Learning SQL for Database Intrusion Detection Using Context-Sensitive Modelling (Extended Abstract)

Christian Bockermann[1], Martin Apel[2], and Michael Meier[2]

[1] Artificial Intelligence Group
christian.bockermann@udo.edu
[2] Information Systems and Security Group
{martin.apel,michael.meier}@udo.edu

Department of Computer Science
Technische Universität Dortmund

Abstract. Modern multi-tier application systems are generally based on high performance database systems in order to process and store business information. Containing valuable business information, these systems are highly interesting to attackers and special care needs to be taken to prevent any malicious access to this database layer. In this work we propose a novel approach for modelling SQL statements to apply machine learning techniques, such as clustering or outlier detection, in order to detect malicious behaviour at the database transaction level. The approach incorporates the parse tree structure of SQL queries as characteristic e.g. for correlating SQL queries with applications and distinguishing benign and malicious queries. We demonstrate the usefulness of our approach on real-world data.

1 Introduction

The majority of today's web-based applications does rely on high performance data storage for business processing. A lot of attacks on web-applications are aimed at injecting commands into database systems or try to otherwise trigger transactions to gain unprivileged access to records stored in these systems. See [1] for a list of popular attacks on web applications.

Traditional network-based firewall systems offer no protection against these attacks, as the malicious (fractions of) SQL or tampered requests are located at the application layer and thus are not visible to most of these systems.

The usual way of protecting modern application systems is by introducing detection models on the network layer or by the use of web application firewall systems. These systems often employ a misuse detection approach and try to detect attacks by matching network traffic or HTTP request against a list of known attack patterns. A very popular system based on pattern matching is for instance the Snort IDS [2]. Another project aiming at the detection of tampered HTTP requests is the ModSecurity module, which provides a rule-engine for employing pattern based rules within a Web-Server [3].

U. Flegel and D. Bruschi (Eds.): DIMVA 2009, LNCS 5587, pp. 196–205, 2009.

Instead of using pattern based approaches, there exists a variety of papers on employing anomaly-based methods for detecting web-based intrusions [4,5,6]. These either try to analyze log-files or protocol-level information to detect anomalies based on heuristics or data-mining techniques. We earlier proposed a rule based learning approach using the ModSecurity module in [7].

These approaches are rooted at the network or application protocol layer. In this work we focus on the detection at the database layer, i.e. the detection of anomalous SQL statements, that are either malicious in the sense that they include parts of injected code or differ from the set of queries usually issued within an application. The main contribution of our work is the use of a grammar based analysis, namely tree-kernel based learning, which became popular within the field of natural language processing (NLP). Our approach incorporates the parse tree structure of SQL queries as characteristic e.g. for correlating SQL queries with applications and distinguishing benign and malicious queries. By determining a context sensitive similarity measure we can locate the nearest legal query for an malicious statements which tremendously helps in root cause analysis.

The remainder of this paper is organized as follows: Section 2 states the problem in detail and gives an overview of related work regarding intrusion detection in databases. In Section 3 we give a short introduction to kernel-based learning algorithms in general and their application on structured data in detail. Following this overview we define our tree-kernel based method and describe its application to learning SQL for intrusion detection in databases in Section 4. Finally we present our results on real-world data in Section 5 and summarize our experiments.

2 Problem and Related Work

Executing malicious statements on a database may result in severe problems, which can range from exposure of sensitive information to loosing records or broken integrity. Once an attacker manages to inject code into a database this will likely not only affect specific records, but may lead to a compromise of the complete application environment. This in turn can cause severe outages with respect to data records and a company's public reputation.

Although the risk may seem low on a first glance, given the database layer is separated from the public interface (web/presentation layer) and not directly accessible from the outside, anomalous queries caused by e.g. SQL injection attacks are a widespread problem. The *Web Hacking Incident Database* provides a listing of recent web hacks, a lot of them relying on SQL injections [8].

