Leveraging Tokens in a Natural Language Query for NLIDB Systems

Thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science by Research

in

Computational Linguistics

by

Ashish Palakurthi
201125156
ashish.palakurthi@research.iiit.ac.in

International Institute of Information Technology
Hyderabad - 500 032, INDIA
April 2017
International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled “Leveraging Tokens in Natural Language Query for NLIDB Systems” by Ashish Palakurthi has been carried out under my supervision and is not submitted elsewhere for a degree.

Date

Adviser: Dr. Radhika Mamidi
To My Parents
Acknowledgments

I owe my gratitude to all those people who have made this work possible. My deepest gratitude is to my advisor, Dr. Radhika Mamidi. I have been amazingly fortunate to have an advisor who gave me the freedom to explore on my own and at the same time the guidance to recover when my steps faltered. Her patience and support helped me overcome many crisis situations and finish my thesis.

I am thankful to my professors Dipti Misra Sharma, Manish Shrivastava and Soma Paul from whom I have learned a lot.

I am greatly indebted to Arjun R. Akula who has constantly supported me. Throughout this work, he provided thorough guidance and motivation. The detailed discussions and his constructive insights were very helpful in guiding my thoughts.

I am thankful to Ruthu S.M for supporting me during the initial stages of my research. Her inputs aided me in getting the early strides going forward.

I would also like to thank Sai Krishna Srirampur, Ravi Chandibhamar and Riyaz Ahmad Bhatt. Most importantly, I am thankful to my parents and grandparents for supporting me. I am sure I forgot to acknowledge many people who contributed vastly to the completion of this thesis; thank you all.
Abstract

Natural Language Interface to Database (NLIDB) systems convert a Natural Language (NL) query into a Structured Query Language (SQL) and then use the SQL query to retrieve information from a database. The main advantage with this function of an NLIDB system is that it makes information retrieval much easier and more importantly, it also allows non-technical users to query a database.

In this work, we present an effective usage of tokens in an NL query to address various problems in an NLIDB system. The conversion of an NL query to SQL query is framed as a token classification problem, using which we unveil a novel method of mapping an NL query to an SQL query. Our approach reasonably addresses domain dependency which is one of the major drawbacks of NLIDB systems.

Concepts Identification is a major component of NLIDB systems (Gupta et al., 2012; Srirampur et al., 2014). We improve Concepts Identification (Srirampur et al., 2014) by making use of Stanford Parser Dependencies. Our approach is more robust than previously proposed methods. In addition, we also discuss how Concepts Identification can be applied to address Ellipsis in a dialogue process.

In addition to providing results to a user, it is essential to provide a relevant and a compact set of results. We propose a new problem of generating a compact set of results to a user query. At a higher level, user-system interactions are modeled based on patterns frequently observed between a user’s current query and his previous queries, while interacting with a system. Using these models, we propose a novel method of system prompting to help a user obtain a smaller and a relevant set of results.

In addition to providing a compact and a relevant set of results, it is imperative that answers of an NLIDB system are comprehensible even by the people who are less familiar with a common language like English. NLIDB systems use Natural Language Generation (NLG) modules to provide answers in the form of sentences. It is important to make sure that an answer generated by an NLG module is very simple to understand. This brings in the problem of text simplification, wherein, one of the most crucial and initial sub-problems is Complex Word Identification (CWI). We address the problem of CWI by distinguishing words as simple and complex. A plethora of classifiers were explored to identify the complex words. This information helps us in improving the simplification of an NLIDB system’s final output to a user.

To summarize, in addition to addressing problems within an NLIDB system, this work touches post-NLIDB problems like results processing. All the proposed issues are tackled using tokens in an NL query as the basic and a driving unit of force.
Contents

Chapter                        Page

1 Introduction                  1
   1.1 Motivation                 2
   1.2 History of NLIDB Systems   3
   1.3 Challenges in Dialogue and NLIDB Systems  5
   1.4 Contributions of this Thesis  6
   1.5 Thesis Organization       7

2 Classification of Attributes in a Natural Language Query into Different SQL Clauses  9
   2.1 Introduction              9
   2.2 Related Work              10
   2.3 Problem                  10
   2.4 Explicit Attribute Classification  11
   2.5 Methodology              13
      2.5.1 Classification Features  14
      2.5.2 Completing the SQL Query  15
   2.6 Experiments and Discussions  16
      2.6.1 Data                  16
      2.6.2 Experimental Results  17
         2.6.2.1 Baseline Method   17
         2.6.2.2 Conditional Random Fields  17
      2.6.3 Error Analysis in Attribute Classification  20
   2.7 Conclusion and Future Work  21

3 Concepts Identification  22
   3.1 Handling Ellipsis in a Dialogue Process  26
   3.2 Our Solutions              30
   3.3 Conclusions                31

4 Towards Producing Compact Results in User-System Interactions for NLIDB System  32
   4.1 Introduction             32
   4.2 Specificity of a User Query  33
   4.3 Modeling User-System Interactions  34
      4.3.1 Prompting the User Hierarchically  35
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.4 Reducing results without prompting a user</td>
<td>37</td>
</tr>
<tr>
<td>4.5 Experiments and Analysis</td>
<td>37</td>
</tr>
<tr>
<td>4.6 Conclusion</td>
<td>38</td>
</tr>
<tr>
<td>5 Complex Word Identification in an NL query</td>
<td>40</td>
</tr>
<tr>
<td>5.1 System Description</td>
<td>42</td>
</tr>
<tr>
<td>5.2 Experiments and Discussions</td>
<td>43</td>
</tr>
<tr>
<td>5.2.1 Data</td>
<td>43</td>
</tr>
<tr>
<td>5.2.2 Discussions</td>
<td>43</td>
</tr>
<tr>
<td>5.2.3 Evaluation Metric</td>
<td>45</td>
</tr>
<tr>
<td>5.2.4 Results</td>
<td>45</td>
</tr>
<tr>
<td>5.3 Complex Word Identification for NLIDB</td>
<td>47</td>
</tr>
<tr>
<td>5.3.1 How else can Lexical Simplification help NLIDB ?</td>
<td>50</td>
</tr>
<tr>
<td>5.4 Conclusion and Future Work</td>
<td>50</td>
</tr>
<tr>
<td>6 Evolution of the NLIDB</td>
<td>51</td>
</tr>
<tr>
<td>7 Future Directions</td>
<td>57</td>
</tr>
<tr>
<td>Appendix A: Examples of Complex Words</td>
<td>59</td>
</tr>
<tr>
<td>Appendix B: Examples of User queries for Hierarchical and Non-Hierarchical Models</td>
<td>60</td>
</tr>
<tr>
<td>Bibliography</td>
<td>63</td>
</tr>
</tbody>
</table>
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Working Flow of an NLIDB System</td>
<td>2</td>
</tr>
<tr>
<td>5.1</td>
<td>Lexical Simplification Pipeline Image reproduced from [37]</td>
<td>40</td>
</tr>
<tr>
<td>6.1</td>
<td>NLIDB-2012</td>
<td>51</td>
</tr>
<tr>
<td>6.2</td>
<td>NLIDB-2013</td>
<td>51</td>
</tr>
<tr>
<td>6.3</td>
<td>NLIDB-2013</td>
<td>52</td>
</tr>
<tr>
<td>6.4</td>
<td>NLIDB-2014</td>
<td>52</td>
</tr>
<tr>
<td>6.5</td>
<td>NLIDB-2014</td>
<td>53</td>
</tr>
<tr>
<td>6.6</td>
<td>NLIDB-2015</td>
<td>53</td>
</tr>
<tr>
<td>6.7</td>
<td>NLIDB-2015</td>
<td>54</td>
</tr>
<tr>
<td>6.8</td>
<td>NLIDB-2015</td>
<td>54</td>
</tr>
<tr>
<td>6.9</td>
<td>NLIDB-2016</td>
<td>55</td>
</tr>
</tbody>
</table>
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Sample domain dictionary</td>
</tr>
<tr>
<td>2.2</td>
<td>Example of tagging scheme</td>
</tr>
<tr>
<td>2.3</td>
<td>Corpus statistics</td>
</tr>
<tr>
<td>2.4</td>
<td>Baseline method results</td>
</tr>
<tr>
<td>2.5</td>
<td>Results obtained without considering contextual features</td>
</tr>
<tr>
<td>2.6</td>
<td>Results obtained on adding contextual features</td>
</tr>
<tr>
<td>2.7</td>
<td>Cross domain results</td>
</tr>
<tr>
<td>3.1</td>
<td>Semantic frame for teach</td>
</tr>
<tr>
<td>3.2</td>
<td>Semantic frame for register</td>
</tr>
<tr>
<td>3.3</td>
<td>Concepts tagging example</td>
</tr>
<tr>
<td>3.4</td>
<td>Concepts Identification Results with old features</td>
</tr>
<tr>
<td>3.5</td>
<td>Concepts Identification Results with additional features</td>
</tr>
<tr>
<td>3.6</td>
<td>Concepts Identification Results on new data without additional features</td>
</tr>
<tr>
<td>3.7</td>
<td>Concepts Identification Results on new data with additional features</td>
</tr>
<tr>
<td>4.1</td>
<td>Concepts Identification</td>
</tr>
<tr>
<td>5.1</td>
<td>Cross-validation results on Training Data</td>
</tr>
<tr>
<td>5.2</td>
<td>Results on Test Data</td>
</tr>
<tr>
<td>5.3</td>
<td>Results on Test Data using additional features</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

