Syntactic Structural Kernels for Natural Language Interfaces to Databases

Alessandra Giordani and Alessandro Moschitti

Department of Computer Science and Engineering
University of Trento
Via Sommarive 14, 38100 POVO (TN) - Italy
{agiordani,moschitti}@disi.unitn.it

Abstract. A core problem in data mining is to retrieve data in a easy and human friendly way. Automatically translating natural language questions into SQL queries would allow for the design of effective and useful database systems from a user viewpoint. Interesting previous work has been focused on the use of machine learning algorithms for automatically mapping natural language (NL) questions to SQL queries.

In this paper, we present many structural kernels and their combinations for inducing the relational semantics between pairs of NL questions and SQL queries. We measure the effectiveness of such kernels by using them in Support Vector Machines to select the queries that correctly answer to NL questions. Experimental results on two different datasets show that our approach is viable and that syntactic information under the form of pairs of syntactic tree fragments (from queries and questions) plays a major role in deriving the relational semantics between the two languages.

Keywords: Natural Language Processing; Kernel Methods; Support Vector Machines.

1 Introduction

In the last decade many natural language interfaces to database (NLIDBs) have been proposed to translate the human intent into machine-readable instructions [1,2,3,4,5,6,7,8]. Despite this, little progress has been made in developing an interface that can be used by any untrained user without manual annotation and intervention. The problem of automatically translating natural language (NL) questions into SQL queries is an interesting and appealing research in data mining. For example, solving this problem would suggest the role of syntax for mapping NLs to artificial languages; this would have a direct impact in the field of information systems. Unfortunately computational linguistics and artificial intelligence research [9] has shown that such mapping problem cannot be addressed with a deep semantic approach, thus a concrete solution should rely on shallow and statistical methods.

W. Buntine et al. (Eds.): ECML PKDD 2009, Part I, LNAI 5781, pp. 391–406, 2009.

In this paper, we propose a set of structural kernels, e.g. Sequence and Tree Kernels [10,11,12,13], and Support Vector Machines (SVMs) to map NL into SQL.

First, starting from a set of correct pairs of questions and the related SQL queries, available for our target DBs, we design an algorithm to produce incorrect pairs and additional correct pairs, i.e. negative and positive examples, respectively.

Second, we model a representation of the above question/query pairs in terms of syntactic structures, i.e. we build pairs of syntactic parse trees automatically derived by off-the-shelf natural language parsers and the straightforward application of SQL grammar.

Third, we train SVMs with the above data, where the structural representation of the pairs is encoded by means of different types of kernels, i.e. linear, polynomial, string and tree kernels and their combinations. This allows us to automatically exploit the associative patterns between NL and SQL syntax to detect correct and incorrect pairs from an operational semantics viewpoint.

Finally, given a new question and the set of available queries (i.e. the repository of queries asked to the target DB), we produce the set of pairs containing such question and then we use SVMs to rank pairs in terms of *correctness*. We select the top scored pair as the query that answers the given question.

The new contributions with respect to our previous research on Natural Language Interface to Databases [14] are the following:

- We propose and study sequence kernels to provide a pair representation that is shallower than the one based on deep syntactic parsing. Although, they prove to not be essential for the design of the most accurate model, their comparison with polynomial kernels gives some indications on the role of feature pair spaces.
- We experimented with a large number of kernels showing that, in contrast with our previous findings, complex kernels relevantly improve the simple space of term (word) pairs.
- We applied our semi-automatic algorithm to a second dataset of correct question and query pairs, namely RestQueries [7], to design a new dataset for classification. We made it available[1] along with the one derived by GeoQueries.

The use of RestQueries allowed us to (1) assess the high effectiveness of product kernels and the feature pair spaces, which, even in their simple form (e.g. word pairs), highly improve the traditional linear kernel; (2) show that syntactic information is very important since it improves the best model by about 10 absolute percent points; and (3) find out that complex kernels such as the polynomial expansion of pairs of tree fragments and bigrams can produce the highest results.

In the remainder, Section 2 shows our proposed algorithm to generate a training set of question and query pairs used by the kernel-based classifier as described

[1] `http://disi.unitn.it/~moschitt/corpora.htm`

in Section 3. Section 4 discusses the experimental setup and results, Section 5 reviews some state of the art NLIDBs, to which we compare, and finally, Section 6 draws conclusions.

2 Dataset Generation

The goal of this research is the development of a NLIDB that maps NL questions into SQL queries based on a machine learning approach; consequently, we need to have training data, i.e. a set of positive and negative examples. In practical cases, we can assume to have a set of positive examples consisting of correct question/query pairs[2], i.e. such that the execution of the query retrieves a correct answer for the question. Assuming the availability of negative examples is a more strong assumption since providing the correct query for an user information need is a more natural task than providing the incorrect solution.

Therefore, to create negative examples, we may use the initial set of questions and queries in the correct pairs and randomly pair them. Unfortunately, this may generate false negatives since different questions may have more than one answer (and vice-versa), thus a manual verification of such pairs is required. To reduce such costly manual intervention, we can exploit the semantic equivalency between the pairs' members and its transitivity closure to pair questions to their correct queries. This allow us to extend the set of positive examples.

The semantic equivalency can be calculated by means of a clustering algorithm, which groups questions that represent the same information need with queries that correctly retrieve it. An approach to effectively detect such equivalence is the generalization of questions and queries, e.g. *What are some good restaurants in Berkeley* becomes *What are some good restaurants in a city*.

In the next sections, we describe our approach to automatically generate the target dataset. The main steps are: (1) generalize question and query instances, (2) cluster the generalized pairs, (3) generate all the true positives by pairing questions and queries belonging to the same clusters, and (4) annotate as true negatives all remaining pairings between questions and queries of distinct clusters. This also requires a limited manual intervention.