There have been approaches to apply data-mining and machine learning methods to detect intrusions in databases. Lee et al [9] suggest learning fingerprints of access patterns of genuine database transactions (e.g. read/write sequences) and using them to identify potential intrusions. Typically there are many possible SQL queries, but most of them only differ in constants that represent the user's input. SQL queries are summarized in fingerprints (regular expressions) by replacing the constants with variables or wild-cards. Such fingerprints capture some structure of the SQL queries. Following the approach of [9], queries

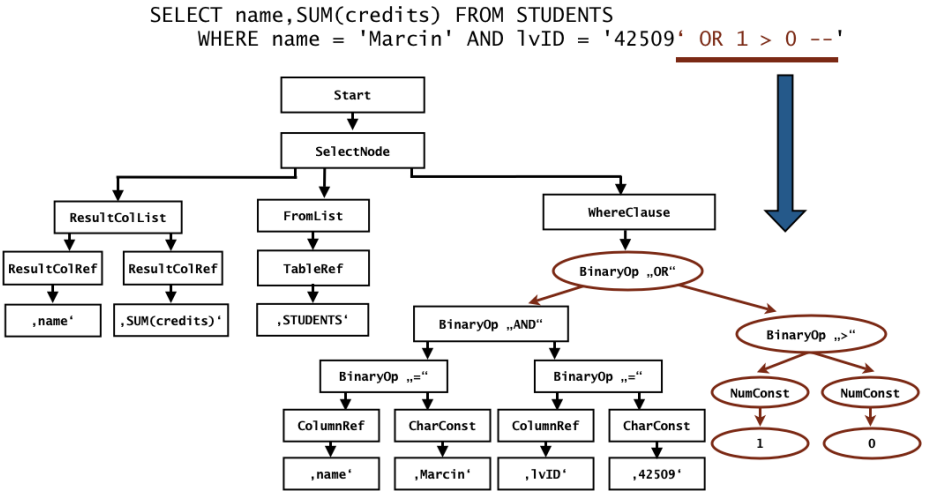

Fig. 1. SQL parse tree of an SQL injection

not matching any of the existing fingerprints are reported as malicious. A drawback of this approach is its inability to correlate and identify fingerprints with applications.

In [10] the authors also try to detect SQL injections by a kind of fingerprints. They use parse trees of queries as fingerprints for the queries structure. The main idea here is to compare the parse tree of an SQL statement before and after user-variables have been inserted. Injected SQL fragments will typically significantly change the trees structure. An example of such structural changes in the parse tree of a query is shown in figure 1. In this figure, the rounded nodes of the tree indicate the additional parts that have been added due to the injection SQL fragment `' OR 1 > 0 --`. As this work only uses a one-to-one comparison on parse-trees it is missing any generalization capabilities and thus not applicable for machine learning methods, such as clustering and outlier detection.

A similar grammar-based approach has been used in [11], which studied the use of syntax-aware analysis of the FTP protocol using tree-kernel methods on protocol parse-trees. A slightly different approach was taken in [12] where the parse tokens are used along with their values to detect anomalies in HTTP-traffic. The latter approach does not use the full parse tree but its leaves. Our work is similar to [11,12] in the sense that it employs machine learning methods on syntax trees derived from a protocol parser.

Also approaches on investigating data dependencies have been proposed in [13] and [14]. Data dependencies refer to access correlations among sensitive data items. Data dependencies are generated in form of classification rules like *before an update of item1 a read of item2 is likely.* Transactions not compliant to these rules are flagged as malicious. Srivastava et al [14] further distinguish different levels of sensitivity of data items which need to be specified by hand.

Both approaches ignore the structure of SQL queries and are unable to correlate SQL queries with applications. A more recent work has been presented in[15], focusing on the sequential nature of SQL queries. These studies also make use of a smart modelling technique to easily apply data mining methods on their SQL representations.

3 A Grammar-Based Modelling

Since most learning approaches work on vectorized data, a key issue when using machine learning for intrusion detection is the representation of monitored data to apply any learning algorithm. A popular technique in IDS is the exhaustive creation of n-grams, yielding histogram vectors for observed input data. These do not maintain any syntactical information of SQL. A little more syntax is regarded by creating *term-vectors* of a query. A *term-vector* can be obtained by splitting the query in a "proper way", i.e. by splitting on whitespace characters (optionally maintaining quoted strings).

As in this work we are dealing with the detection of malicious database queries, we choose a grammar based approach to represent SQL queries. We propose two alternative modelling approaches for making SQL queries suitable for machine learning.

3.1 Parsing SQL

The basic idea of [10] is to detect SQL injection attacks by means of changes in a queries syntax tree. An example of such a tree has been shown before (see figure 1). In order to obtain such a parse tree, a parser for the SQL dialect is required. Usually complex parsers are automatically generated based on a given grammar description using tools such as *yacc*, *antlr* or *javacc*. Unfortunately, the availability of proper grammar descriptions for SQL is pretty sparse and most existing parser implementations are tightly wired into the corresponding DBMS, making it laborious to extract a standalone parser.

