# Classification of Attributes in a Natural Language Query into Different SQL Clauses

**Ashish Palakurthi**[†], **Ruthu S. M.**[†], **Arjun R. Akula**[†,*] **and Radhika Mamidi**[†]
[†]Language Technologies Research Center, IIIT Hyderabad, India.
[*]IBM Research, Bangalore, India.
ashish.palakurthi@research.iiit.ac.in,
ruthus.m@students.iiit.ac.in, arakula@in.ibm.com,
radhika.mamidi@iiit.ac.in

## Abstract

Attribute information in a natural language query is one of the key features for converting a natural language query into a Structured Query Language[1] (SQL) in Natural Language Interface to Database systems. In this paper, we explore the task of classifying the attributes present in a natural language query into different SQL clauses in a SQL query. In particular, we investigate the effectiveness of various features and Conditional Random Fields for this task. Our system uses a statistical classifier trained on manually prepared data. We report our results on three different domains and also show how our system can be used for generating a complete SQL query.

## 1 Introduction

Databases have become one of the most efficient ways to store and retrieve information. Database systems require a user to have the knowledge of structured languages in order to be able to retrieve information from them. As a result, it becomes difficult for people of non-technical background to use databases. Natural Language Interface to Database (NLIDB) (Androutsopoulos et al., 1995; Catalina Hallett and David Hardcastle, 2008; Pazos et al., 2002; Popescu et al., 2003; Giordani and Moschitti, 2009; Gupta et al., 2012) systems provide an interface through which a user can ask a query in natural language and get the required information from the database. NLIDB systems translate the user's natural language (NL) query into a SQL query, thereby allowing the user to retrieve the answer from the database. However, NLIDB systems are not widely used because of their inability to process ambiguity and complexity of natural language, which makes them more error prone. Thus, it becomes very important to capture even the smallest of the information from a NL query before converting it into a SQL query.

A relational database contains objects called tables in which information is stored. These tables contain columns and rows. The column names in the tables are known as attributes. A SQL query is composed of different SQL clauses like SELECT, FROM, WHERE, GROUP BY, HAVING and ORDER BY. Since clauses in a SQL query have attributes, attribute information becomes very important for an effective conversion of a NL query into a SQL query. Explicit attributes are the attributes mentioned by the user in the NL query text. When a NL query is converted into SQL query, explicit attributes may belong to different SQL clauses. In this paper, we use Conditional Random Fields (CRF) for classifying the explicit attributes in a NL query to different SQL clauses.

## 2 Related Work

There have been significant research efforts in the area of NLIDB systems. Different approaches have been proposed to deal with these systems.

In (Gupta et al., 2012), the authors propose a NLIDB system based on Computational Paninian Grammar (CPG) Framework (Bharati et al., 1996). They emphasize on syntactic elements as well as the semantics of the domain. They convert a NL query into a SQL query by processing the NL query in three stages, viz. the syntactic stage, the semantic stage and the graph processing stage. In the semantic stage, they identify attribute-value pairs for various entities using noun frames. The problem with this proposal is that it becomes costly in terms of space to use high number of frames in large domains. In (Khalid et al., 2007), machine learning was used in Question Answering systems.

[1]Structured Query Language is a specialized language used for relational database management and data manipulation.

The system proposed by them maps an input query to certain tables containing attributes which can provide the required answer. They specify that identifying the related tables and attributes from the knowledge base is very important for answering an incoming question. Amany Sarhan (2009) emphasized on the importance of identifying the table names of attributes in a NL query. He shows that attribute and table information help in minimizing the effort to build SQL queries. Thus, attribute information plays a very important role in both NLIDB systems and Question Answering systems. To our knowledge, this work is the first attempt to classify attributes directly from a NL query to different SQL clauses in their SQL queries. In (Srirampur et al., 2014), the authors address the problem of Concepts Identification of a NL query in NLIDB, which plays a crucial role for our system to generate a complete SQL query.

The remainder of this paper is structured as follows. Section 3 describes the problem. In section 4, we illustrate the concept of explicit attribute classification. Section 5 explains the methodology along with the features adopted for the classification. We also discuss on generating the complete SQL query. In section 6, we show experimentations and results along with error analysis. We conclude in section 7.

## 3 Problem

An attribute in a NL query can correspond to various SQL clauses. We define two types of attributes which can be found in a NL query.

**Explicit attributes:** Explicit attributes are the attributes which are directly mentioned by the user in the NL query.

**Implicit attributes:** Implicit attributes are not directly mentioned by the user in the NL query. These attributes are identified with the help of values mentioned by the user in the NL query. For identifying these attributes, domain dictionaries can be used. Table 1 shows a sample domain dictionary. The following examples illustrate explicit and implicit attributes in a user query.