2.1 Generalizing Pairs

The aim of pair generalization is to make the detection of semantically equivalent questions and queries easy. Our approach consists in considering questions or queries having similar structures instantiated by the same semantic concepts. The latter are generalizations of important domain terms occurring both in the question and in the related query. For example, terms like *Berkeley*, *San Francisco*, etc., are substituted with the concept *city*. Note that: (a) we can identify

[2] For example, correct pairs may be defined when databases are designed and validated. Also, we may ask the DB operator to collect the set of queries that she/he designed in response of typical specific (questions) asked by DB users.

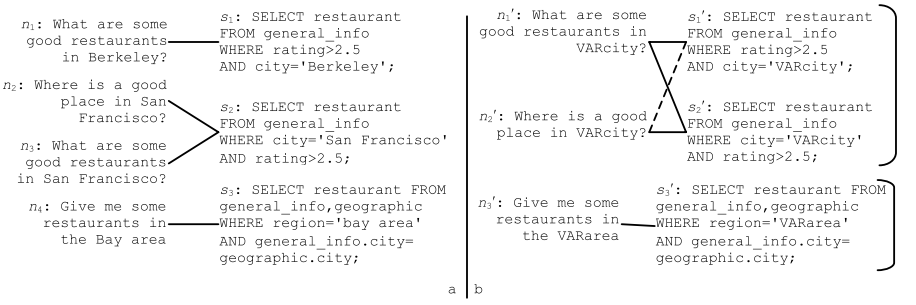

Fig. 1. Example of the initial corpus (*A, on the left*) and the generalized version (*B, on the right*). The latter is divided in two clusters (*identified by the two brackets*).

concepts by extracting the column names (in the database) that naturally store domain terms; and (b) concepts are expressed in the *WHERE* condition of the given SQL queries.

An example of the generalization phase is shown in Figure 1, which reports questions and queries of a restaurant domain. More in detail, on the left there is a set of four pairs containing four distinct questions and their three related queries (connected by lines) whereas on the right four generalized pairs are shown. In the question and query pair $\langle n_1, s_1 \rangle$, where n_1: "What are some good restaurants in Berkeley?" and s_1: `SELECT restaurant FROM general_info WHERE rating>2.5 AND city='Berkeley'`, since *Berkeley* is associated with the column *city*, its occurrences in n_1 and s_1 are substituted with the concept/variable *VARcity*.

We note that, after substituting instances with variables, both n_1 and n_3 are generalized into n_1', which can then be paired with two distinct SQL queries, i.e. s_1' and s_2'. This is correct since there can be more SQL queries that correctly retrieve an answer to an NL question. We can define them to be *semantically equivalent*, i.e. $s_1' \equiv s_2'$. Conversely, there can be many NL questions that map to the same query, e.g. $n_2 \equiv n_3$[3].

It is worth noting that with the generalization process, we introduce redundancy that we eliminate by removing duplicated questions and queries. Thus, the output dataset is usually smaller than the initial one. However the number of training examples will be larger, not only because of the introduction of negatives but also due to the automatic discovering of new positives.

2.2 Pair Clustering and Final Dataset Annotation

Once the pairs have been generalized, we cluster them according to their semantic equivalence so that we can automatically derive new positive examples by swapping their members. We define semantically equivalent pairs those correct pairs with (a) equivalent NL questions, i.e. whose generalized version is the same or (b) equivalent SQL queries. Given that two equivalent queries must retrieve

[3] It is worth noting that in this equivalence is true in this domain but in other domains can be false, since *good places* not necessarily refers to restaurants.

the same result set, we can automatically test their equivalence by simply executing them. Unfortunately, this is just a necessary condition (e.g. two different queries can have the same answer) therefore we manually evaluate new pairings obtained applying this condition.

Note that automatically detecting semantic equivalence of natural language questions with perfect accuracy is a hard task, so we consider as semantically equivalent either identical questions (after generalization) or those associated with semantic equivalent queries. We also apply transitivity closure to both members of pairs to extend the set of equivalent pairs.

For example, in Figure 1.b s'_1 and s'_2 retrieve the same results so we verify that they are semantically equivalent queries and we assign them to the same cluster (CL1), i.e. information need about good restaurants in a city (with a rating larger than 2.5 stars). Alternatively, we can also consider that n'_1 and n'_2 are both paired with s'_2 to derive that they are equivalent, avoiding the human intervention. Concerning s'_3, it retrieves a result set different form the previous one so we can automatically assign it to a different cluster (CL2), i.e. involving questions about restaurants in a region. Note that, once n'_2 is shown to be semantically equivalent to n'_1, we can pair them with s'_1 to create the new pair (indicated by the dashed line) $\langle n'_2, s'_1 \rangle$. Indeed the negative example set is $\langle n'_3, s'_1 \rangle, \langle n'_3, s'_2 \rangle, \langle n'_1, s'_3 \rangle, \langle n'_2, s'_3 \rangle$.

3 Kernel Methods for Question/Query Representation

Kernel Methods refer to a large class of learning algorithms based on inner product vector spaces, among which Support Vector Machines (SVMs) are one of the most well-known algorithms. The main idea is that the parameter model vector $\boldsymbol{w}$ generated by SVMs (or by other kernel-based machines) can be rewritten as $\sum_{i=1..l} y_i \alpha_i \boldsymbol{x}_i$, where y_i is equal to 1 for positive and -1 for negative examples, $\alpha_i \in \Re$ with $\alpha_i \geq 0, \forall i \in \{1, .., l\}$ $\boldsymbol{x}_i$ are the training instances. Therefore we can express the classification function as $\text{Sgn}(\sum_{i=1..l} y_i \alpha_i \boldsymbol{x}_i \cdot \boldsymbol{x} + b) = \text{Sgn}(\sum_{i=1..l} y_i \alpha_i \phi(o_i) \cdot \phi(o) + b)$, where $\boldsymbol{x}$ is a classifying object, b is a threshold and the product $K(o_i, o) = \langle \phi(o_i) \cdot \phi(o) \rangle$ is the kernel function associated with the mapping ϕ.