We therefore decided to modify an existing open-source DBMS, in our case the Apache Derby database, which provides a standalone deployment. The Derby parser is itself generated off a grammar file using *javacc*, but does not explicitly output a syntax tree suitable for our decomposition. Using the tree-interface of the parser, we derived a tree-inspection tool which traverses the tree object of a query and writes out the corresponding node information.

3.2 Vectorization of SQL Queries

To incorporate more syntax, we determine the parse tree of a query. As we are interested in the detection of abnormal queries within our database application, we are looking for a similarity measure for the space of structured objects, i.e. the space of valid SQL parse trees. Thus, we are faced with the problem of having to create a distance function for matching trees.

Definition: Let q be an SQL query and τ_q the parse tree of q, identifying with τ_q the root node of the tree. Each node n within that tree is labeled with an identifier $\text{type}(n)$*, reflecting the node type.*

For a node n within τ_q we denote by $\text{succ}(n)$ *the ordered set of successors of n and by* $\text{succ}_i(n)$ *the ith child of n.*

This definition is basically just a formalization of a query's syntax tree. It allows us to enlist the production or grammar rules, which generate a given SQL query q. This list of production rules will be defined as follows:

Definition: For a node n within the parse tree τ_q of a query q, the list of production rules $P(n)$ is given by

$$P(n) = \biguplus_{c \in \text{succ}(n)} \{\text{type}(n) \to \text{type}(c)\} \quad \uplus \quad \biguplus_{c \in \text{succ}(n)} P(c).$$

Given $P(n)$, denote by $|P(n)|_r$ the number of times the rule r occurs in $P(n)$.

Please note that we use the $\uplus$ notation here for list concatenation, thus, the resulting list may contain the same rule more than once. Now, denoting with Q the set of all valid trees for a given SQL dialect, these simple definitions allow us to define a mapping $\varphi : Q \to \mathbb{R}^n$, by following the *bag of words* approach known from text classification tasks like *spam detection* as proposed in [16].

Definition: Let R be the sorted set of all possible production rules, defined by some SQL grammar and r_i the ith rule of R. For an SQL query q with the associated parse tree τ_q the rule vector $\mathbf{v} \in \mathbb{R}^{|R|}$ is given by $v_i = |P(\tau_q)|_{r_i}$.
The function φ maps an SQL query q to the vector space $\mathbb{R}^{|R|}$ by $\varphi(q) = \mathbf{v}$.

Since an SQL query usually consists of only a small fraction of the complete SQL grammar, these rule vectors are typically very sparse. Based on this mapping we can now define a distance measure on SQL queries using any distance function Δ in the vector space $\mathbb{R}^{|R|}$ by defining the corresponding distance function Δ_{SQL} using

$$\Delta_{SQL}(q_1, q_2) := \Delta(\varphi(q_1), \varphi(q_2)), \tag{1}$$

where q_1, q_2 are any two SQL statements of a common dialect. This allows for the application of a wide range of distance based learning algorithms such as clustering or outlier detection.

4 Using Tree-Kernels for SQL Grammars

The simple vectorization of SQL queries defined above includes a weak context based reasoning to be used within the distance measure in $\mathbb{R}^{|R|}$. It can be seen as an an *explicit feature extraction* approach, as it explicitly creates feature vectors from SQL statements. Unfortunately, the rule counting does only incorporate direct antecessor relationships, limiting the contextual scope.

4.1 Introduction to Tree-Kernels

To overcome these limitations the natural language processing community makes use of context based tree-kernels, which provide a so-called *kernel-function* over trees. In the machine learning community kernel-based methods have received a lot of attention not ultimately owing to the well-known *support vector machine* method, which has also been used for intrusion detection [17,12]. These methods make use of a kernel-function to measure the similarity between instances of some input space $\mathcal{X}$, i.e. a kernel k is symmetric and positive (semi-) definite function

$$k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$$

which implicitly computes an inner product in a reproducing kernel Hilbert space. There exists kernel functions for complex structures like trees or graphs, which are often defined as convolution kernels [18]. For these kernels one defines a kernel over atomic structures and defines the convolution kernel for complex objects by combining the kernel function of its sub structures.