A conversation between two humans or between a human and a machine is known as a dialogue. A dialogue system [18], [60], [25], [11], [41], [46], [28] is a system which interacts with humans in natural language, similar to the way in which humans interact with each other. There has been a lot of active research going in the field of dialogue systems as they pose multiple challenges. Typically, a dialogue system consists of six components [4], which are as follows:

1. **Input Decoder**: This module recognizes the input and converts it into text. For eg. speech to text.

2. **Natural Language Understanding**: The input query is processed through various natural language techniques like Parts Of Speech tagging, Named Entity Recognition, Morph analysis etc. This module provides some useful knowledge about the query to the dialogue manager.

3. **Dialogue Manager**: The dialogue manager carries multiple functions and is said to be the core part of the dialogue system. It communicates with various task managers in the process of generating answers. It maintains dialogue history, handles linguistic issues and decides the best response for the user.

4. **Domain Specific Component**: A dialogue system often needs to interact with external interfaces to access something like a database that would provide the main system with answers for a user query. One such interface is the Natural Language Interface to Database.

5. **Response Generator**: This module gives the output that is to be presented to the user. It could be generated based on templates or by using natural language generation techniques.

6. **Output Renderer**: Converts text output of the previous module to speech and presents it to the user.

Natural Language Interface to Database (NLIDB) system [3], [1], [13], [14], [8], [22], [20] is a system which takes a natural language query as an input, processes it and then gives the required results. To fetch results from a database, a formal language is required. To fulfill this requirement, an NLIDB system converts the natural language query into a formal language query (for eg., SQL) and then uses
this SQL query to produce appropriate data for the given input query. Figure 1.1 shows the functioning flow of a typical NLIDB system. UI is the user interface which takes the input natural language query. The Mapper module maps the NL query to an SQL query. The SQL query is applied to a database. This generates appropriate results from the database. These results are returned back to the user through the UI.

Figure 1.1 Working Flow of an NLIDB System

1.1 Motivation

One of the most important aims of Artificial Intelligence is to make things easily and quickly accessible for humans. The access to information is invaluable and it should be available for everyone. However, one needs to have the knowledge regarding formal languages to access information from current systems and this hinders non-technical people from obtaining the information they want. It is crucial for systems to be user-friendly in order to obtain the highest benefits. NLIDB systems try to make information accessible to everyone who knows a natural language. The aim of NLIDB systems is to break the barriers for non-technical users and make information easily accessible. However, given the ambiguity and noise found in natural language, NLIDB systems still need a lot of improvements to make them more accurate and beneficial to common users. NLIDB systems can come in handy in many ways like:

1. Help Desk: NLIDBs can be useful at help desks in different domains. For eg.,
   - A restaurant: A customer may want to query about the dishes in the restaurant.
   - A university: A person might need some information about the availability of seats in each branch.
   - An airport: A tourist might need some information about flights.
2. Service: Responding to customers’ questions about services. Many current day services like Foodpanda, Commut. Co use humans for chatting with customers. However, it is not possible to do this round the clock. Since each service only accounts to questions related to their own, this is where NLIDB systems can be best applicable. A few current commercial applications where NLIDB systems can be of supreme use for personalized service:

- Haptik\(^1\): It is a chat-bot which handles questions in a limited number of different domains like Movies, Travel, Restaurants, Shopping, Recharge, etc. Its prime service is answering user queries and helping people in getting things done quickly.
- Super Genie\(^2\): This is a chat application that is focused on making ordering food, booking hotels, paying bills, buying groceries as easy as possible for users. Users can access the service just at a distance of a ping.
- NikiAi\(^3\): Niki is an Artificial Intelligence powered personal digital assistant that is set out to simplify ordering experience. It works through a simple chat interface. Again, this service is restricted to questions pertaining to ordering online.

3. Education: Students can be provided with problem solving advice and be guided with suggestions.

4. Call Centers: Many dialogue systems are used in call centers to decrease human workload.

One may argue that question answering (QA) systems do the same task of answering questions and providing results to a user. However, QA systems are more generalized in terms of retrieving answers and less accurate when it comes to handling questions in a single domain. NLIDB systems use domain and structured knowledge to answer user queries, and thus perform better on restricted domains.

1.2 History of NLIDB Systems

NLIDB systems have been active since 1970s and it still remains as an open research problem. As mentioned earlier, there are multiple applications that can find NLIDBs useful. From mobile applications to Personal Digital Assistants (PDAs), NLIDBs provide supreme advantages over filling a template or a form which can be tiring to a user. Easy usage is another advantage. It is easy because the user only needs to type in a query and not fill multiple fields. Moreover, the user is not constrained from adding any information he wishes.

The Lunar Sciences Natural Language Information System \[59\] came with an idea that it is better if machines can adapt human language rather than humans learning new languages to gain information. Lunar is a research system where the given query is processed through a module for the best parsings of a sentence and then uses various semantic heuristics to produce a formal language query using which

---

\(^1\)http://www.haptik.co/
\(^2\)http://supergenieapp.com/
\(^3\)https://niki.ai/
information is retrieved from the database. However, the LUNAR system is said to be deprived of flexibility issues in terms of handling a variety of queries. This is mainly because of its architecture that was built particularly for the NASA project. The LADDER system [24] uses semantic grammar to parse the input natural language query and map it to a formal language query. This system was developed to answer questions related to the US Navy ships. This system could be fabricated to be interfaced with different database management systems. PLANES [55] is an example of a semantic grammar system where user inputs are translated into semantic tree, which is further mapped to a formal language. The PLANES system also handles problems like ellipsis and semi-grammatical queries.

PRECISE [43], developed by the University of Washington, uses a heuristic based approach to track semantically tractable sentences which are unique semantic representations based on the designed constraints. They convert NL query to an SQL query. Being rule-based, the system cannot address all types of queries. WASP [57], developed by the University of Texas, learns a semantic parser given parallel mappings of natural language and formal language. WASP uses statistical machine translation techniques for learning the mappings. However, the under usage of Database knowledge of a given domain prevents WASP from performing better.