Example 1: *List the grades of all the students in Mathematics.*

In this example, *grade* is an explicit attribute as it is directly mentioned by the user in the NL query. The user also mentions the value *Mathematics*. This value when checked in the domain dictionary gives the attribute course_name as *Mathematics* is the name of a course. Thus, course_name is an implicit attribute. The attribute *students* or student_name is another explicit attribute in the above example.

Example 2: *What course does Smith teach?*

In this example, *course* or course_name is an explicit attribute as it is directly mentioned by the user. The user mentions the value *Smith*. This value when checked in the domain dictionary gives the attribute professor_name if *Smith* is a name of a professor. Thus, the attribute professor_name is an implicit attribute.

Implicit attributes generally correspond to the WHERE clause in a SQL query as they are associated with a value. This paper focusses on classifying explicit attributes into different SQL clauses in a SQL query.

| Value | Attribute |
|---|---|
| Smith | professor_name |
| ABCD | lab_name |
| John | student_name |
| Science | course_name |

Table 1: Sample domain dictionary

## 4 Explicit Attribute Classification

In this section, we illustrate the classification of explicit attributes from a NL query into different clauses in a SQL query. In all the examples shown in Figure 1, course_name (courses) is explicitly mentioned by the user. In each example, the attribute course_name belongs to a different SQL clause. In Example 1 (Figure 1), course_name belongs to the SELECT clause as the user has asked to the list courses taught by Smith. In Example 2 (Figure 1), course_name belongs to the WHERE clause as it gives information about a course (Science). Note that Science can be identified as course_name from the domain dictionary as well. In Example 3, the user is asking to show a student from each course. So it is required to group the students according to their course (course_name) and then list them. Thus, the attribute course_name should belong to the GROUP BY clause. Note that, in the same example the user has also mentioned another attribute (students) explicitly. Since he has asked to list the students, the attribute student_name will belong to the SELECT clause. In Example 4, the two explicit attributes mentioned by the user are professors and courses. Here, the user is asking to list only those professors who teach more than two courses. Here, we group according to professors and then for each professor, we count the number of courses taught. Only if the count is greater than 2, we select the professor and list his name. Thus, professor_name belongs to the GROUP BY clause. The condition on professors is COUNT *(course_name)* $> 2$. Therefore, the attribute *courses* or course_name belongs to the HAVING clause. In this way, by identifying the clauses to which the attributes belong, we can improve the translation of NL queries to SQL queries. In the next section, we describe how we classify these explicit attributes to their SQL clauses.

1. *What are the courses taught by Smith?*
**SELECT** course_name
**FROM** COURSES, TEACH, PROFESSOR
**WHERE** professor_name= "Smith" AND
prof_id=prof_teach_id AND
course_teach_id=course_id.
2. *Who teaches Science course?*
**SELECT** professor_name
**FROM** COURSES, TEACH, PROFESSOR
**WHERE** course_name="Science" AND
course_id =course_teach_id AND
prof_teach_id=prof_id
3. *List a student from each course.*
**SELECT** student_name, course_name
**FROM** STUDENTS, REGISTER, COURSES
**WHERE** stud_id=stud_reg_id AND
reg_id=course_id
**GROUP BY** course_name
4. *Who are the professors teaching more than 2 courses?*
**SELECT** professor_name
**FROM** COURSES, TEACH, PROFESSOR
**WHERE** course_id=course_teach_id AND
prof_teach_id =prof_id
**GROUP BY** professor_name
**HAVING** COUNT(course_name) > 2

Figure 1: Examples of NL queries and their SQL queries

## 5 Methodology

We manually prepared a dataset of queries on the Academic domain of our university. The university database was used as the source of information. Examples of tables in the database schema are *courses, labs, students* consisting of attributes like course_name, course_id, student_name, lab_name etc. The database has relationships like register (between student and course), teach (between professor and course) etc. Each token in the sentence is given a tag and a set of features. If a token is an attribute, it is assigned a tag which corresponds to a SQL clause to which the attribute belongs. If a token is not an attribute, it is given a NULL (O) tag. The tagging was done manually. Our tag set is simple and consists of only 4 tags, where each tag corresponds to a SQL clause. The tags are SELECT, WHERE, GROUP BY, HAVING. Formally, our task is framed as assigning label sequences to a set of observation sequences.

| Token | Attribute | Tag |
|---|---|---|
| What | 0 | O |
| are | 0 | O |
| the | 0 | O |
| courses | 1 | GROUP BY |
| with | 0 | O |
| less | 0 | O |
| than | 0 | O |
| 25 | 0 | O |
| students | 1 | HAVING |
| ? | 0 | O |