In [19] Collins and Duffy propose a simple kernel over trees for use in natural language processing. The basic idea is to capture structural information over trees in the kernel function by incorporating all sub-trees occuring within the trees of interest. Let $\mathcal{T}$ be the space of all trees in question and denote with T the ordered set of all possible sub-trees in $\mathcal{T}$. For a tree $\tau \in \mathcal{T}$ denote by $h_i(\tau)$ the number of occurrences of the i-th sub-tree of $\mathcal{T}$ in τ and with $N(\tau)$ the set of all nodes in τ. For two trees τ_1, τ_2 the tree-kernel in [19] is defined by

$$K_C(\tau_1, \tau_2) = h_i(\tau_1)h_i(\tau_2) = \sum_{n_1 \in N(\tau_1), n_2 \in N(\tau_2)} \Delta(n_1, n_2).$$

The function Δ is defined as follows

$$\Delta(n_1, n_2) = \begin{cases} 0 & \text{if } P(n_1) \neq P(n_2) \\ \lambda & \text{if height}(n_1) = \text{height}(n_2) = 1 \\ \Delta^*(n_1, n_2) & \text{otherwise,} \end{cases}$$

where $\Delta^*(n_1, n_2)$ is recursively defined as

$$\Delta^*(n_1, n_2) = \lambda \prod_{k=1}^{|\,\text{succ}(n_1)|} [1 + \Delta(\text{succ}_k(n_1), \text{succ}_k(n_2))]$$

Roughly speaking, this kernel measures the similarity of two trees by the set of common sub trees. As it does not consider the context of a sub tree, Zhou et al [20] designed a *context-sensitive convolution* tree-kernel, by taking into account a sub trees' context by means of its ancestors.

Starting with a tree τ, a root node path of length l in τ is a path from the root node τ or any of its successors to a node in τ, which has a length of l. Following the notation of [20], the set of all root node paths for a tree τ_j with a maximal

length of m is denoted by $N^m[j]$. Given a maximum length m for the root node paths considered, the context-sensitive tree-kernel is given be

$$K_{CSC}(\tau_1, \tau_2) = \sum_{i=1}^{m} \sum_{n_1^i[1] \in N_1^i[1], n_2^i[2] \in N_1^i[2]} \Delta_{CSC}(n_1^i[1], n_2^i[2]),$$

where $n_1^i[j] = (n_1, n_2, \ldots, n_i)[j]$ denotes a root node path of length i in tree τ_j. This kernel will therefore incorporate the similarity of common sub-trees.

4.2 Using Tree-Kernels for SQL Parse-Trees

As mentioned in the beginning, the use of tree-kernels in intrusion detection has been proven to provide a syntax-oriented analysis in protocols such as FTP or HTTP [11,12]. To exploit the benefit of syntax-level awareness in SQL query-analysis, we derive the distance measure induced by a tree-kernel function to directly measure the similarity of SQL queries by means of their parse-trees.

For a kernel k and examples x_1, x_2, such a distance can be obtained by

$$d(x_1, x_2) = \sqrt{k(x_1, x_2) - 2k(x_1, x_2) + k(x_1, x_2)}. \tag{2}$$

Using a tree-kernel we can therefore use this kernel to directly compute the distance of two SQL parse-trees using (2).

5 Experimental Analysis and Results

For an evaluation of the different modelling approaches we collected data of the popular Typo3 content management system. This application heavily depends on the use of SQL for various tasks beyond page content storage, such as session-persistence, user-management and even page-caching.

We created a set of distinct queries and added synthetic attacks, which closely reflect modifications that would follow from SQL injections, by inserting typical injection vectors such as `OR 'a' = 'a'` or the like into legal statements. The intention was to observe whether, using different models, the SVM is to distinguish between legal and malicious statements even though the latter were only marginally different. We created two sets with different ratios of normal to malicious queries, one with 200:15, the other with 1000:15 queries.

5.1 Importance of Context

A central question in our work is the importance of contextual information when analyzing SQL queries. We therefore analyzed approaches such as n-grams, term-vector and the SQL vectorization described in section 3.2. In this experiment we did not mean to train a detector, but wanted to explore the expressiveness of the different models and determined the detection rate (TPR) and the false-positive rate (FPR) of the different modelling approaches. As learning algorithm we used an SVM approach within a 10-fold cross-validation.

