance, the query *"Bush > meet > [person]"* could yield a large number of irrelevant results, if

a user is only interested in a list of diplomatic meetings. The query can be refined as *"Bush >meet > [person/name]."* In this case, the addition of an ontology tag restricts the number of meetings to those that are likely to involve named political personalities of some relevance. If the user is primarily interested in meeting that involve talks on nuclear arms control, the query can be further refined as *"Bush > meet > [person/name] ∼ nuclear arms control."* Similarly, the query *"[country] > produce > uranium"* can be turned into the query *"[country] > produce >[numeric] uranium"* if a user is after quantities of uranium that are being produced around the world. In general, we observe that users accustomed to keyword search believe that specifying more terms translates into more accurate results. In moving these users to Fact Search we must encourage them to start as simple as possible, since the IQL can express in two words what would take 20 lines using Boolean language.

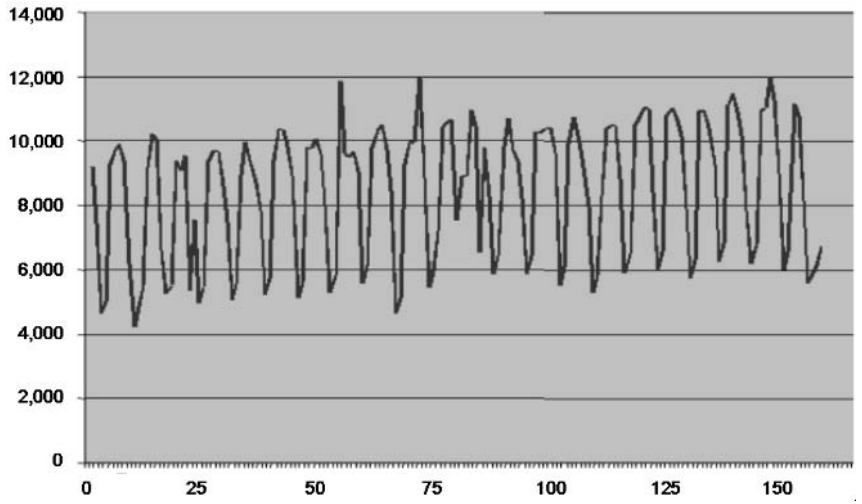


Fig. 5.11. Queries/day vs day of operation (June 22, 2005, to November 30, 2005).

Finally, Figure 5.11 shows overall query volumes (keyword search and Fact Search) as a function of day from the first day of operation (June 22 to November 30, 2005). The cyclic nature of the graph derives from the fact that most user access the site during the working week. Figure 5.12, which displays query volumes vs week of operation, clearly shows a positive trend: overall traffic to the site has increased by almost 40% ever since we introduced InFact search. The most interesting metrics relate to the percentage of users that derive value from Fact Search. The most effective mechanism to promote natural language search, as we have seen, are the tips. Figure 5.13 shows a 60% increase in the number of users that click on the tips automatically generated by InFact's advanced linguistic analysis over our entire period of operation. The overall percentage has increased from 4% to 10%. Our analysis also suggests that the best way to teach users how to write good queries is to first expose them to the summary result displays that ensues from a natural language query. The sooner users become aware of the type of results that a natural

language query can yield, the higher the chances that they learn how to use the new search functions correctly. This reinforces the idea that the result display may be a driver of Fact Search.

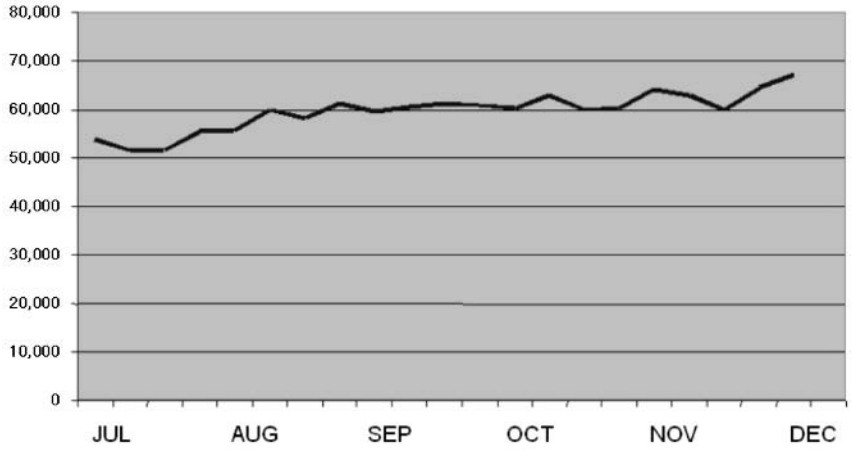


Fig. 5.12. Queries/week vs week of operation (June to November, 2005).