Chat-80[56] is a system which uses grammar formalisms to convert natural language queries into a formal language called Prolog which can be used to get answers from a database. Unlike many other NLIDBs, Chat-80 also focuses on optimizing the formal language query before it is applied to a database. Different from many systems, ASK [54] is one system in which there is a special dialogue which can be used to design system guided dialogues to accomplish tasks of a user. The user aids the computer to design a complete query. ASK also features interacting with multiple databases.

A few commercially available NLIDBs developed in the early 90s were INTELLECT [23], RUS [6], DATATALKER [8] and English Wizard from Linguistic Technology Corporation.

Many modern approaches have also been explored in building NLIDB systems. Advanced machine learning methods [14] like Support Vector Machines along with kernel methods were used to encode relations between parse trees of natural language and parse tree of formal language to develop NLIDB systems that are capable of handling advanced queries as well. Algorithms [16] were formed using syntactic dependencies, wherein various pieces of SQL clause information were combined to form a complete SQL query. On recursively applying the algorithms, nested SQL queries could be formed.

An Ontology based approach [35] was devised wherein mapping between natural language and formal language query happens using ontology concepts. The authors also claim that the approach is both domain and language independent.

Paninian NLIDB [20], developed by IIIT Hyderabad, is a system that uses the Computational Paninian Grammar Framework. The system processes the natural language query through three stages namely, syntactic, semantic and graph processing stage. They finally convert the NL query into an SQL query. The syntactic stage maps Stanford dependencies into Karakas and this information is further used in the semantic stage. The semantic stage involves Concepts Identification and the output of the semantic stage is used by the graph processing stage wherein the system finally produces a graph to generate an
SQL query. In this work, we try to improve the Paninian NLIDB system by focusing on improving the performance of its components.

1.3 Challenges in Dialogue and NLIDB Systems

Current Dialogue systems and NLIDB systems face multiple challenges [17], [2]. Few of them are:

1. Evaluation: One of the toughest challenges in dialogue systems is the evaluation of progress. Different evaluation criteria have been introduced for this task, but no approach still guarantees a reliable metric for evaluating dialogue systems. Commonly used techniques to evaluate dialogue systems are:

   - Sub-components: Improvement in the performance of each sub-component of a dialogue system can improve the performance of the entire system. A reduction in errors of one module would help a lot in overall improvement, as lesser number of errors would propagate across other modules.

   - Task completion: In case a dialogue system addresses transactions for users, it is possible to know if the transaction has been completed successfully. If the transaction has been completed successfully, then probably the interaction with the user was helpful and hence one may conclude it to be successful.

   - Statistics: A few statistics like time taken for dialogue completion, number of turns etc. may be considered. However, these statistics are generally not as useful as user satisfaction. A better evaluation metric is to observe if the users are actually re-using the system or recommending it to someone else.

2. Time taken to generate the final output: As mentioned above, it is important that a system does not take too much time in generating results. A system may generally take more time for the following reasons:

   - A module or a sub-module may consume time in sending its output to the next module.

   - Overall, a system may find it difficult to interpret the user’s needs, as the user may not be satisfied with the results generated and may thus re-prompt the user with some more details. This would result in continuation of the interaction between the user and the system and hence consume more time. A user may not be satisfied with the results given because of many reasons. A few of them are:

      - Quality: It is important that the results presented are useful, relevant and comprehensible for the user.

      - Relevance: Results should be relevant to the user’s query.

      - Usefulness: A user should gain maximum benefit from the results provided to him.
* Comprehensible: The user should understand the output of the system.
  
  – Quantity: It is important that the results are compact, wherein large size of results would make it time-consuming for the user to pick the ones relevant to him.

  - Complexity in functioning of sub-components: It is imperative that complexities of algorithms for various components in an NLIDB are improved while they simultaneously use lesser resources such as features incorporated for learning various hypotheses.

3. Driving a dialogue [1], [31] depends on a user. The capabilities of the system, the problem that the system is focusing on, the time to generate output and many other factors. Deciding the right trade-off between different factors is difficult.

4. Portability: It is important to design systems which are modularized and usable for different domains and languages. The Paninian NLIDB [2] is one such system which provides domain and language portability. A dialogue system or its domain specific component needs to be robust in domain specific elements, which is essential for being effective in real-time applications. Portability is a serious ongoing challenge that is being worked on.

5. Apart from this, other problems like natural language ambiguity, ellipsis and noise in sentences pose additional challenges.

1.4 Contributions of this Thesis

The major contributions of this thesis are:

1. We present a novel approach to map an NL query to an SQL query.
   - We classify attributes present in an NL query into different SQL clauses.
   - We show a new application of Concepts Identification [52] in completing an SQL query.
   - We show that the attribute classification is helpful in reasonably addressing the domain independence problem found in NLIDB systems.

2. We use Stanford dependencies to improve Concepts Identification. We ensemble the theories of two previously proposed approaches [52], [20] and show that our method performs better. We re-conduct experiments using a finer version of the previously proposed tagset.

3. We propose a new solution to generate compact set of results to a user query. At a high level, user-system interactions are modeled based on patterns frequently observed between a user’s current query and his previous queries, while interacting with a system. Using these models, we propose a novel method of system prompting to help a user obtain a smaller and a relevant set of results.
4. We address the problem of Complex Word Identification by distinguishing words as simple and complex. Various classifiers were explored to identify the complex words. This information helps us in improving the simplification of an NLIDB system’s final output to a user.

5. We present a re-engineered version of the NLIDB system proposed by Gupta et al., 2012. In particular, we show how the CPG NLIDB, proposed in 2012, has been developed since then.

6. We release the Concepts Identification System and the data related to it.

7. We release the dialogue data collected for our experiments of Chapter-4.

Problems in Natural Language Interface to Databases are categorized [36] into two modules:

- Database Module: Mapping words in natural language of the input to formal objects in the database.
- Linguistic Module: Translates natural language into formal language.

In addition to addressing both the above modules, we also touch post NLIDB issues.

1.5 Thesis Organization

The remainder of this thesis is organized as follows:

- Chapter 2: *Classification of Attributes in a Natural Language Query into Different SQL Clauses*: This chapter explains our approach to classify attributes in a natural language query into different SQL clauses. This helps us in building a partial SQL query. We complete the SQL query using existing methods. The procedure of mapping the natural language query to an SQL query is explained in detail. We also discuss our error analysis.

- Chapter 3: *Concepts Identification*: This chapter explains a hybrid usage of previously proposed techniques to identify concepts of a natural language query. We discuss the utility of our approach for identifying the concepts.

- Chapter 4: *Towards Producing Compact Results in User-System Interactions for NLIDB System*: In this chapter, we propose our method for reducing the size of results by making a user’s initial query more specific. This is done through system-prompting.

- Chapter 5: *Complex Word Identification (CWI)*: Complex word identification is a crucial step in the process of lexical simplification of a given text. We frame CWI as a classification problem and identify complex words. We discuss our experiments and analysis in detail.

- Chapter 6: *The NLIDB Timeline*: In this chapter, we take a look back at how the NLIDB system at LTRC, IIIT-H, has improved since 2012.
• Chapter 7: We discuss the future direction related to each chapter discussed in this thesis.

Appendix

• The sample examples related to Chapter 2 are shown in Appendix A.

• The sample examples related to Chapter 3 are shown in Appendix B.

• The sample examples related to Chapter 4 are shown in Appendix C.