Table 2: Example of tagging scheme

We followed two guidelines while tagging sentences. Sometimes it is possible that an attribute can belong to more than one SQL clause. If an attribute belongs to both SELECT and GROUP BY clause, we tag the attribute as a GROUP BY clause attribute. This is done with an aim to identify higher number of GROUP BY clause attributes as SELECT clause attributes are very common and are comparatively easier to identify. The second guideline that we followed was, if an attribute belongs to both the SELECT and the WHERE clause, we tag the attribute as a SELECT clause attribute. This is done because the WHERE clause attributes can often be identified through a domain dictionary. Table 2 shows an example of the tagging scheme. Each token in a sentence is given a set of features and a tag. In Table 2, we have shown only one feature due to space constraints. We trained our data and created models for testing. We used Conditional Random Fields (Lafferty et al., 2001) for the machine learning task. The next subsection describes the features employed for the classification of explicit attributes in a NL query.

## 5.1 Classification Features

The following features were used for the classification of explicit attributes in a NL query.

**Token-based Features** These features are based on learning of tokens in a sentence. The *isSymbol* feature checks whether a token is a symbol (>, <) or not. Symbols like > (greater than), < (less than) are quite commonly used as aggregations in NL queries. This feature captures such aggregates. We also took lower case form of a token as a feature for uniform learning. We considered a particular substring as a feature. If that substring is found in the token, we set the feature to 1 else 0 (for example, in batch wise or batchwise, the attribute *batch* is identified as GROUP BY clause attribute using substring *wise*).

**Grammatical Features** POS tags of tokens and grammatical relations (e.g. nsubj, dobj ) of a token with other tokens in the sentence were considered. These features were obtained using the Stanford parser[2] (Marneffe et al., 2006).

**Contextual Features** Tokens preceding and following (*local context*) the current token were also considered as features. In addition, we took the POS tags of the tokens in the local context of the current token as features. Grammatical relations of the tokens in local context of the current token were also considered for learning.

**Other Features:**

**isAttribute** This is a basic and an important feature for our problem. If a token is an attribute, we set the feature to 1, else 0.

**Presence of other attributes** This feature aims to identify the GROUP BY clause attributes only. In SQL, the HAVING clause generally contains a condition on the GROUP BY clause. If a NL query is very likely (>95%) to have a HAVING clause attribute, then the SQL clause will certainly have a GROUP BY clause as well. This feature is marked as 1 for an attribute if it has a local context which may trigger a GROUP BY clause and at the same time if the NL query is very likely to have the HAVING clause attribute. The likeliness of the HAVING clause attribute is again decided based on the local context of the attribute. Thus, GROUP BY clause attribute is not just identified using its local context, but also depending on the presence of HAVING clause attribute. In simple terms, this feature increases the weight of an attribute to belong to the GROUP BY clause of the SQL query.

**Trigger words** An external list is used to determine whether a word in the local context of an attribute may trigger a certain SQL clause for the attribute. (eg., the word *each* may trigger GROUP BY clause).

## 5.2 Completing the SQL Query

Until now, we have only identified attributes and their corresponding SQL clauses. But this is not sufficient to get a complete SQL query. In this section, we describe how we can generate a complete SQL query after the classification of attributes. To build a complete SQL query we would require:
1. Complete attribute and entity information.
2. Concepts[3] of all the tokens in the given query.
3. Mapping of identified entities and relationships in the Entity Relationship schema to get the joint conditions in WHERE clause.

Our system can extract attribute information using explicit attribute classifier for explicit attributes and domain dictionaries for implicit attributes. Sometimes, we may not have complete attribute information to form a SQL query. That is, there can be attributes other than explicit attributes and implicit attributes in a SQL query. For example, consider:

Example 1: *Which professor teaches* NLP *?*
Example 2: *Who teaches* NLP *?*

The SQL query for both the examples is:
SELECT professor_name
FROM prof, teach, course
WHERE course_name= NLP AND
course_id=course_teach_id AND
prof_teach_id=prof_id .