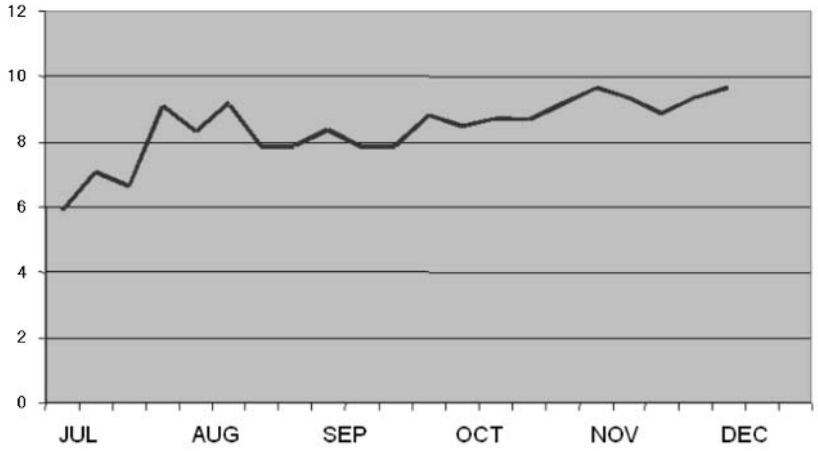


Fig. 5.13. Percentage of tips clicked (June to November, 2005).

5.5 Conclusion

We deployed a natural language based search to a community of Web users, and measured its popularity relative to conventional keyword search. Our work addressed criticisms of NLP approaches to search to the effect that they are not scalable and are too complex to be usable by average end-users. Our approach rests on a sophisticated index parameterization of text content, that captures syntactic and semantic roles, in addition to keyword counts, and enables interactive search and retrieval of events patterns based on a combination of keyword distributions and natural language attributes. Our distributed indexing and search services are designed to scale to large document collections and large numbers of users. We successfully deployed on a Web site that serves a community of 100,000 users. An analysis of query logs shows that, during the first six months of operation, traffic has increased by almost 40%. Even more significantly, we are encountering some success in promoting natural language searches. Our study demonstrates that the percentage of users that avail themselves of guided fact navigation based on natural language understanding has increased from 4% to 10% during the first six months of operation. Going forward, we will focus on increasing this percentage with a more innovative UI.

5.6 Acknowledgments

This work was partially supported by Dr. Joseph Psotka of the US Army Research Institute under contract No. W74V8H-05-C-0016. We are also indebted to John Pike, director of GlobalSecurity.org and his staff for providing us with many months of user traffic Web logs prior to going live.

References

1. D. Appelt and D. Israel. Introduction to information extraction technology. IJCAI-99 tutorial. http://www.ai.sri.com/∼appelt/ie-tutorial/ijcai99.pdf.
2. D. Appelt and D. Israel. Semantic approaches to binding theory. In *Proceedings of the Workshop on Semantic Approaches to Binding Theory. ESSLLI*, 2003.
3. A. Arampatzis, T. van der Weide, P. van Bommel, and C. Koster. Linguistically-motivated information retrieval. In M. Dekker, editor, *Encyclopedia of Library and Information Science*, Springer Verlag, volume 69, pages 201–222. 2000.
4. C. F. Baker, C. J. Fillmore, and J. B. Lowe. The Berkeley FrameNet project. In C. Boitet and P. Whitelock, editors, *Proceedings of the Thirty-Sixth Annual Meeting of the Association for Computational Linguistics and Seventeenth International Conference on Computational Linguistics*, pages 86–90, San Francisco, California, 1998. Morgan Kaufmann Publishers.
5. I. Dagan, O. Glickman, and B. Magnini. The pascal recognizing textual entailment challenge. In *Proceedings of the PASCAL Challenges Workshop Recognizing Textual Entailment*, 2005.
6. M. Dimitrov. A light-weight approach to coreference resolution for named entities in text. Master's thesis, University of Sofia, 2002.

7. D. Gildea and D. Jurafsky. Automatic labeling of semantic roles. *Computational Linguistics*, 28(3):245–288, 2002.
8. D. Gildea and M. Palmer. The necessity of parsing for predicate argument recognition. In *Proceedings of the 40th Meeting of the Association for Computational Linguistics (ACL 2002)*, pages 239–246, 2002.
9. GlobalSecurity.org. http://www.globalsecurity.org/org/overview/history.htm.
10. M. Kameyama. Recognizing referential links: An information extraction perspective. In *Proceedings of the ACL'97/EACL'97 Workshop on Operation Factors in Practical, Robust Anaphora Resolution*, pages 46–53, 1997.
11. A. Kao and S. Poteet. Report on KDD conference 2004 panel discussion can natural language processing help text mining? *SIGKDD Explorations*, 6(2):132–133, 2004.
12. C. Kennedy and B. Boguraev. Anaphora for everyone: Pronominal anaphora resolution without a parser. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING'96)*, pages 113–118, 1996.
13. S. Lappin and H. Leass. An algorithm for pronominal anaphora resolution. *Computational Linguistics*, 20(4):535–561, 1994.
14. D. Lin and P. Pantel. DIRT - discovery of inference rules from text. In *Knowledge Discovery and Data Mining*, pages 323–328, 2001.
15. C. Manning and H. Schutze. *Foundation of Statistical Natural Language Processing.* The MIT Press, 2000.
16. R. Miltov. Robust pronoun resolution with limited knowledge. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING'98)/ACL'98*, pages 869–875.
17. R. Miltov. Anaphora resolution: The state of the art. Working paper. University of Wolverhamption, 1999.
18. National Institute of Standards and Technology. Automatic content extraction (ACE). http://www.itl.nist.gov/iaui/894.01/tests/ace.
19. M. Surdeanu, S. Harabagiu, J. Williams, and P. Aarseth. Using predicate-argument structures for information extraction. In *41th Annual Meeting of the Association for Computational Linguistics*, pages 8–15, 2003.