• The sample examples related to Chapter 5 are shown in Appendix D.
Chapter 2

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

2.1 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) [3], [22], [45], [44], [13], [20] 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 formal language query, for eg., an 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 information from an NL query before converting it into an 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. An SQL query is composed of different SQL clauses like SELECT, FROM, WHERE, GROUP BY, HAVING and ORDER BY. Since clauses in an SQL query have attributes, attribute information becomes very important for an effective conversion of an NL query into an SQL query. Explicit attributes are the attributes mentioned by the user in the NL query text. When an NL query is converted into SQL query, explicit attributes may belong to different SQL clauses. In this work, we use Conditional Random Fields (CRF) for classifying the explicit attributes in an NL query to different SQL clauses.
2.2 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 [20], the authors propose an NLIDB system based on Computational Paninian Grammar (CPG) Framework [5]. They emphasize on syntactic elements as well as the semantics of the domain. They convert an NL query into an 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 large number of frames in large domains. In [30], machine learning was used in Question Answering systems. 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. Ammany Sarhan [48] emphasized on the importance of identifying the table names of attributes in an 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 an NL query to different SQL clauses in their SQL queries. In [52], the authors address the problem of Concepts Identification of an NL query in NLIDB, which plays a crucial role for our system to generate a complete SQL query.

The remainder of this chapter 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 generating the complete SQL query. In section 6, we show experimentations and results along with error analysis. We conclude in section 7.

2.3 Problem

An attribute in an NL query can correspond to various SQL clauses. We define two types of attributes which can be found in an 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 2.1 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 an SQL query as they are associated with a value. This work focuses on classifying explicit attributes into different SQL clauses in an SQL query.

<table>
<thead>
<tr>
<th>Value</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>professor_name</td>
</tr>
<tr>
<td>ABCD</td>
<td>lab_name</td>
</tr>
<tr>
<td>John</td>
<td>student_name</td>
</tr>
<tr>
<td>Science</td>
<td>course_name</td>
</tr>
</tbody>
</table>

Table 2.1 Sample domain dictionary

2.4 Explicit Attribute Classification

In this section, we illustrate the classification of explicit attributes from an NL query into different clauses in an 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 (below), course name belongs to the SELECT clause as the user has asked to list courses taught by Smith. In example 2 (below), 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.

**Example 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.
```

**Example 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
```

**Example 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
```

**Example 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
```
2.5 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 an 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 an 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.

<table>
<thead>
<tr>
<th>Token</th>
<th>Attribute</th>
<th>Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>What</td>
<td>0</td>
<td>O</td>
</tr>
<tr>
<td>are</td>
<td>0</td>
<td>O</td>
</tr>
<tr>
<td>the</td>
<td>0</td>
<td>O</td>
</tr>
<tr>
<td>courses</td>
<td>1</td>
<td>GROUP BY</td>
</tr>
<tr>
<td>with</td>
<td>0</td>
<td>O</td>
</tr>
<tr>
<td>less</td>
<td>0</td>
<td>O</td>
</tr>
<tr>
<td>than</td>
<td>0</td>
<td>O</td>
</tr>
<tr>
<td>25</td>
<td>0</td>
<td>O</td>
</tr>
<tr>
<td>students</td>
<td>1</td>
<td>HAVING</td>
</tr>
<tr>
<td>?</td>
<td>0</td>
<td>O</td>
</tr>
</tbody>
</table>

Table 2.2 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 2.2 shows an example of the tagging scheme. Each token in a sentence is given a set of features and a tag. In Table 2.2, we have shown only one feature due to space constraints. We trained our data and created models for testing. We used Conditional Random Fields [32] for the machine learning task. The next subsection describes the features employed for the classification of explicit attributes in an NL query.
2.5.1 Classification Features

The following features were used for the classification of explicit attributes in an 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\(^1\) [10].

**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 an 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).

\(^1\)http://nlp.stanford.edu/software/lex-parser.shtml
2.5.2 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\(^2\) of all the tokens in the given query.
3. Mapping of identified entities and relationships in the Entity Relationship schema to get the join 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 an SQL query. That is, there can be attributes other than explicit attributes and implicit attributes in an 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 attributes, 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.

\(^2\)Concept of an NL token maps the NL token to the database schema. The entities, attributes and relationships in the database schema constitute concepts.
All the identified entities and relationships can now be mapped to the Entity Relationship (ER) schema to get the join conditions [2] 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 [2] discusses the problem of handling join 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.

2.6 Experiments and Discussions

2.6.1 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\(^3\). We considered the Restaurant and the Geoquery domains [58] 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 2.3 shows the number of sentences considered for training and testing in each domain.

<table>
<thead>
<tr>
<th>Domain</th>
<th>Train</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Academic</td>
<td>711</td>
<td>305</td>
</tr>
<tr>
<td>Restaurant</td>
<td>150</td>
<td>100</td>
</tr>
<tr>
<td>Geoquery</td>
<td>400</td>
<td>224</td>
</tr>
</tbody>
</table>

Table 2.3 Corpus statistics

\(^3\)http://www.cs.utexas.edu/users/ml/nlda.html
2.6.2 Experimental Results

We used the metrics of Precision (P), Recall (R) and F-measure\(^4\) (F) for evaluation.

2.6.2.1 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.

<table>
<thead>
<tr>
<th>Domain</th>
<th>P(%)</th>
<th>R(%)</th>
<th>F(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Academic</td>
<td>46.29</td>
<td>46.37</td>
<td>46.33</td>
</tr>
<tr>
<td>Restaurant</td>
<td>47.75</td>
<td>43.80</td>
<td>45.69</td>
</tr>
<tr>
<td>Geoquery</td>
<td>63.08</td>
<td>63.08</td>
<td>63.08</td>
</tr>
</tbody>
</table>

Table 2.4 Baseline method results

2.6.2.2 Conditional Random Fields

We used Conditional Random Fields for the classification experiments since it represents the state 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++\(^5\) 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%.

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

\[ F - \text{measure} = \frac{2 \times P \times R}{P + R} \]

\(^4\)https://code.google.com/p/crfpp
### Table 2.5 Results obtained without considering contextual features.

<table>
<thead>
<tr>
<th>Domain</th>
<th>Clause</th>
<th>P(%)</th>
<th>R(%)</th>
<th>F(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Academic</td>
<td>SELECT</td>
<td>60.94</td>
<td>89.80</td>
<td>72.61</td>
</tr>
<tr>
<td></td>
<td>WHERE</td>
<td>48.81</td>
<td>57.75</td>
<td>52.90</td>
</tr>
<tr>
<td></td>
<td>GROUP BY</td>
<td>72.37</td>
<td>38.73</td>
<td>50.46</td>
</tr>
<tr>
<td></td>
<td>HAVING</td>
<td>14.29</td>
<td>1.52</td>
<td>2.74</td>
</tr>
<tr>
<td>Overall</td>
<td>SELECT</td>
<td>60.04</td>
<td>60.50</td>
<td>60.27</td>
</tr>
<tr>
<td></td>
<td>WHERE</td>
<td>72.61</td>
<td>57.75</td>
<td>60.27</td>
</tr>
<tr>
<td></td>
<td>HAVING</td>
<td>15.29</td>
<td>1.52</td>
<td>2.74</td>
</tr>
<tr>
<td></td>
<td>Overall</td>
<td>60.94</td>
<td>60.50</td>
<td>60.27</td>
</tr>
</tbody>
</table>

### Table 2.6 Results obtained on adding contextual features.

<table>
<thead>
<tr>
<th>Domain</th>
<th>Clause</th>
<th>P(%)</th>
<th>R(%)</th>
<th>F(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Academic</td>
<td>SELECT</td>
<td>88.12</td>
<td>93.88</td>
<td>90.91</td>
</tr>
<tr>
<td></td>
<td>WHERE</td>
<td>56.41</td>
<td>92.96</td>
<td>70.21</td>
</tr>
<tr>
<td></td>
<td>GROUP BY</td>
<td>96.58</td>
<td>79.58</td>
<td>87.26</td>
</tr>
<tr>
<td></td>
<td>HAVING</td>
<td>96.88</td>
<td>46.97</td>
<td>63.27</td>
</tr>
<tr>
<td>Overall</td>
<td>SELECT</td>
<td>83.49</td>
<td>83.97</td>
<td>83.73</td>
</tr>
<tr>
<td></td>
<td>WHERE</td>
<td>94.64</td>
<td>81.54</td>
<td>87.60</td>
</tr>
<tr>
<td></td>
<td>HAVING</td>
<td>97.30</td>
<td>89.26</td>
<td>93.10</td>
</tr>
<tr>
<td>Geoquery</td>
<td>SELECT</td>
<td>94.04</td>
<td>99.02</td>
<td>93.76</td>
</tr>
<tr>
<td></td>
<td>WHERE</td>
<td>97.94</td>
<td>79.17</td>
<td>87.56</td>
</tr>
<tr>
<td>Overall</td>
<td>SELECT</td>
<td>91.69</td>
<td>91.69</td>
<td>91.69</td>
</tr>
</tbody>
</table>

an attribute is important in deciding its SQL clause. Table 2.5 and Table 2.6 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 2.5 and Table 2.6, 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.