In example 1, our system has complete attribute information to form the SQL query. Since professor is explicitly mentioned by the user in the query, here professor_name is identified as a SELECT clause attribute by our system. But in example 2, we do not have complete attribute information. Here identifying the SELECT clause attribute professor_name is a problem, as there is no clue (neither explicit attribute nor implicit attribute) in the query which points us to the attribute professor_name. To identify attributes which cannot be identified as implicit attributes or explicit at-

[2] http://nlp.stanford.edu/software/lex-parser.shtml

[3] Concept of a NL token maps the NL token to the database schema. The tables, attributes and relations in the database schema constitute concepts.

tributes, Concepts Identification (Srirampur et al., 2014) is used. In Concepts Identification, each token in the NL query is tagged with a concept. Using Concepts Identification, we can directly identify *Who* as professor_name. These attributes are known as the *Question class* attributes. Most of the times, since question words are related to the SELECT clause, the attribute professor_name can be mapped to the SELECT clause, thereby giving us complete information of attributes. We also use Concepts Identification to identify relationships in the NL query. In both the examples, *teach* which is a relationship in the Entity Relationship schema can be identified through Concepts Identification (CI). Once the attributes are identified, entities can be extracted. For example, entities for the attributes course_name, professor_name are COURSES and PROFESSOR respectively. The identified entities and relationships are added to the FROM clause.

All the identified entities and relationships can now be mapped to the Entity Relationship (ER) schema to get the joint conditions (Arjun Reddy Akula, 2015) in the WHERE clause. We create an ER graph using the ER schema of the database with entities and relationships as vertices in the ER graph. We apply a Minimum spanning tree (MST) algorithm on the ER graph to get a non-cyclic path connecting all the identified vertices in the ER graph. With this, we get the required join conditions in the WHERE clause. Arjun Reddy Akula (2015) discusses the problem of handling joint conditions in detail. Note that new entities and relationships can also be identified while forming the MST. These extra entities and relationships are added to the FROM clause in the SQL query. We now have a complete SQL query.

## 6 Experiments and Discussions

### 6.1 Data

We manually prepared a rich dataset ensuring that NL queries when converted into SQL queries, a wide variety of SQL queries are covered for the classification. We tested our classifier on these queries. Apart from the Academic domain, we also experimented on Mooney's dataset[4]. We considered the Restaurant and the Geoquery domains (Wong and Mooney, 2006) in Mooney's dataset. The Geoquery dataset (GEO880) consisted of 880 queries. Since we are not addressing nested SQL queries, we removed queries which when converted to SQL queries, involve nested SQL queries. This task was done manually. There were 256 nested SQL queries in the Geoquery dataset. Regarding classification, we mainly focused on the Academic domain as it consists of queries with the SELECT, WHERE, GROUP BY and the HAVING clause attributes. The Restaurant and Geoquery domains had queries with only the SELECT and WHERE clause attributes. This is one of the reasons why we prepared the data ourselves. Table 3 shows the number of sentences considered for training and testing in each domain.

| **Domain** | **Train** | **Test** |
|---|---|---|
| Academic | 711 | 305 |
| Restaurant | 150 | 100 |
| Geoquery | 400 | 224 |

Table 3: Corpus statistics

### 6.2 Experimental Results

We used the metrics of Precision (P), Recall (R) and F-measure[5] (F) for evaluation.

#### 6.2.1 Baseline Method

We first determine the majority class *C* of an explicit attribute *A* found in the training data. The baseline system then labels all occurrences of *A* found in the test data with class *C*, irrespective of the context of the attribute. In all the three domains, SELECT clause attribute was the majority class attribute. Table 4 summarizes the results of the baseline method in all the domains.

| **Domain** | **P**(%) | **R**(%) | **F**(%) |
|---|---|---|---|
| Academic | 46.29 | 46.37 | 46.33 |
| Restaurant | 47.75 | 43.80 | 45.69 |
| Geoquery | 63.08 | 63.08 | 63.08 |

Table 4: Baseline method results

#### 6.2.2 Conditional Random Fields

We used Conditional Random Fields for the classification experiments since it represents the state

[4]http://www.cs.utexas.edu/users/ml/nldata.html

[5]
$$F - measure = \frac{2 * P * R}{P + R}$$

of the art in sequence modeling and has also been very effective at Named Entity Recognition (NER). As our problem is very similar to NER, we used CRF. CRF++[6] tool kit was used for this. CRFs are a probabilistic framework used for labeling sequence data. CRF models effectively solve the label bias problem, which make it better than HMMs which are generally more likely to be susceptible to the label bias problem. Our discussions mainly focus on Academic domain.

We conducted experiments in three phases. Phase one involved using features only for the current token. The system achieved a F-measure of 60.27%.

| **Domain** | **Clause** | **P**(%) | **R**(%) | **F**(%) |
|---|---|---|---|---|
| Academic | SELECT | 60.94 | 89.80 | 72.61 |
| | WHERE | 48.81 | 57.75 | 52.90 |
| | GROUP BY | 72.37 | 38.73 | 50.46 |
| | HAVING | 14.29 | 1.52 | 2.74 |
| | **Overall** | **60.04** | **60.50** | **60.27** |
| Restaurant | SELECT | 81.08 | 92.31 | 86.33 |
| | WHERE | 96.30 | 92.86 | 94.55 |
| | **Overall** | **87.50** | **92.56** | **89.96** |
| Geoquery | SELECT | 78.21 | 98.05 | 87.01 |
| | WHERE | 94.12 | 53.33 | 68.09 |
| | **Overall** | **81.54** | **81.54** | **81.54** |

Table 5: Results obtained without considering contextual features.