Note that it is not necessary to apply the mapping ϕ, we can use $K(o_i, o)$ directly. This allows, under the Mercer's conditions [15] for defining abstract functions which generate implicit feature spaces. The latter allow for an easier feature extraction and the use of huge feature spaces (possibly infinite), where the scalar product (i.e. $K(\cdot, \cdot)$) is implicitly evaluated.

In the following section, we first propose a structural representation of the question and query pairs, then we report the two more adequate kernels for syntactic structure representation, i.e. the Syntactic Tree Kernel (STK) [11], which computes the number of syntactic tree fragments and the Extended Syntactic Tree Kernel (STK_e) [16], which includes leaves in STK. In the last subsection we show how to engineer new kernels from them.

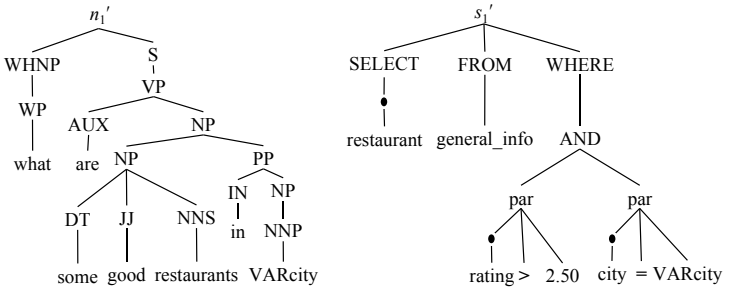

Fig. 2. Question/Query Syntactic trees

3.1 Representing Question and Queries Pairs

In Data Mining and Information Retrieval the so-called bag-of-words (BOW) has been shown to be effective to represent textual documents, e.g. [17,18]. However, in case of questions and queries we deal with small textual objects in which the semantic content is expressed by means of few words and poorly reliable probability distributions. In these conditions the use of syntactic representation improves BOW and should be always used, [13,19,20,21,22,23].

Therefore, in addition to BOW, we represent questions and queries using their syntactic trees[4]. As shown in Figure 2 for questions (a) we use the Charniak's syntactic parser [25] while for queries (b) we implemented an ad-hoc SQL parser. The latter builds a SQL parse tree for each query following its syntactic derivation according to MySQL grammar. The grammar has been slightly modified to accommodate the usage of the symbol • for the production of *items* in the SELECT clause and in WHERE conditions. In such an SQL tree, the internal nodes are only the SQL keywords of the query plus the special symbol • whereas the leaves are names of tables and columns of the database, category variables or operators. Note that, although we eliminated comma and dot from the grammar, it is still possible to obtain the original SQL query, by just performing a preorder traversal of the tree.

To represent the above structures in a learning algorithm we use tree kernels described in the following section.

3.2 Tree Kernels

The main underlying idea of tree kernels is to compute the number of common substructures between two trees T_1 and T_2 without explicitly considering the whole fragment space. Let $\mathcal{F} = \{f_1, f_2, \ldots, f_{|\mathcal{F}|}\}$ be the set of tree fragments and $\chi_i(n)$ an indicator function equal to 1 if the target f_i is rooted at node n and equal to 0 otherwise. A tree kernel function over T_1 and T_2 is defined as $TK(T_1, T_2) = \sum_{n_1 \in N_{T_1}} \sum_{n_2 \in N_{T_2}} \Delta(n_1, n_2)$, where N_{T_1} and N_{T_2} are the sets of

[4] Early work on the use of syntax for text categorization were based on part-of-speech tags, e.g. [24].

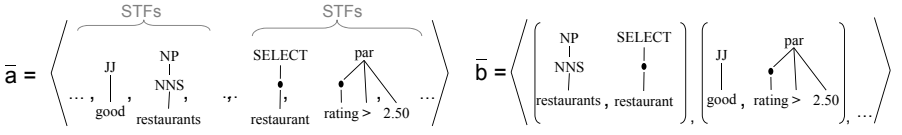

Fig. 3. Feature spaces for the tree pair in Figure 2 a) joint space STK+STK b) Cartesian product STK×STK

nodes in T_1 and T_2, respectively, and $\Delta(n_1, n_2) = \sum_{i=1}^{|\mathcal{F}|} \chi_i(n_1)\chi_i(n_2)$. The Δ function is equal to the number of common fragments rooted in nodes n_1 and n_2, and thus, depends on the fragment type. We report its algorithm for the evaluation of the number of syntactic tree fragments (STFs).

A syntactic tree fragment (STF) is a set of nodes and edges from the original tree which is still a tree and with the constraint that any node must have all or none of its children. This is equivalent to state that the production rules contained in the STF cannot be partial. To compute the number of common STFs rooted in n_1 and n_2, the STK uses the following Δ function [11]:

1. if the productions at n_1 and n_2 are different then $\Delta(n_1, n_2) = 0$;
2. if the productions at n_1 and n_2 are the same, and n_1 and n_2 have only leaf children (i.e. they are pre-terminal symbols) then $\Delta(n_1, n_2) = \lambda$;
3. if the productions at n_1 and n_2 are the same, and n_1 and n_2 are not pre-terminals then
$\Delta(n_1, n_2) = \lambda \prod_{j=1}^{l(n_1)} (1 + \Delta(c_{n_1}(j), c_{n_2}(j)))$,

where $l(n_1)$ is the number of children of n_1, $c_n(j)$ is the j-th child of the node n and λ is a decay factor penalizing larger structures.