<table>
<thead>
<tr>
<th>Train</th>
<th>Test</th>
<th>P(%)</th>
<th>R(%)</th>
<th>F(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Academic</td>
<td>Restaurant</td>
<td>69.63</td>
<td>77.69</td>
<td>73.44</td>
</tr>
<tr>
<td>Academic</td>
<td>Geoquery</td>
<td>70.99</td>
<td>70.77</td>
<td>70.88</td>
</tr>
<tr>
<td>Restaurant</td>
<td>Academic</td>
<td>52.55</td>
<td>51.15</td>
<td>51.84</td>
</tr>
<tr>
<td>Restaurant</td>
<td>Geoquery</td>
<td>80.66</td>
<td>60.31</td>
<td>69.01</td>
</tr>
<tr>
<td>Geoquery</td>
<td>Academic</td>
<td>50.57</td>
<td>50.95</td>
<td>50.76</td>
</tr>
<tr>
<td>Geoquery</td>
<td>Restaurant</td>
<td>77.78</td>
<td>86.78</td>
<td>82.03</td>
</tr>
</tbody>
</table>

Table 2.7 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 2.7 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 [43] 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 [29], 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 [15] 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 [16] 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 [7] 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 an SQL query, we focus on attributes to
build an SQL query. After attribute classification, Concept Identification and identification of the join 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\(^6\) and Academic domains respectively.

We define accuracy as

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

These accuracies\(^7\) are on the same test datasets used for attribute classification(Table 2.3). 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.

### 2.6.3 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.**

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_

\(^6\)The results on Geoquery domain may actually be worse as the data (Table 2.3) we considered is a subset of the GEO880.

\(^7\)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.
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.

2.7 Conclusion and Future Work

In this work, we investigate a CRF model to classify attributes present in an NL query into different SQL clauses in an 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 chapter are:

- We show that within a sentence, attributes can be used to build an 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 an NLIDB system. Exploring patterns on how these attributes are used by a user in an NL query can be useful to form an 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.
Chapter 3

Concepts Identification

Concepts Identification\(^1\) is a technique where tokens in a natural language query are mapped to elements of a domain. [20] store multiple concepts for each verb based on different Karaka roles identified in a query, for the semantic stage in their approach towards mapping natural language queries to SQL queries. Taking an example [20], consider

**Which professor took NLP?**

<table>
<thead>
<tr>
<th>Karaka Role</th>
<th>Token</th>
<th>Concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>root</td>
<td>teach</td>
<td>teach</td>
</tr>
<tr>
<td>K1</td>
<td>Who</td>
<td>professor.name</td>
</tr>
<tr>
<td>K2</td>
<td>NLP</td>
<td>course.name</td>
</tr>
</tbody>
</table>

**Table 3.1 Semantic frame for teach**

**Which student took NLP?**

<table>
<thead>
<tr>
<th>Karaka Role</th>
<th>Token</th>
<th>Concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>root</td>
<td>register</td>
<td>register</td>
</tr>
<tr>
<td>K1</td>
<td>Who</td>
<td>student.name</td>
</tr>
<tr>
<td>K2</td>
<td>NLP</td>
<td>course.name</td>
</tr>
</tbody>
</table>

**Table 3.2 Semantic frame for register**

The above tables convey the following:

\(^1\)The reader is advised to read our paper [52] which was a collaborated work between Ashish Palakurthi and Saikrishna Srirampur. To avoid redundancy, we do not repeat most of the content in this paper as the first author has talked about it in his thesis. The work discussed in this chapter improves the work discussed by Saikrishna Srirampur. Uncompromising care has been taken to ensure that there is NO repeat of content in this chapter with regards to Concepts Identification previously discussed by Saikrishna Srirampur.
• For the verb *teach* in the first sentence, the concept to be assigned to K1 (Who) is professor.name and the concept to be assigned to K2 (NLP) is course.name.

• For the verb *register* in the second sentence, the concept to be assigned to K1 (Who) is student.name and the concept to be assigned to K2 (NLP) is course.name.

However, the problem with this [20] approach is that semantic verb frames as shown in Table 3.1 and Table 3.2 need to be stored in advance to addressing a user query. This approach poses the following problems:

• *Verb frames:* It is not possible to predict all the verbs beforehand and create semantic frames for them. This approach would fail if a new verb is encountered.

• *High dependency on Karaka roles:* The output of the SQL query is strongly dependent on the Karaka labels. Wrong Karaka labels could trigger undesirable issues.

• *Manual effort:* The semantic frames require manual effort to be created.

Addressing these issues, [52] came with a data-driven machine learning approach to identify concepts in a natural language query. Each token in the query was given a set of features like POS tags, alpha-numeric features, length of tokens etc., and then a sequence learning algorithm was applied to predict a token’s concept. Sequence learning was applied for effectively capturing contextual features.

The following is an example from [52] illustrating the assignment of concepts to each token in a natural language query,

*List UG3 students who registered for Computational Linguistics in 2012?*

<table>
<thead>
<tr>
<th>Token</th>
<th>Concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>List</td>
<td>B-User.name</td>
</tr>
<tr>
<td>UG3 students</td>
<td>B-User.batch</td>
</tr>
<tr>
<td>who</td>
<td>O</td>
</tr>
<tr>
<td>registered</td>
<td>O</td>
</tr>
<tr>
<td>for</td>
<td>O</td>
</tr>
<tr>
<td>Computational</td>
<td>B-register</td>
</tr>
<tr>
<td>Linguistics</td>
<td>O</td>
</tr>
<tr>
<td>in</td>
<td>B-Courses.name</td>
</tr>
<tr>
<td>2012</td>
<td>I-Courses.name</td>
</tr>
<tr>
<td>?</td>
<td>O</td>
</tr>
<tr>
<td></td>
<td>B-Semester.year</td>
</tr>
</tbody>
</table>

| Table 3.3 Concepts tagging example |
The usage of novel set of features along with context helps in tackling new verbs, new values and also requires comparatively lesser human effort. However, they[52] do not capture few essential semantics which were claimed to be important by [20]. It is important to note that context is not always helpful to identify the concept of a token. This is the reason why [20] use karaka labels. It would be ideal to blend [52] along with karakas for an effective concepts identification. However, the absence of a statistical and a reliable model that gives karakas at run time is not available. We thus go forward and blend the statistical concept identification module [52] with close cousins of karakas which are the Stanford dependency relations.

The below is an example, where [52] would fail to capture the concepts correctly.

*List all the UG3 students who registered for Computational Linguistics in 2012.*

Though *List* is tagged as B-User.name in the above table (Table 3.3), *List* cannot be tagged B-User.name in the above mentioned example where the token *students* is out of the context window (size=3). This is because [52] only identify concepts using context but syntactic relations are not considered. Note that context will not be helpful always if the critical information lies outside the window considered. Considering larger window will likely over fit the learning process and lead to lesser generalization in performance. Thus to overcome this problem we use Stanford dependencies to identify what the question words and other words like *list, show, tell* are tagged according to what the user is asking irrespective where the target token of the required concept lies in the sentence.