In phase two, we added contextual features as well. The contextual features include tokens surrounding the current token, POS tags of the tokens surrounding the current token and also the grammatical relations of the tokens surrounding the current token.

Incorporating contextual features showed a significant improvement in the classification. At the end of phase two, the F-measure of the system was 83.73%. This shows that the local context of an attribute is important in deciding its SQL clause. Table 5 and Table 6 show the classification results of phase one and phase two respectively. By local context, we mean the neighbouring tokens or features of neighbouring tokens of the attribute in the NL query. After a few pilot experiments, context window of size three was found to be optimal in Academic domain and context window of size one was enough for Restaurant and Geoquery domains. Window size of three was required specially for HAVING clause attributes. This is probably because HAVING clause attributes are generally associated with aggregations. Hence, local context of an attribute is very important for the HAVING class attributes. As can be seen from Table 5 and Table 6, adding contextual features increased the F-measure of HAVING clause attributes by 60.53 percentage points. The presence of attribute feature is very important for identifying the GROUP BY clause attributes. F-measure of GROUP BY clause attributes increased by 11.72 percentage points on adding this feature. The reason for higher F-measures in the Restaurant and the Geoquery domains is mainly because these domains had NL queries with only the SELECT and the WHERE clause attributes, thus making classification much easier. Moreover, the randomness found in queries was comparatively lesser than the Academic domain. In addition, the problem of contextual conflicts was not seen in these domains. Contextual conflicts are discussed in the error analysis section.

[6] https://code.google.com/p/crfpp

| **Domain** | **Clause** | **P**(%) | **R**(%) | **F**(%) |
|---|---|---|---|---|
| Academic | SELECT | 88.12 | 93.88 | 90.91 |
| | WHERE | 56.41 | 92.96 | 70.21 |
| | GROUP BY | 96.58 | 79.58 | 87.26 |
| | HAVING | 96.88 | 46.97 | 63.27 |
| | **Overall** | **83.49** | **83.97** | **83.73** |
| Restaurant | SELECT | 94.64 | 81.54 | 87.60 |
| | WHERE | 100.00 | 98.21 | 99.10 |
| | **Overall** | **97.30** | **89.26** | **93.10** |
| Geoquery | SELECT | 89.04 | 99.02 | 93.76 |
| | WHERE | 97.94 | 79.17 | 87.56 |
| | **Overall** | **91.69** | **91.69** | **91.69** |

Table 6: Results obtained on adding contextual features.

| **Train** | **Test** | **P**(%) | **R**(%) | **F**(%) |
|---|---|---|---|---|
| Academic | Restaurant | 69.63 | 77.69 | 73.44 |
| Academic | Geoquery | 70.99 | 70.77 | 70.88 |
| Restaurant | Academic | 52.55 | 51.15 | 51.84 |
| Restaurant | Geoquery | 80.66 | 60.31 | 69.01 |
| Geoquery | Academic | 50.57 | 50.95 | 50.76 |
| Geoquery | Restaurant | 77.78 | 86.78 | 82.03 |

Table 7: Cross domain results

In phase three, we performed cross domain experiments. Here, we train a model on a dataset of one domain and test the model on the dataset of a different domain. We do not consider current token as a feature since the attributes are different in each domain. But, features like POS and grammatical relations of the current token were taken. Contextual tokens and other features of contextual tokens were considered. Table 7 shows the results of phase three experiments. Using contextual features in a supervised machine learning framework captures a strong generalization for classifying the attributes.

Finally, we compare the final results we were able to achieve to the state-of-the-art. Many NLIDB systems have been proposed using different approaches. We discuss few of them. PRECISE (Popescu et al., 2003) is a system which converts semantic analysis to a graph matching problem using schema elements. A class of semantically tractable queries is proposed and the system can form SQL queries only if a query belongs to the proposed class. PRECISE achieves an overall F-measure of 87% on 700 tractable queries from the GEO880 (Geoquery domain) corpus and a recall of 95% in restaurant domain. In KRISP (Kate et al., 2006), a user query is mapped to its formal meaning representations using kernel based classifiers. These classifiers were trained on string subsequence kernels and were used to build complete meaning representation of the user query. They achieve a precision of 94% and recall of 78% on the GEO880 corpus.