Figure 3.a shows some STFs of the NL and SQL trees in Figure 2. STFs satisfy the constraint that grammatical rules cannot be broken. For example, *[VP [AUX NP]]* is a STF, which has two non-terminal symbols, *AUX* and *NP*, as leaves whereas *[VP [AUX]]* is not a STF.

STK does not include individual nodes as features. As shown in [16] using its extension (STK_e) we can include at least the leaves, (which in constituency trees correspond to words) by simply inserting the following step 0 in the algorithm above [16]:

0. if n_1 and n_2 are leaf nodes and their labels are identical then $\Delta(n_1, n_2) = \lambda$;

3.3 String Kernels

The String Kernels that we consider count the number of substrings containing gaps (i.e. some of the symbols of the original string are skipped) shared by two sequences. We adopted the efficient algorithm described in [15,10,26,27]. Characters in the sequences can be substituted with any set of symbols. In our study we preferred to use words obtaining word sequences. For example, given the query: `Select restaurant from general_info` sample substrings, extracted by the Sequence Kernel (SK), are: `Select restaurant`, `Select from general_info`, `Select general_info`, `Select from`, etc. It is worth noting that: (i) longer

subsequences receive lower weights, (ii) some words can be omitted, i.e. gaps and (iii) gaps determine a weight since an exponential decay factor is applied, where the exponent is the number of words and gaps between the first and last words.

3.4 Kernel Engineering for Pair Representation

Kernel engineering [28,29,30] can be carried out by combining basic kernels with additive or multiplicative operators or by designing specific data objects, e.g. the tree representation for the SQL syntax, to which standard kernels are applied. Since our data is a set of pairs, we need to represent the members of a pair and their interdependencies. For this purpose, given two kernel functions, $k_1(.,.)$ and $k_2(.,.)$, and two pairs, $p_1 = \langle n_1,s_1\rangle$ and $p_2 = \langle n_2,s_2\rangle$, a first approximation is given by summing the kernels applied to the components: $K(p_1, p_2) = k_1(n_1, n_2) + k_2(s_1, s_2)$. This kernel will produce the union of the feature spaces of questions and queries. For example, the explicit vector representation of the space of STK of the pair in Figure 2 is shown in Figure 3.a. The Syntactic Tree Fragments of the question will be in the same space of the Syntactic Tree Fragments of the query.

In theory a more effective kernel is the product $k(n_1, n_2) \times k(s_1, s_2)$ since it generates pairs of fragments as features, where the overall space is the Cartesian product of the used kernel spaces. For example Figure 3.b shows pairs of STF fragments, which are essential to capture the relational semantics between the syntactic tree subparts of the two languages [31]. In particular, the second fragment pair of the figure may suggest that the adjective phrase *good* expresses similar semantics of the syntactic construct `WHERE rating>2.5`. In other words, the above pair feature suggests that the whole query may be a correct translation of the given question.

As additional feature and kernel engineering, we also exploit the ability of the polynomial kernel to add feature conjunctions. By simply applying the function $(1 + K(p_1, p_2))^d$, we can generate conjunction up to d features. Thus, we can obtain tree fragment conjunctions and conjunctions of pairs of tree fragments.

The next section will show the results using different kernel combination for pair representation.

4 The Experiments

We ran several experiments to evaluate the accuracy of our approach in automatically selecting correct SQL queries for NL questions, where the selection of the correct query is modeled as a ranking problem. The ranker is constituted by SVMs and by the kernels described in Section 3. To show the generality of our approach we created two different datasets by applying our algorithm described in Section 2 to two different corpora.

4.1 Setup

We address the problem of finding a query whose result answers to a question according to the following ranking problem. Given a question $n \in \mathcal{N}$ and the complete set of the available queries $\mathcal{S}$, we classify the set of all possible pairs $P(n) = \{\langle n, s\rangle : s \in \mathcal{S}\}$. Then we use the classification score to rank the element of $P(n)$ and select the pair with the highest score[5].

To learn the classifier we used SVM-Light-TK[6], which extends the SVM-Light optimizer [18] with tree kernels. i.e. Syntactic Tree Kernel (STK) and its extension (STK_e) as described in Section 3. We implemented the String Kernel [10] (word sequence kernel [26]) and modeled many different combinations described in the next section. We used the default parameters, i.e. the cost and trade-off parameters = 1 (for normalized kernels) and $\lambda = 0.4$ (see Sec. 3.2).

To generate our datasets we applied our algorithm described in Section 2 to GeoQueries250 and RestQueries corpora[7].

The first corpus is about geography questions. After the generalization process the initial 250 pairs of questions/queries were reduced to 155 pairs containing 154 NL questions and 79 SQL queries. We found 76 clusters, from which we generated 165 positive and 12,001 negative examples for a total of 154 × 79 pairs. Such dataset will be referred to as Geo.

The second dataset regards questions about restaurants. The initial 250 pairs were generalized by 197 pairs involving 126 NL questions and 77 SQL queries. We clustered these pairs in only 26 groups, which lead to 852 positive examples and 9,702 negatives. Such dataset will be referred to as Rest.

To evaluate the results of our mapping models, we applied standard 10-fold cross validation and measure the average accuracy and the Std Dev. of selecting the correct query for each question.