*List all the UG3 students who registered for Computational Linguistics in 2012.*

Stanford dependencies root(ROOT-0, List-1)
det:predet(students-5, all-2)
det(students-5, the-3)
compound(students-5, UG3-4)
**dobj(List-1, students-5)**
nsubj(registered-7, students-5)
ref(students-5, who-6)
acl:relcl(students-5, registered-7)
case(Linguistics-10, for-8)
compound(Linguistics-10, Computational-9)
nmod:for(registered-7, Linguistics-10)
case(2012-12, in-11)
nmod:in(Linguistics-10, 2012-12)

*List UG3 students who registered for Computational Linguistics in 2012.*
Stanford dependencies
root(ROOT-0, List-1)
nummod(students-3, UG3-2)
**dobj(List-1, students-3)**
Observe the dependency lines marked in bold. In both the case, irrespective of the distance between the tokens *List* and *students*, the relation dobj handles the problem of assigning the token *students* or the attribute student.name (which here is B-User.name) to question words or words like *List* in the user query. This is what [20] essentially do through karaka relations. But we handle this through Stanford Dependency relations given its statistical training strength and reliability. However, they [20] do not handle concepts related to attribute-value pairs properly, but [52] handle the concepts related to attribute-values (like NLP(value)=course.name(attribute)). We thus hypothesize a combination of both the approaches resulting in more robust performance.

Again consider the following simple queries,

- **Who taught NLP?**
  - nsubj(taught-2, Who-1)
  - root(ROOT-0, taught-2)
  - dobj(taught-2, NLP-3)

- **Who will teach NLP?**
  - nsubj(teach-3, Who-1)
  - aux(teach-3, will-2)
  - root(ROOT-0, teach-3)
  - dobj(teach-3, NLP-4)

- **Who is being taught NLP?**
  - nsubjpass(taught-4, Who-1)
  - aux(taught-4, is-2)
  - auxpass(taught-4, being-3)
  - root(ROOT-0, taught-4)
  - dobj(taught-4, NLP-5)
Who was taught NLP?
- nsubjpass(taught-3, Who-1)
- auxpass(taught-3, was-2)
- root(ROOT-0, taught-3)
- dobj(taught-3, NLP-4)

From the above examples, one can clearly understand the importance of Stanford dependency relations while assigning concepts to question words/tokens in the NL query. In the first two cases, the relation between teach and Who is nsubj whereas the relation between Who and teach in the last two examples is nsubjpass. This states that the token Who in the first two cases has to be given a concept different from the token Who in the last two examples. The correct concept in the first two cases is professor.name and the correct concept in the last two cases is student.name. Note that, we fetch both the relation and the corresponding token as features for learning which handle such cases. However, [52] does not handle such important cases which may look simple but are very deceiving.

In addition to Stanford dependencies, we also use named entities as a feature. This feature addresses concepts like names of professor, student in addition to labs.

We re-run experiments with previous set of features [52]. Table 3.4 shows the results obtained using those features. We then append new features to the previous set and compute the results. These results can be seen in Table 3.5. F-measure has improved from 91.78 to 92.21. The difference may not be significant. This is due to small test data size. In conclusion, the above examples and discussions show that Stanford dependencies are important for Concepts Identification module can be improved by considering them as features.

The tagset proposed by [52] does not distinguish student and professor in its tagging. It assigns both student and professor the same tag (User.name). We make the tagset a bit finer by handling student and professor as different tags. We run our experiments on this new data (with these additional tags) two times (without additional features, with additional features) and compute the results. Table 3.6 shows the results obtained on the new dataset with new tags (in bold in Table 3.6, 3.7) and without additional features. Results in Table 3.7 involve additional features (Stanford dependencies, Named Entities).

It is not surprising to see the results falling down slightly when experimenting with new data. This is because the tag-set is now finer with User.professor and User.student replacing User.name. Also, note that the additional features show a better performance again on the new data when compared to previous set of features.

### 3.1 Handling Ellipsis in a Dialogue Process

Ellipsis is a common problem observed in user-system interactions as users often tend to be less specific while asking questions. Previous work [47] addresses elliptical problems commonly found in
<table>
<thead>
<tr>
<th>Tag</th>
<th>Precision</th>
<th>Recall</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Users.name</td>
<td>92.83</td>
<td>93.45</td>
<td>93.14</td>
</tr>
<tr>
<td>Users.roll</td>
<td>95.00</td>
<td>95.00</td>
<td>95.00</td>
</tr>
<tr>
<td>Users.stream</td>
<td>89.36</td>
<td>93.33</td>
<td>91.30</td>
</tr>
<tr>
<td>Users.batch</td>
<td>91.86</td>
<td>91.86</td>
<td>91.86</td>
</tr>
<tr>
<td>Users.email</td>
<td>100.00</td>
<td>100.00</td>
<td>100.00</td>
</tr>
<tr>
<td>Courses.name</td>
<td>87.43</td>
<td>91.14</td>
<td>89.25</td>
</tr>
<tr>
<td>Courses.code</td>
<td>83.33</td>
<td>100.00</td>
<td>90.91</td>
</tr>
<tr>
<td>Courses.credits</td>
<td>96.15</td>
<td>94.34</td>
<td>95.24</td>
</tr>
<tr>
<td>Courses.type</td>
<td>81.82</td>
<td>72.97</td>
<td>77.14</td>
</tr>
<tr>
<td>Semester.name</td>
<td>90.32</td>
<td>82.14</td>
<td>86.25</td>
</tr>
<tr>
<td>Semester.year</td>
<td>90.32</td>
<td>93.33</td>
<td>91.80</td>
</tr>
<tr>
<td>Teach</td>
<td>93.21</td>
<td>95.37</td>
<td>94.27</td>
</tr>
<tr>
<td>Register</td>
<td>93.94</td>
<td>93.94</td>
<td>93.94</td>
</tr>
<tr>
<td>Course_TA</td>
<td>90.00</td>
<td>75.00</td>
<td>81.82</td>
</tr>
<tr>
<td>Course_Marks</td>
<td>88.89</td>
<td>80.00</td>
<td>84.21</td>
</tr>
<tr>
<td>Teach.overview</td>
<td>100</td>
<td>77.78</td>
<td>87.50</td>
</tr>
<tr>
<td>Course_Marks.marks</td>
<td>75.00</td>
<td>90.00</td>
<td>81.82</td>
</tr>
<tr>
<td>Binary</td>
<td>96.91</td>
<td>96.91</td>
<td>96.91</td>
</tr>
<tr>
<td>Count</td>
<td>96.61</td>
<td>97.44</td>
<td>97.02</td>
</tr>
<tr>
<td><strong>Overall</strong></td>
<td><strong>91.74</strong></td>
<td><strong>92.09</strong></td>
<td><strong>91.78</strong></td>
</tr>
</tbody>
</table>

**Table 3.4** Concepts Identification Results with old features

user queries. The authors categorize problems into a nine cases, of which 3 can be handled by Concepts Identification. For each case, the authors propose procedures to retrieve the missing information. However, in the context of real-time interactions, such algorithms are not user-friendly and are time consuming. We identify an interesting application of Concepts Identification to handle 3 cases from the nine cases the previous authors [47] have proposed. We mark the case numbers as used by them. We show their approach along with ours. We advice the reader to refer their work for a list of all the cases. Case numbers below are as given by [47].