Support Vector Machines with kernel methods (Giordani et al., 2009) were adopted to represent syntactic relationships between NL and SQL queries. The authors apply different combinations of kernels and derive an automatic translator of NL query to SQL query. Their system achieves an overall accuracy of 76% and 84.7% for forming SQL queries in the Geoquery and the Restaurant domains respectively.

A set of candidate SQL queries (Giordani et al., 2012) are produced using lexical dependencies and metadata of the database. These SQL queries are then re-ranked using SVM with tree kernels. Using few heuristics they generate final list of SQL queries. They achieved F-measure of 85% on the GEO880 corpus. Recent work (Clarke et al., 2010) tackles semantic parsing using supervision. Here, the system predicts complex structures based on feedback of external world. From the GEO880 corpus, they randomly select 250 queries for training and 250 queries for testing and achieved an overall F-measure of 73%.

However, there have not been any efforts in mapping NL queries to SQL queries exclusively from an attribute point of view. Attributes being the building blocks of a SQL query, we focus on attributes to build a SQL query. After attribute classification, Concept Identification and identification of the joint conditions in the WHERE clause, we evaluate the overall SQL query formation. Even if one attribute is wrongly tagged, we consider the SQL query wrong. After accounting to Concepts Identification errors and domain dictionary errors, the final accuracies achieved by our system were 75%, 71% and 64% in Restaurant, Geoquery[7] and Academic domains respectively.
We define accuracy as

$$Accuracy = \frac{\text{Number of correctly retrieved SQL queries}}{\text{Total Number of queries}}$$

These accuracies[8] are on the same test datasets used for attribute classification(Table 3). Apart from wrong tagging of attributes, one interesting error we found while forming SQL queries was domain dictionary error. For example, consider the query, *What length is the Mississippi?*. Here, the user is talking about Mississippi river, but the domain dictionary tags Mississippi as state_name. However, if the query had been asked as *What length is the Mississippi river ?*, the system uses the explicit attribute *river* and retrieves Mississippi as river_name. In summary, we achieve competitive results using a novel approach and move towards tackling domain independency.

### 6.3 Error Analysis in Attribute Classification

Most of the errors occurred due to contextual conflicts which are of two types. Contextual conflict between two attributes $A$ and $B$ is an instance wherein both the attributes $A$ and $B$ have same local context but are found to be classified under different SQL clauses $\hat{A}$ and $\hat{B}$. We say that there is a contextual conflict between $\hat{A}$ and $\hat{B}$ clause attributes. The observed contextual conflicts ($>$ 90%) were:

**SELECT clause vs GROUP BY clause attributes.**

[7]The results on Geoquery domain may actually be worse as the data (Table 3) we considered is a subset of the GEO880.

[8]Various state-of-the-art approaches show results on different train-test data splits. We achieved an accuracy of 74% in restaurant domain using standard 10-fold cross validation method. We do not show 10-fold cross validation results for Geoquery domain as the corpus we considered in this domain is a subset of the GEO880 dataset. However, the difference in results is likely not statistically significant.

For example, consider *List the courses in our college* and *List the batches in our college with more than 100 students*. Here, context of *courses* and *batches* is same. In the first example, the attribute *courses* (course_name) is a SELECT clause attribute and in the second example, the attribute batches (batch_name) is a GROUP BY clause attribute. But *batches* was misclassified as SELECT clause attribute. It should belong to the GROUP BY clause according to our annotation guidelines.

**WHERE clause vs HAVING clause attributes.** In the examples, *Who are the students with more than 8 marks in NLP?* and *What are the batches with more than 8 students?*, the prefix context of *marks* in the first example is same as the prefix context of *students* (more than 8) in the second example. The attribute *marks* in the first example belongs to the WHERE clause and *students* in the second example belongs to the HAVING clause. But *students* was misclassified as WHERE clause attribute. Another reason why one yields WHERE and the other HAVING is due to the way the database is organized internally. If the *batch* table has a *number of students* attribute, then the second example would also yield a WHERE clause. This is an inherent limitation of the NLIDB approach, not related to the features, classifier or the overall approach used.

Contextual conflicts were less in the Restaurant and the Geoquery domains when compared to the Academic domain, as they consisted of only SELECT and WHERE clause attributes. Errors in these domains were mainly token based errors.