4.2 Results on Geo Dataset

We tested several models for ranking based on different kernel combinations whose results are reported on Table 1 and Table 2. The first column of Table 1 lists kernel combination by means of product and sum between pairs of basic kernels used for the question and the query, respectively. The latter column shows the average accuracy (over 10 folds) ± Std. Dev.

More in detail, our basic kernels are: (1) linear kernel (LIN) built on the bag-of-stems (BOS) of the questions or of the query; (2) a polynomial kernel of degree 3 on the above BOSs (POLY); (3) the Syntactic Tree Kernel (STK) on the parse tree of the question or the query and (4) STK extended with leaf features (STK_e). Note that we can also sum or multiply different kernels, e.g. POLY×STK.

[5] More effective approaches have been proposed [11,32].

[6] `http://disi.unitn.it/~moschitt/Tree-Kernel.htm`

[7] Questions in both corpora were originally collected from a web-based interface and manually translated into logical formulas in Prolog by Mooney's group [7]. Popescu et al. [2] manually converted them into SQL. Thanks to our clustering algorithm we discovered and fixed many errors and inconsistencies in SQL queries.

Table 1. Kernel combination accuracies (± Std. Dev) for GEO dataset

Combination	Accuracy
LIN + LIN	**57.3**±10.4
LIN × LIN	**70.7**±12.0
POLY × POLY	71.9±11.5
STK × STK	70.3±9.3
$STK_e \times STK_e$	70.1±10.9
LIN × STK	74.6±9.6
LIN × STK_e	**75.6**±13.1
POLY × STK	73.8±9.5
POLY × STK_e	73.5±10.4
STK × LIN	64.7±11.5
STK_e × LIN	68.3±9.6
STK × POLY	65.4±10.9
STK_e × POLY	68.3±9.6

Table 2. Advanced kernel combination accuracies (± Std. Dev) for GEO dataset

Advanced Kernels	Accuracy
STK^2+$POLY^2$	72.7±9.7
STK_e^2+$POLY^2$	73.2±11.4
$(1+LIN^2)^2$	**73.6**±9.4
$(1+POLY^2)^2$	73.2±10.9
$(1+STK^2)^2$	69.4±10.0
$(1+STK_e^2)^2$	70.0±12.2
$(1+LIN^2)^2$+STK^2	75.6±8.3
$(1+POLY^2)^2$+STK^2	72.6±10.5
$(1+LIN^2)^2$+LIN×STK	**75.9**±9.6
$(1+POLY^2)^2$+POLY×STK	73.2±10.9
POLY×STK+STK^2+$POLY^2$	73.9±11.5
POLY×STK_e+STK_e^2+$POLY^2$	75.3±11.5
LIN×STK+STK^2+LIN^2	74.5±9.1
LIN×STK_e+STK_e^2+LIN^2	74.9±11.8

An examination of the reported figures suggests that: first, the basic traditional model based on linear kernel and BOS, i.e. LIN + LIN, provides an accuracy of only 57.3%, which is greatly improved by LIN×LIN=LIN^2, i.e. by 13.5 points [8]. The explanation is that the sum cannot express the relational feature pairs coming from questions and queries, thus LIN cannot capture the underlying shared semantics between them. It should be noted that only kernel methods allow for an efficient and easy design of LIN^2; the traditional approach would have required to build the Cartesian product of the question BOS by query BOS. This can be very large, e.g. 10K features for both spaces lead to a pair space of 100M features.

Second, the feature pair space is essential since the accuracy of all kernels implementing the union spaces of question and query representations[9] is much lower than the baseline model for feature pairs, i.e. LIN^2.

Third, if we include conjunctions in the BOS representation by using POLY, we improve the LIN model, i.e. 71.9% vs. 70.8%. $POLY^2$ is also better than STK^2 since it includes individual term/word bigrams that are not included by STK.

Next, the lower accuracy provided by STK^2 and STK_e^2 suggests that syntactic models can improve BOS although too many (possibly incorrect) syntactic features (generated by the syntactic parser) make the model unstable. This consideration leads us to experiment with the model LIN × STK and LIN × STK_e, which combine words of the questions with syntactic constructs of SQL queries.

[8] Although the Std. Dev. associated with the model accuracy is high, the one associated with the distribution of difference between the model accuracy is much lower, i.e. 5%.

[9] Given the limited space, we could not report the results of these poorly accurate kernels.

Table 3. Kernel combination accuracies (± Std. Dev) for REST dataset

Combination	Accuracy
LIN + LIN	**20.9**±11.9
LIN × LIN	**37.1**±16.2
POLY × POLY	74.5±14.0
STK × STK	71.8±10.8
STK_e × STK_e	62.5±11.6
LIN × STK	**79.1**±11.5
LIN × STK_e	77.2±12.8
POLY × STK	**82.3**±11.8
POLY × STK_e	78.0±12.2
STK × LIN	38.3±13.4
STK × POLY	45.5±11.8
SK_3 × SK_3	67.4±11.1
SK_3 × STK	**81.7**±13.3
SK_3 × STK_e	78.5±11.5

Table 4. Advanced kernel combination accuracies (± Std. Dev) for REST dataset

Advanced Kernels	Accuracy
STK^2+$POLY^2$	78.6±11.9
STK_e^2+$POLY^2$	73.3±10.2
$(1+LIN^2)^2$	52.5±10.2
$(1+POLY^2)^2$	74.5±14.0
$(1+STK^2)^2$	74.5±14.0
$(1+STK_e^2)^2$	62.5±11.6
$(1+SK_3^2)^2$	69.8±10.0
$(1+POLY \times STK)^2$	**84.7**±11.5
$(1+POLY^2)^2+STK^2$	78.7±12.1
$(1+POLY^2)^2+POLY \times STK$	78.1±13.8
$POLY \times STK+STK^2+POLY^2$	78.6±11.9

They produce statistically significant higher results (at 90% of confidence), i.e. 74.6% and 75.6%. This suggests that the syntactic parse tree of the SQL query is very reliable (indeed, it is obtained with 100% of accuracy) while the natural language parse tree, although accurate, introduces noise that degrades the overall feature representation. As a consequence it is more effective to use words only in the representation of the first member of the pairs.