Table 1. Separation capabilities of the different models based on a 10-fold cross-validation

Model	Ratio 200:15			Ratio 1000:15		
	TPR	FPR	time	TPR	FPR	time
3-gram	0.6667	0.000	71 s	0.6667	0.002	643 s
4-gram	0.3333	0.000	149 s	0.7333	0.002	1055 s
Term vectors	0.6667	0.005	2 s	0.7333	0.002	283 s
SQL vectors	0.8667	0.000	16 s	0.8667	0.001	67 s

As you can see from table 1, the use of context information results in performance gains especially with respect to the detection rate (TPR) and the fraction of false positives (FPR). This supports our thesis on the importance of the context when analyzing SQL queries. It is worth noting, that the variance in TPR/FPR within the 10-fold cross validation proved to be much smaller for the context-sensitive methods. Additionally, the training time using term- or sql-vectorization decreased due to the smaller number of (irrelevant) attributes. The times in table 1 refer to the complete parameter-optimization and 10-fold cross-validation process.

5.2 Query Analysis Using Tree-Kernels

Using the tree-kernel similarity we are interested in analyzing an application's structure by means of different sets of similar statements used. Therefore we used the kernel similarity within an interval self-organizing map (ISOM) to create a visualization of an application's statements. In figure 2 you see the ISOM of 200 regular queries taken from Typo-3 (dots), supplemented by 15 modified "malicious" modifications (squares).

As can be seen in figure 2 the kernel does consolidate similar queries into clusters, an inspection of the clustered regions revealed very reasonable groups, such as "all page-content queries", "all session update queries" and so on. The heaps of dots turned out to be of a very similar structure, only differing in terminal symbols. Further adding edges to the ISOM showed, that the modified queries are consolidated very late, showing that they are highly dissimilar.

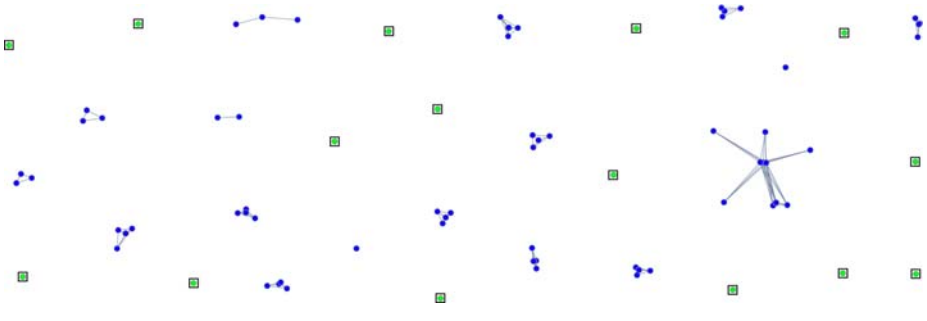

Fig. 2. ISOM of 215 Typo-3 queries (200 legal, 15 anomalous), created by the CSC tree-kernel ($\lambda = 0.6, m = 10$)

Fig. 3. Intra-Cluster ISOM of a cluster consisting of 46 legal queries and one single anomalous modification, which resulted from adding SQL injection elements

5.3 Intra-cluster ISOMs

As the ISOM experiments proved to be useful to get a feeling for the similarity measure, we employed a KMedoids clustering algorithm based on the tree-kernel distance and inspected the clusters by creating ISOMs of each cluster separately. Figure 3 shows the ISOM of a cluster containing "attacks" which are similar to the majority of the queries, but differ by injected SQL fragments.

Within this cluster the anomalous queries is the one most dissimilar from all other, resulting in isolation. The queries in the left-hand group are related to selecting language-specific content from the database, whereas the group on the right contains queries selecting page-content related to a user-id `UID`. The anomalous query contains an additional `OR UID > 0`, neutralizing the UID check.

This yields a two-way analysis which uses a clustering approach to first group the different kinds of statements and then uses an intra-cluster outlier detection for the detection of malicious queries.

6 Conclusions and Future Work

We presented two approaches for a context sensitive modelling/fingerprinting of SQL queries by use of generic models. Using tree-kernels for analyzing SQL statements brings together the results of natural language processing with a highly structured query language. The results confirm the benefit of incorporation of syntax information of previous works [11,12] in the domain of SQL queries.

The consideration of the SQL structures shows performance gains in both performance and speed, the later due to the fewer but far more meaningful features. Compared to previous approaches the tree-kernels allow for a similarity measure on SQL statements providing flexible generalization capabilities.