## 7 Conclusion and Future Work

In this paper, we investigate a CRF model to classify attributes present in a NL query into different SQL clauses in a SQL query. We believe that this is the core part of SQL query formation. For explicit attribute classification, our system achieved overall F-measures of 83.73%, 93.10%, 91.69% in Academic, Restaurant and Geoquery domains respectively. We also achieved accuracies of 64%, 75% and 71% in forming SQL queries in Academic, Restaurant and Geoquery domains respectively. The main contributions of this paper are:

- We show that within a sentence, attributes can be used to build a SQL query. For this, the local context of an attribute can be helpful to identify its clause in the SQL query.
- We primarily focus on attribute classification as they are the building blocks of the SQL query. We then use an existing system to complete the SQL formation. We achieved promising results in forming SQL queries using a novel approach.
- The work presents a significant study on SQL clauses like GROUP BY and HAVING by manually creating a new dataset. To the best of our knowledge, benchmark datasets do not cover these SQL clauses as good as they cover SQL clauses like SELECT and WHERE.
- Experiments in cross domain datasets suggest that the proposed feature set learns a strong generalization for classifying the attributes in the NL query. To an extent, this certainly addresses the disadvantage of domain independency in NLIDB systems. Another advantage of learning the context of an attribute is that, it can be useful in classifying an unseen attribute within the same domain.

Finally, we claim that attributes are an important part of a user query to a NLIDB system. Exploring patterns on how these attributes are used by a user in a NL query can be useful to form a SQL query. The proposed approach may break down with NL queries having less explicit attributes, where the NL query may require deeper semantic processing. It would be interesting if we can combine our approach with existing parsing based approaches. In our future work, we will further improve the explicit attribute classification, incorporate semantic features to improve SQL query formation and handle nested SQL queries.

## Acknowledgements

We would like to thank the anonymous reviewers for the valuable feedback on this work. This research was supported in part by the Information Technology Research Academy (ITRA), Government of India under ITRA-Mobile grant ITRA/15(62)/Mobile/VAMD/01. Any opinions, findings, conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the ITRA.

## References

Rashid Ahmad, Mohammad Abid Khan, and Rahman Ali 2009. Efficient Transformation of a Natural Language Query to SQL for Urdu. In *Proceedings of the Conference on Language & Technology*, page p53.

Arjun R. Akula, Rajeev Sangal, Radhika Mamidi. 2013. A Novel Approach Towards Incorporating Context Processing Capabilities in NLIDB System. *In Proceedings of the International Joint Conference on Natural Language Processing (IJCNLP)*,pages 1216-1222, Nagoya, Japan.

Arjun R. Akula. 2015. A Novel Approach Towards Building a Generic, Portable and Contextual NLIDB System. International Institute of Information Technology Hyderabad.

Ioannis Androutsopoulos, Graeme D. Ritchie, and Peter Thanisch. 1995. Natural language interfaces to databasesan introduction. *Natural language engineering*,1(01), 29-81.

Raffaella Bernardi and Manuel Kirschner. 2008. Context modeling for iqa: the role of tasks and entities. In *Coling 2008: Proceedings of the workshop on Knowledge and Reasoning for Answering Questions,* pages 2532. Association for Computational Linguistics.

Nuria Bertomeu, Hans Uszkoreit, Anette Frank, Hansulrich Krieger and Brigitte Jorg. 2006. Contextual phenomena and thematic relations in database qa dialogues: results from a wizard-of-oz experiment. In *Proceedings of the Interactive Question Answering Workshop at HLT-NAACL* , pages 1-8 Association for Computational Linguistics.

Akshar Bharati, Medhavi Bhatia, Vineet Chaitanya, and Rajeev Sangal. 1996. Paninian grammar framework applied to English. In *Technical Report TRCS-96-238,CSE,IIT Kanpur, Tech.Rep.*

Joyce Y Chai and Rong Jin. 2004. Discourse structure for context question answering. In *Proceedings of the Workshop on Pragmatics of Question Answering at HLT-NAACL*, pages 23-30.

James Clarke, Dan Goldwasser, Ming-wei Chang and Dan Roth. 2010. Driving semantic parsing from the worlds response. In *ACL Conference on Natural Language Learning (CoNLL).*

Faraj A. El-Mouadib, Zakaria Suliman Zubi, Ahmed A.Almagrous, I. El-Feghi 2009. Generic interactive natural language interface to databases(GINLIDB). In *International Journal of Computers* 3(3).

Alessandra Giordani. 2008. Mapping Natural Language into SQL in a NLIDB. *In Natural Language and Information Systems* (pp. 367-371). Springer Berlin Heidelberg.

Alessandra Giordani and Alessandro Moschitti. 2009. Syntactic structural kernels for natural language interfaces to databases. *In Machine Learning and Knowledge Discovery in Databases,* pages 391406. Springer.

Alessandra Giordani and Alessandro Moschitti. 2009. Semantic Mapping between Natural Language Questions and SQL Queries via Syntactic Pairing. In *14th International Conference on Applications of Natural Language to Information Systems, Saarbrucken, Germany*.