This is also shown by the last four lines of Table 1, showing the low accuracies obtained when relying on NL syntactic parse trees and SQL BOSs. Thus the only viable possibility to improve LIN × STK_e was to use the polynomial kernel in the combination POLY × STK_e. The slightly lower outcome shows that POLY is equivalent to LIN.

Moreover, we experimented with very advanced kernels built on top of feature pair spaces as shown in Table2. For example, we sum different pair spaces, STK_e^2 and $POLY^2$, and we apply the polynomial kernel on top of pair spaces by creating conjunctions, over feature pairs. This operation tends to increase too much the cardinality of the space and makes it ineffective. However, using the simplest initial space, i.e. LIN, to build pair conjunctions, i.e. $(1+LIN^2)^2$, we obtain a very interesting and high result, i.e. 73.6%. Using the joint space of the kernel above and kernel products, we can still improve our models, e.g. $(1+LIN^2)^2+LIN \times STK$.

This suggests that kernel methods have the potentiality to describe relational problems using simple building blocks although new theory describing the degradation of kernels when the space is too complex is required.

Finally, to study the stability of our complex kernels, we compared the learning curve of the baseline model, i.e. LIN+LIN, with the those of the best models, i.e. $LIN \times STK_e$ and $STK^2+(1+LIN^2)^2$. Figure 4 shows that surprisingly, complex

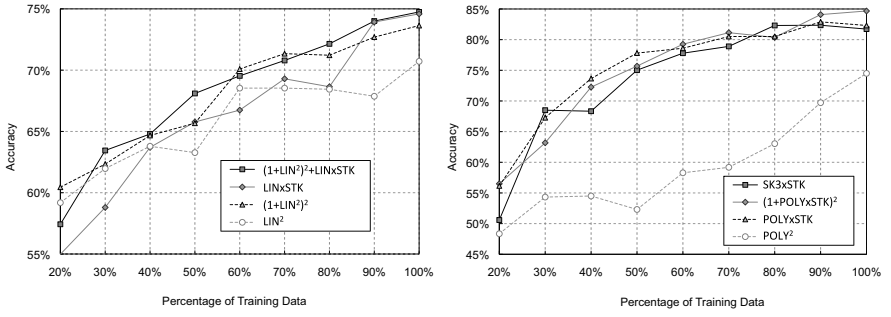

Fig. 4. Learning curves for GeoQueries and RestQueries corpora

kernels are not only more accurate but also more stable, i.e. their accuracy increases smoothly according to the availability of training data.

4.3 Results on Rest Dataset

The previous results, although interesting, show that syntactic information plays a minor role and that complex kernels do not significantly improve our question translator (similar findings were derived in the preliminary experiments in [14]).

To verify such hypothesis, we experimented with the second dataset whose results, reported in tables Table 3 and Table 4, provide more interesting data.

First, the baseline model, i.e. LIN + LIN, produces very low accuracy, i.e. only 20.9%, which is highly improved by LIN^2, also showing a very low result, i.e. 37.1%, although.

Second, surprisingly, including conjunctions in the BOS representation, i.e. by using POLY^2, the accuracy of LIN^2 doubled (74.5%). Moreover, if we combine POLY with syntactic SQL subtrees, i.e. POLY×STK, we obtain another relevant improvement, i.e. 82.3%. This confirms that it is better to use stems in the representation of the first member of the pairs and syntactic parse trees in the second member.

Third, given that n-gram based text representation technique has shown to outperform bag-of-words approaches [10], we experimented with String Kernel (SK). The results confirm that sequences of three words (SK_3^2) better represent questions than BOS (see SK_3^2 vs. LIN^2). Nevertheless, the conjunctions of POLY are more effective.

Next, we experimented with advanced kernel combinations. The results, listed in table Table 4, show that the advanced polynomial kernel combination $(1+\text{POLY} \times \text{STK})^2$ outperforms[10] the best kernel combination, POLY × STK.

Finally, Figure 4 illustrates the learning curve of the best kernels, i.e. POLY × STK, the advanced polynomial kernel applied on top of it, i.e. $(1+\text{POLY}\times\text{STK})^2$,

[10] Although the Std. Dev. associated with the model accuracy is high, the one associated with the distribution of difference between the model accuracy is much lower (about 2%). Considering also that we used 10 folds, we could verify that the first is better than the second at 90% of confidence limit.

$POLY^2$ and $SK_3 \times STK$. The plots show that the best kernels including the syntactic information are superior to the very accurate and rich kernels based on only BOS.

5 Related Work and Discussion

In this section we discuss some NLIDBs that have been tested on GeoQueries[11] or RestQueries datasets. NLIDBs can be classified according to the approach used to retrieve an answer to a given question from a database. For sake of space limit, we only discuss a system for each different approach. For a complete review of other state-of-the-art systems, please refer to Chandra and Mihalcea [33].

Authoring systems rely on semantic grammar specified by an expert user (i.e. the author) to interpret question over a database. The author has to name database elements, tailor entries and define additional concepts. CatchPhrase [3] is an authoring system that has been evaluated on GeoQueries250. In particular, two students were asked to author the system to cover 100 of the 250 questions each. Then the remaining questions were split in 2 test sets and translated by the system first into logical queries in tuple calculus representation and then into SQL queries. The average accuracy was 69%.