However, a drawback in the use of tree-kernels is their computational overhead. Given a set of 1015 queries, the computation of the kernel matrix took about 210 seconds. Use of hierarchical models, such as hierarchical clustering, may lower the impact of this performance decrease for future detection models.

Here, our first Clustering and ISOM experiments in 5 show the usefulness of tree-kernels as a similarity measure in order to visualize SQL queries in applications. However, the tree-kernel approach still offers a lot of optimization possibilities and needs further investigation. In future works we therefore plan on using inter-cluster outlier detection to create hierarchical anomaly detection models based on tree-kernels over SQL parse-trees.

References

1. Open Web Application Security Project. The Top list of most severe web application vulnerabilities (2004)
2. Roesch, M.: Snort: Lightweight intrusion detection for networks. In: Proc. of LISA, pp. 229–238. USENIX (1999)
3. Ristic, I.: ModSecurity - A Filter-Module for the Apache Webserver (1998)
4. Kruegel, C., Vigna, G.: Anomaly Detection of Web-based Attacks. In: Proc. of ACM CCS, pp. 251–261. ACM Press, New York (2003)
5. Kruegel, C., Vigna, G., Robertson, W.: A Multi-model Approach to the Detection of Web-based Attacks. Computer Networks 48(5), 717–738 (2005)
6. Valeur, F., Vigna, G., Kruegel, C., Kirda, E.: An Anomaly-driven Reverse Proxy for Web Applications. In: Proc. of ACM SAC (2006)
7. Bockermann, C., Mierswa, I., Morik, K.: On the automated creation of understandable positive security models for web applications. In: Proc. of IEEE PerCom, pp. 554–559. IEEE Computer Society, Los Alamitos (2008)
8. Shezaf, O., Grossman, J.: Web Hacking Incident Database (2008)
9. Lee, S.-Y., Low, W.L., Wong, P.Y.: Learning fingerprints for a database intrusion detection system. In: Gollmann, D., Karjoth, G., Waidner, M. (eds.) ESORICS 2002. LNCS, vol. 2502, pp. 264–280. Springer, Heidelberg (2002)
10. Buehrer, G., Weide, B.W., Sivilotti, P.A.G.: Using parse tree validation to prevent sql injection attacks. In: Proc. of SEM, pp. 106–113. ACM, New York (2005)
11. Gerstenberger, R.: Anomaliebasierte Angriffserkennung im FTP-Protokoll. Master's thesis, University of Potsdam, Germany (2008)
12. Düssel, P., Gehl, C., Laskov, P., Rieck, K.: Incorporation of application layer protocol syntax into anomaly detection. In: Sekar, R., Pujari, A.K. (eds.) ICISS 2008. LNCS, vol. 5352, pp. 188–202. Springer, Heidelberg (2008)
13. Hu, Y., Panda, B.: A data mining approach for database intrusion detection. In: Proc. of ACM SAC, pp. 711–716. ACM, New York (2004)
14. Srivastava, A., Sural, S., Majumdar, A.K.: Database intrusion detection using weighted sequence mining. JCP 1(4), 8–17 (2006)
15. Roichman, A., Gudes, E.: DIWeDa - detecting intrusions in web databases. In: Atluri, V. (ed.) DAS 2008. LNCS, vol. 5094, pp. 313–329. Springer, Heidelberg (2008)
16. Lewis, D.D.: Naive (bayes) at forty: The independence assumption in information retrieval. In: Nédellec, C., Rouveirol, C. (eds.) ECML 1998. LNCS, vol. 1398, pp. 4–15. Springer, Heidelberg (1998)
17. Rieck, K., Holz, T., Willems, C., Düssel, P., Laskov, P.: Learning and classification of malware behavior. In: Zamboni, D. (ed.) DIMVA 2008. LNCS, vol. 5137, pp. 108–125. Springer, Heidelberg (2008)
18. Haussler, D.: Convolution kernels on discrete structures. Technical report, Dept. of Computer Science, UC Santa Cruz (1999)
19. Collins, M., Duffy, N.: Convolution kernels for natural language. In: Advances in Neural Information Processing Systems 14, pp. 625–632. MIT Press, Cambridge (2001)
20. Zhou, G.D., Zhang, M., Ji, D.H., Zhu, Q.M.: Tree kernel-based relation extraction with context-sensitive structured parse tree information. In: Proc. of Joint Conf. on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, pp. 728–736. Assoc. for Computer Linguistics (2007)