Alessandra Giordani and Alessandro Moschitti. 2010. Corpora for Automatically Learning to Map Natural Language Questions into SQL Queries. In *Proceedings of the Seventh conference on International Language Resources and Evaluation(LREC), Malta.*

Alessandra Giordani and Alessandro Moschitti. 2012. Generating SQL Queries Using Natural Language Syntactic Dependencies and Metadata. *NLDB* 164-170.

Alessandra Giordani and Alessandro Moschitti. 2012. Automatic Generation and Reranking of SQL-Derived Answers to NL Questions. In *Proceedings of the Joint workshop on Intelligent Methods for Software System Engineering (JIMSE)held in ECAI 2012. Montpellier, France.*

Alessandra Giordani and Allesandro Moschitti 2012. Translating Questions to SQL Queries with Generative Parsers Discriminatively Reranked *COLING.*

Abhijeet Gupta, Arjun Akula, Deepak Malladi, Puneeth Kukkadapu, Vinay Ainavolu and Rajeev Sangal. 2012. A novel approach towards building a portable nlidb system using the computational paninian grammar framework. *In Asian Language Processing (IALP), 2012 International Conference on* (pp. 93-96). IEEE.

Catalina Hallett and David Hardcastle 2008. Towards a bootstrapping nlidb system In *Natural Language and Information Systems,* Springer, 2008, pp. 199-204.

Xiaofei Jia, and Mengchi Liu. 2003. Towards an Intelligent Information System *SBBD.*

Rohit J. Kate and Raymond J.Mooney. 2006. Using string-kernels for learning semantic parsers. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics. ACL.*

Mahboob Alam Khalid, Valentin Jijkoun and Maarten de Rijke 2007. Machine learning for question answering from tabular data. In *Proceedings of Database and Expert Systems Applications*:392-396. IEEE.

John Lafferty, Andrew McCallum, and Fernando CN Pereria. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Int.Conf. Machine Learning, San Francisco, CA*.

Yunyao Li, Huahai Yang, and HV Jagadish 2005. Nalix: an interactive natural language interface for querying xml. In Proceedings of the 2005 ACM SIGMOD international conference on Management of data, pages 900902. ACM.

Marie-Catherine de Marneffe, Bill MacCartney and Christopher D.Manning. 2006. Generating Typed Dependency Parses from Phrase Structure Parses. In *Proceedings of LREC.*

Xiaofeng Meng and Shan Wang. 2001 Nchiql: The chinese natural language interface to databases. In *Database and Expert Systems Applications* pages 145154. Springer.

M.Minock, Peter Olofsson and Alexander Nasslund. 2008. Towards building robust natural language interfaces to databases. In *Natural language and Information systems. Springler Berlin Heidelberg.* 187-198.

Ana-Maria Popescu, Oren Etzioni and Henry Kautz 2003. Towards a theory of natural language interfaces to databases. In *Proceedings of the 8th international conference on Intelligent user interfaces. ACM.*

Filipe P. Porfirio , and Nuno J. Mamede. 2000. Databases and Natural Language Interfaces *JISBD*.

Rodolfo A. Pazos Rangel , Alexander Gelbukh , J. Javier Gonzlez Barbosa , Erika Alarcn Ruiz , Alejandro Mendoza Meja , and A. Patricia Domnguez Snchez 2002. Spanish Natural Language Interface for a Relational Database Querying System. In *Sojka, P., Kopeek, I., Pala, K. (eds.) TSD 2002. LNCS (LNAI),* vol. 2448, pp. 123-130. Springer, Heidelberg (2002).

Amany Sarhan. 2009. A proposed architecture for dynamically built NLIDB systems. In *Knowledge-based and Intelligent Engineering Systems Journal* 13(2),IOS.

Saikrishna Srirampur, Ravi Chandibhamar, Ashish Palakurthi and Radhika Mamidi. 2014 Concepts identification of an NL query in NLIDB systems. In *Asian Language Processing (IALP), 2014 International Conference* on, pp. 230-233. IEEE.

Niculae Stratica, Leila Kosseim, and Bipin C Desai 2005 Using semantic templates for a natural language interface to the cindi virtual library. *Data and Knowledge Engineering*, 55(1):4-19.

Lappoon R. Tang and Raymond J. Mooney. 2001. Using Multiple Clause Constructors in Inductive Logic Programming for semantic Parsing In *Proceedings of the 12th European Conference on Machine Learning(ECML):*466-477.

Kristina Toutanova, Dan Klein, Christopher Manning and Yoram Singer. 2003. Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. In *Proceedings of HLT-NAACL* 252-259.

Y. W. Wong and R. J. Mooney. 2006. Learning for semantic parsing with statistical machine translation. In *In Proceedings of HLT-NAACL-06*, pages 439-446, New York City, NY.