Many systems instead adopt a machine learning approach to induce semantic grammars from corpora of correct pairs of questions and queries, e.g. Krisp [1]. This takes pairs of sentences and their computer-executable meaning representations as training input to find a mapping between sentences and Prolog assertions using an SVM classifier. For each production in the meaning representation language the model is learned using string subsequence kernels. Then the classification is used to compositionally represent a natural language sentence in its meaning representation. The reported experiments using standard 10-fold cross validation show an accuracy of 70% and of 75% on GeoQueries880 and GeoQueries250, respectively.

In Precise [2], the derivation of semantic interpretation of ambiguous phrases is reduced to a graph matching problem. Precise finds valid mapping(s) from a complete tokenization of a given question to a set of database elements and then converts them into a SQL query (queries). The system achieves 100% Precision on a subset of questions while rejecting semantically intractable questions for a final Recall of 77.5% and 95% for GeoQueries880 and RestQueries respectively.

The performance of the above mentioned systems were originally measured according to Precision and Recall since they do not to generate a correct answer in particular output conditions. In contrast our approach allows for always having one answer, therefore it can be more appropriately measured with accuracy. We note that our approach is comparable to Krisp obtaining the similar outcome,

[11] GeoQueries250 is a subset of GeoQueries880 dataset whose questions are also available in other languages. Since our learning algorithm is language independent, we plan to experiment with other natural languages but also with GeoQueries880 so we report others' result also on this dataset.

i.e. 75.6% vs 75% (of Krisp), on GeoQueries250. Regarding the comparison with Precise, it should be noted that we corrected several errors in the SQL testset prepared in [2] (many of queries did not return the correct values and others were syntactically incorrect).

It is worth noting that our system, in contrast with previous generative approaches, retrieves the best matching query among the given set of all possible queries. One could argue that we cannot find a correct answer to a given unseen NL question if the SQL query is not present in the initial dataset. However, we can rely on query logs which reliably represent frequent and required queries asked to DBs.

There exist other systems [4,5,7,8] that were tested on GeoQueries880 with different experimental-setup, so results are not directly comparable.

With respect to other natural language tasks that employ tree kernels, several models have been proposed, e.g. [11,34,35,36,37,38,39,40,41,42].

6 Conclusions

In this paper, we approach the problem of deriving a shared semantic between natural language and programming language by automatically learning a model based the syntactical representation of the training examples. In our experiments we consider pairs of NL questions and SQL queries as training examples. These are annotated by means of our algorithm starting from a given initial annotation. In particular we experimented with the annotation available in GeoQueries250 and RestQueries corpora. We generated new datasets adding new positive pairs, creating negatives example set and also fixing some errors. Our datasets are publicly available so that other systems can be compared with our benchmark corpora.

To represent syntactic/semantic relationships expressed by training pairs, we encode such pairs in SVM by means of kernel functions. We designed innovative combinations between different kernels for structured data applied to pairs of objects, that, to the best of our knowledge, represent a novel approach to describe relational semantics between NL and SQL languages.

Experimental results show a promising accuracy, which can be largely improved, e.g. by model tuning. This suggests that our approach is viable to mine semantic relations between natural language and SQL.

In the future we would like to extend this research by focusing on advanced shallow semantic approaches such as predicate argument structures [43].

References

1. Kate, R.J., Mooney, R.J.: Using string-kernels for learning semantic parsers. In: Proceedings of the 21st ICCL and 44th Annual Meeting of the ACL, Sydney, Australia, July 2006, pp. 913–920. Association for Computational Linguistics (2006)
2. Popescu, A.M., Etzioni, A.O., Kautz, A.H.: Towards a theory of natural language interfaces to databases. In: Proceedings of the 2003 International Conference on Intelligent User Interfaces, Miami, pp. 149–157. Association for Computational Linguistics (2003)

3. Minock, M., Olofsson, P., Näslund, A.: Towards building robust natural language interfaces to databases. In: Kapetanios, E., Sugumaran, V., Spiliopoulou, M. (eds.) NLDB 2008. LNCS, vol. 5039, pp. 187–198. Springer, Heidelberg (2008)
4. Zettlemoyer, L.S., Collins, M.: Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In: UAI, pp. 658–666 (2005)
5. Wong, Y.W., Mooney, R.: Learning for semantic parsing with statistical machine translation. In: Proceedings of the Human Language Technology Conference of the NAACL, Main Conference, New York City, USA, June 2006, pp. 439–446. Association for Computational Linguistics (2006)
6. Dale, R., Somers, H.L., Moisl, H. (eds.): 9. In: Database Interfaces, pp. 209–240. Marcel Dekker Inc., New York (2000)
7. Tang, L.R., Mooney, R.J.: Using multiple clause constructors in inductive logic programming for semantic parsing. In: Proceedings of the 12th European Conference on Machine Learning, Freiburg, Germany, pp. 466–477 (2001)
8. Ge, R., Mooney, R.: A statistical semantic parser that integrates syntax and semantics. In: Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005), Ann Arbor, Michigan, June 2005, pp. 9–16. Association for Computational Linguistics (2005)
9. Winograd, T.: Understanding Natural Language. Academic Press, New York (1972)
10. Lodhi, H., Taylor, J.S., Cristianini, N., Watkins, C.J.C.H.: Text classification using string kernels. In: NIPS, pp. 563–569 (2000)
11. Collins, M., Duffy, N.: New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In: Proceedings of ACL 2002 (2002)
12. Vishwanathan, S.V.N., Smola, A.J.: Fast kernels for string and tree matching. In: Advances in Neural Information Processing Systems, vol. 15, pp. 569–576. MIT Press, Cambridge (2003)
13. Moschitti, A.: Efficient convolution kernels for dependency and constituent syntactic trees. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) ECML 2006. LNCS (LNAI), vol. 4212, pp. 318–329. Springer, Heidelberg (2006)
14. Giordani, A., Moschitti, A.: Semantic mapping between natural language questions and sql queries via syntactic pairing. In: NLDB 2009: Proceedings of the 13th international conference on Natural Language and Information Systems (2009)
15. Shawe-Taylor, J., Cristianini, N.: Kernel Methods for Pattern Analysis. Cambridge University Press, Cambridge (2004)
16. Zhang, D., Lee, W.S.: Question classification using support vector machines. In: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval, pp. 26–32. ACM Press, New York (2003)
17. Salton, G.: Recent trends in automatic information retrieval. In: SIGIR 1986, Proceedings of the 9th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Pisa, Italy, September 8-10, 1986, pp. 1–10. ACM, New York (1986)
18. Joachims, T.: Making large-scale SVM learning practical. In: Schölkopf, B., Burges, C., Smola, A. (eds.) Advances in Kernel Methods (1999)
19. Moschitti, A., Quarteroni, S., Basili, R., Manandhar, S.: Exploiting syntactic and shallow semantic kernels for question/answer classification. In: Proceedings of ACL 2007, Prague, Czech Republic (2007)
20. Moschitti, A., Quarteroni, S.: Kernels on linguistic structures for answer extraction. In: Proceedings of ACL 2008: HLT, Short Papers, Columbus, Ohio (2008)
21. Chali, Y., Joty, S.: Improving the performance of the random walk model for answering complex questions. In: Proceedings of ACL 2008: HLT, Short Papers, Columbus, Ohio, pp. 9–12 (2008)