- **Case 3.1** Example: *Give me the address of the employee Margaret Peacock.*

  In this query a table name (employee) and the values (Margaret Peacock) next to the table name can be identified; however, in this case the name of the columns related to the supplied values have been omitted. The dialogue process devised for solving this kind of problem proceeds as follows:

  – Look in the DB queried for the table (employee) specified.
  – Determine the data type of the supplied value (Margaret) and look into the table for columns whose data types match that of the supplied value.
<table>
<thead>
<tr>
<th>Tag</th>
<th>Precision</th>
<th>Recall</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Users.name</td>
<td>92.63</td>
<td>93.68</td>
<td>93.15</td>
</tr>
<tr>
<td>Users.roll</td>
<td>100.00</td>
<td>95.00</td>
<td>97.44</td>
</tr>
<tr>
<td>Users.stream</td>
<td>87.23</td>
<td>91.11</td>
<td>89.13</td>
</tr>
<tr>
<td>Users.batch</td>
<td>91.95</td>
<td>93.02</td>
<td>92.49</td>
</tr>
<tr>
<td>Users.email</td>
<td>100.00</td>
<td>100.00</td>
<td>100.00</td>
</tr>
<tr>
<td>Courses.name</td>
<td>89.04</td>
<td>90.75</td>
<td>89.89</td>
</tr>
<tr>
<td>Courses.code</td>
<td>83.33</td>
<td>100.00</td>
<td>90.91</td>
</tr>
<tr>
<td>Courses.credits</td>
<td>96.15</td>
<td>94.34</td>
<td>95.24</td>
</tr>
<tr>
<td>Courses.type</td>
<td>88.24</td>
<td>81.04</td>
<td>84.51</td>
</tr>
<tr>
<td>Semester.name</td>
<td>91.25</td>
<td>86.90</td>
<td>89.02</td>
</tr>
<tr>
<td>Semester.year</td>
<td>87.30</td>
<td>91.67</td>
<td>89.43</td>
</tr>
<tr>
<td>Teach</td>
<td>93.21</td>
<td>95.37</td>
<td>94.27</td>
</tr>
<tr>
<td>Register</td>
<td>93.94</td>
<td>93.94</td>
<td>93.94</td>
</tr>
<tr>
<td>Course_TA</td>
<td>90.00</td>
<td>75.00</td>
<td>81.82</td>
</tr>
<tr>
<td>Course_Marks</td>
<td>88.89</td>
<td>80.00</td>
<td>84.21</td>
</tr>
<tr>
<td>Teach_overview</td>
<td>100.00</td>
<td>88.89</td>
<td>94.12</td>
</tr>
<tr>
<td>Course_Marks.marks</td>
<td>75.00</td>
<td>90.00</td>
<td>81.82</td>
</tr>
<tr>
<td>Binary</td>
<td>96.91</td>
<td>96.91</td>
<td>96.91</td>
</tr>
<tr>
<td>Count</td>
<td>99.13</td>
<td>97.44</td>
<td>98.28</td>
</tr>
<tr>
<td><strong>Overall</strong></td>
<td><strong>92.03</strong></td>
<td><strong>92.39</strong></td>
<td><strong>92.21</strong></td>
</tr>
</tbody>
</table>

**Table 3.5 Concepts Identification Results with additional features**

- Display the following question to the user: Regarding Employee, which of the following Margaret may refer to? And display a list (check box type) of column descriptions (corresponding to the columns found in step 2), from which the user has to select a list item.

- Once the omitted column is selected by the user (first name), the query is internally modified by the DM; thus, completing part of the lacking information. In this case, there is still lacking information (the column that contains the value Peacock); therefore, step 3 is executed again but instead of the value Margaret the new question involves value Peacock. The final query would read as follows: Give me the address of the employee whose first name is Margaret and whose last name is Peacock.

- **Case 5.1 Example:** *Give me the address of Alfredo Futterkiste.*

  The algorithm is similar to that for case 4.1, but in this case the problem occurs in the Where clause. In the query a column name (address) is specified as well as a value (Alfredo Futterkiste) associated to the column (address). The algorithm is the following:

  - Look in the DB for tables that contain the table name (address) specified in the query.
  - Display to the user the following question: What does address may correspond to? together with a list (radio button type) of the tables found in Step 1.
<table>
<thead>
<tr>
<th>Tag</th>
<th>Precision</th>
<th>Recall</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>User.professor</td>
<td>86.38</td>
<td>86.75</td>
<td>86.57</td>
</tr>
<tr>
<td>User.student</td>
<td>86.06</td>
<td>86.47</td>
<td>86.27</td>
</tr>
<tr>
<td>Users.roll</td>
<td>95.00</td>
<td>95.00</td>
<td>95.00</td>
</tr>
<tr>
<td>Users.stream</td>
<td>89.36</td>
<td>93.33</td>
<td>91.30</td>
</tr>
<tr>
<td>Users.batch</td>
<td>90.80</td>
<td>91.86</td>
<td>91.33</td>
</tr>
<tr>
<td>Users.email</td>
<td>100.00</td>
<td>100.00</td>
<td>100.00</td>
</tr>
<tr>
<td>Courses.name</td>
<td>87.93</td>
<td>92.32</td>
<td>90.07</td>
</tr>
<tr>
<td>Courses.code</td>
<td>83.33</td>
<td>100.00</td>
<td>90.91</td>
</tr>
<tr>
<td>Courses.credits</td>
<td>96.15</td>
<td>92.59</td>
<td>94.34</td>
</tr>
<tr>
<td>Courses.type</td>
<td>81.82</td>
<td>72.97</td>
<td>77.14</td>
</tr>
<tr>
<td>Semester.name</td>
<td>90.67</td>
<td>80.95</td>
<td>85.53</td>
</tr>
<tr>
<td>Semester.year</td>
<td>88.71</td>
<td>91.67</td>
<td>90.16</td>
</tr>
<tr>
<td>Teach</td>
<td>93.21</td>
<td>95.74</td>
<td>94.46</td>
</tr>
<tr>
<td>Register</td>
<td>93.94</td>
<td>93.94</td>
<td>93.94</td>
</tr>
<tr>
<td>Course_TA</td>
<td>90.00</td>
<td>75.00</td>
<td>81.82</td>
</tr>
<tr>
<td>Course_Marks</td>
<td>88.89</td>
<td>80.00</td>
<td>84.21</td>
</tr>
<tr>
<td>Teach.overview</td>
<td>100</td>
<td>77.78</td>
<td>87.50</td>
</tr>
<tr>
<td>Course_Marks.marks</td>
<td>75.00</td>
<td>90.00</td>
<td>81.82</td>
</tr>
<tr>
<td>Binary</td>
<td>95.96</td>
<td>97.94</td>
<td>96.94</td>
</tr>
<tr>
<td>Count</td>
<td>96.61</td>
<td>97.44</td>
<td>97.02</td>
</tr>
<tr>
<td>Overall</td>
<td>90.04</td>
<td>90.87</td>
<td>90.45</td>
</tr>
</tbody>
</table>

**Table 3.6** Concepts Identification Results on new data without additional features

– Once the omitted table (employee) is selected by the user, the query is internally modified by the DM. The resulting query would be as follows: Give me the address of employee Alfredo Futterkiste. This query is still incomplete; therefore, it has to be further clarified.

**Case 5.2** Example: *How many engines does an M80 have?*

In this query a value (M80) is specified; however, no table name is mentioned nor a column name associated to the supplied value. The algorithm for this type of problem is the following:

– Determine the data type of the supplied value (M80).

– Look for all the tables that constitute the DB.

– Display to the user the question: What does M80 may correspond to? together with a list (radio button type) of the tables found in Step 2.