22. Shen, D., Lapata, M.: Using semantic roles to improve question answering. In: Proceedings of EMNLP-CoNLL (2007)
23. Surdeanu, M., Ciaramita, M., Zaragoza, H.: Learning to rank answers on large online QA collections. In: Proceedings of ACL 2008: HLT, Columbus, Ohio (2008)
24. Basili, R., Moschitti, A., Pazienza, M.: A text classifier based on linguistic processing. In: Proceedings of IJCAI 1999, Machine Learning for Information Filtering (1999)
25. Charniak, E.: A maximum-entropy-inspired parser. In: Proceedings of NAACL 2000 (2000)
26. Cancedda, N., Gaussier, E., Goutte, C., Renders, J.M.: Word sequence kernels. J. Mach. Learn. Res. 3, 1059–1082 (2003)
27. Moschitti, A.: Kernel methods, syntax and semantics for relational text categorization. In: Proceeding of CIKM 2008, NY, USA (2008)
28. Moschitti, A., Bejan, C.: A semantic kernel for predicate argument classification. In: Proceedings of CoNLL 2004, Boston, MA, USA (2004)
29. Moschitti, A., Coppola, B., Pighin, D., Basili, R.: Engineering of syntactic features for shallow semantic parsing. In: Proceedings of ACL 2005 Workshop on Feature Engineering for Machine Learning in NLP, USA (2005)
30. Moschitti, A., Pighin, D., Basili, R.: Tree kernels for semantic role labeling. Computational Linguistics 34(2), 193–224 (2008)
31. Moschitti, A., Zanzotto, F.: Fast and effective kernels for relational learning from texts. In: Ghahramani, Z. (ed.) Proceedings of the 24th Annual International Conference on Machine Learning, ICML 2007 (2007)
32. Moschitti, A., Pighin, D., Basili, R.: Semantic role labeling via tree kernel joint inference. In: Proceedings of CoNLL-X, New York City (2006)
33. Chandra, Y., Mihalcea, R.: Natural language interfaces to databases, University of North Texas, Thesis, M.S. (2006)
34. Kudo, T., Matsumoto, Y.: Fast Methods for Kernel-Based Text Analysis. In: Hinrichs, E., Roth, D. (eds.) Proceedings of ACL, pp. 24–31 (2003)
35. Cumby, C., Roth, D.: Kernel Methods for Relational Learning. In: Proceedings of ICML 2003, Washington, DC, USA, pp. 107–114 (2003)
36. Culotta, A., Sorensen, J.: Dependency Tree Kernels for Relation Extraction. In: ACL 2004, Barcelona, Spain, pp. 423–429 (2004)
37. Kudo, T., Suzuki, J., Isozaki, H.: Boosting-based parse reranking with subtree features. In: Proceedings of ACL 2005, US (2005)
38. Toutanova, K., Markova, P., Manning, C.: The Leaf Path Projection View of Parse Trees: Exploring String Kernels for HPSG Parse Selection. In: Proceedings of EMNLP 2004, Barcelona, Spain (2004)
39. Kazama, J., Torisawa, K.: Speeding up Training with Tree Kernels for Node Relation Labeling. In: Proceedings of EMNLP 2005, Toronto, Canada, pp. 137–144 (2005)
40. Shen, L., Sarkar, A., Joshi, A.k.: Using LTAG Based Features in Parse Reranking. In: EMNLP, Sapporo, Japan (2003)
41. Zhang, M., Zhang, J., Su, J.: Exploring Syntactic Features for Relation Extraction using a Convolution tree kernel. In: Proceedings of NAACL, New York City, USA, pp. 288–295 (2006)
42. Zhang, D., Lee, W.: Question classification using support vector machines. In: Proceedings of SIGIR 2003, Toronto, Canada. ACM Press, New York (2003)
43. Giuglea, A.M., Moschitti, A.: Semantic role labeling via framenet, verbnet and propbank. In: Proceedings of ACL 2006, Sydney, Australia (2006)