– Once the omitted table (aircraft) is supplied by the user, the query is modified by the DM. The resulting query would be as follows: How many engines does an aircraft M80 have? Since this query is still incomplete, it has to be further clarified.
### Table 3.7 Concepts Identification Results on new data with additional features

<table>
<thead>
<tr>
<th>Tag</th>
<th>Precision</th>
<th>Recall</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>User.professor</td>
<td>85.83</td>
<td>88.03</td>
<td>86.92</td>
</tr>
<tr>
<td>User.student</td>
<td>86.96</td>
<td>86.96</td>
<td>86.96</td>
</tr>
<tr>
<td>Users.roll</td>
<td>100.00</td>
<td>95.00</td>
<td>97.44</td>
</tr>
<tr>
<td>Users.stream</td>
<td>89.13</td>
<td>91.11</td>
<td>90.11</td>
</tr>
<tr>
<td>Users.batch</td>
<td>90.91</td>
<td>93.02</td>
<td>91.95</td>
</tr>
<tr>
<td>Users.email</td>
<td>100.00</td>
<td>100.00</td>
<td>100.00</td>
</tr>
<tr>
<td>Courses.name</td>
<td>88.64</td>
<td>91.36</td>
<td>89.98</td>
</tr>
<tr>
<td>Courses.code</td>
<td>83.33</td>
<td>100.00</td>
<td>90.91</td>
</tr>
<tr>
<td>Courses.credits</td>
<td>96.15</td>
<td>92.59</td>
<td>94.34</td>
</tr>
<tr>
<td>Courses.type</td>
<td>85.29</td>
<td>78.38</td>
<td>81.69</td>
</tr>
<tr>
<td>Semester.name</td>
<td>90.00</td>
<td>85.71</td>
<td>87.80</td>
</tr>
<tr>
<td>Semester.year</td>
<td>87.30</td>
<td>91.67</td>
<td>89.43</td>
</tr>
<tr>
<td>Teach</td>
<td>93.56</td>
<td>95.74</td>
<td>94.64</td>
</tr>
<tr>
<td>Register</td>
<td>93.98</td>
<td>94.55</td>
<td>94.26</td>
</tr>
<tr>
<td>Course_TA</td>
<td>90.00</td>
<td>75.00</td>
<td>81.82</td>
</tr>
<tr>
<td>Course_Marks</td>
<td>88.89</td>
<td>80.00</td>
<td>84.21</td>
</tr>
<tr>
<td>Teach.overview</td>
<td>100</td>
<td>88.89</td>
<td>94.12</td>
</tr>
<tr>
<td>Course_Marks.marks</td>
<td>75.00</td>
<td>90.00</td>
<td>81.82</td>
</tr>
<tr>
<td>Binary</td>
<td>95.92</td>
<td>96.91</td>
<td>96.41</td>
</tr>
<tr>
<td>Count</td>
<td>99.13</td>
<td>97.44</td>
<td>98.28</td>
</tr>
<tr>
<td><strong>Overall</strong></td>
<td><strong>90.46</strong></td>
<td><strong>91.17</strong></td>
<td><strong>90.81</strong></td>
</tr>
</tbody>
</table>

It is not reasonable to expect the users to go through such rigorous process for getting help from a system. The user will probably get tired by the time above such processes are repeated two or three times. Thus, an approach that is both user-friendly and time saving is important. Concepts Identification can come in handy in the above mentioned scenarios.

### 3.2 Our Solutions

All the above cases can be solved using Concepts Identification.

- **Case 3.1**: Case 3.1 can be handled as Margaret Peacock is tagged as employee.name through Concepts Identification. Specifically, the token Margaret is tagged as B-employee.name and Peacock is tagged as I-employee.name and this in turn can help in identifying Margaret as as employee.first-name and Peacock as employee.second-name. Note that the additional feature of Named Entity Recognition helps in tackling this problem more effectively. This is again similar to identifying Krishna Reddy as professor.name in courses domain, where Krishna is tagged as...
B-professor.name and Reddy is tagged as I-professor.name. The dialogue process devised for solving this kind of problem proceeds as follows:

- Prompt the user as "Are you talking about employee whose first name is Margaret and second name is Peacock?". Note that if there is another Margaret Peacock who is not an employee but say is customer. In that case, the user would have told no to the previous prompt. Here, the CRF models uses the second most probable concept which could be the customer.name. Note that, such problems are rare and moreover whether the user is talking about employee or customer can be further disambiguated through contextual knowledge [2].

**Case 5.1:** Again similar to the above case, Alfredo Futterkiste is identified as employee.name using Concepts Identification. This tag gives us the information about the missing entity which is employee. Thus, address corresponds to the employee table and this problem is handled by Concepts Identification. The dialogue process devised for solving this kind of problem proceeds as follows:

Prompt the user as *Do you refer to the employee whose name is Alfredo Futterkiste?*.

**Case 5.2:** M80 can be easily identified as aircraft or aircraft.name using Concepts Identification through domain specific feature engineering which is the heart of Concepts Identification. A simple feature like alphanumeric nature of a token can be helpful in identifying M80 as aircraft.name. Note that in addition to table name, Concepts Identification also identifies the attribute name. A similar case to M80 in the aircraft domain is the case of course id in the academic domain. For example, consider, How many assignment does ICS251 have? Again, the token ICS251 is alphanumeric and can be easily identified as course.id.

Given that the database has limited number of table and attributes, it is easy to train such cases to fight the problems of fast and user-friendly response in real-time interactions.

### 3.3 Conclusions

- Stanford dependencies and Named Entities can help improve Concepts Identification.
- We make the previously proposed tagset finer, re-annotate data and re-conduct experiments for concepts identification.
- We release both our data and the Concepts Identification system for the benefit of the community.
- Concepts Identification can help in addressing few scenarios of Ellipsis in a dialogue process.
Chapter 4

Towards Producing Compact Results in
User-System Interactions for NLIDB System

4.1 Introduction

In Natural Language Interface to Database (NLIDB) system, the user asks a query and the system presents the user with few results accordingly. For each query posed by the user, the system tries to convert these individual queries into formal language queries (for example, SQL) and retrieves answers from a database. NLIDB systems [13, 3] can be extended to Dialogue systems [1] wherein, instead of restricting the user to ask only a single question, the user can have a conversation with the system. Very often, people do not give complete information while querying the system. Due to this unspecific nature of user queries, the system may output large number of results. Going through a large number of results and picking a subset of results is a tiresome task for a user. In order to avoid this, the system has to give a compact set of relevant results. In this work, we propose a system which tries to give a compact set of results to a user when the size of results is large. For this, the system has to make the query more specific by interacting with the user.

\[U_1: \text{List the eateries in California.[5000 results]}\]
\[S_1: \text{Which cuisine are you looking for?}\]
\[U_2: \text{Italian Cuisine. Looking for good pasta.}\]
\[S_2: \text{Where do you live in California?}\]
\[U_3: \text{I live in Eden street, San Diego.}\]
\[S_3: \text{Restaurant 1, Restaurant 2... [1850 results]}\]
\[U_4: \text{Restaurants which serve alcohol and have outdoor seating?}\]
\[S_4: \text{Restaurant 1, Restaurant 2... [340 results]}\]

Figure 1: Example of user-system interaction
In [40], the authors propose three types of knowledge acquisition strategies (Query Driven, Personal and Show & Ask) and show that these strategies are helpful for the system to learn new slot/attribute-value pairs. We propose a system driven technique focusing on attribute-value pairs, to make a user query more specific. In [1], the authors propose user-system interaction models in a dialogue process in the background of NLIDB systems. Identifying user system interaction models is essential to know the behaviour of a user. We model user-system interactions with respect to how a user becomes specific while interacting with a system. To the best of our knowledge, no work has previously addressed the problem of filtering results of a user query in an NLIDB system. Considering this as an important problem in real-time user-system interactions, we attempt to tackle this problem.

The rest of the chapter is structured as follows. Section 2 discusses types of user queries based on how specific a user is. In Section 3, we model user-system interactions based on how a user becomes specific in a dialogue. We also describe our approach to reduce the size of the results of a user query in the context of an NLIDB system. Section 4 presents experiments and analysis. We conclude in Section 5.

4.2 Specificity of a User Query

Consider the user-system interaction in Figure 1. Each $U_i$ is a user query and $S_i$ is a system response. The size of results of a user query is largely dependent on the details mentioned by the user in a user query. Values in the query are the details provided by the user. We categorize user queries into four types based on how specific a user is. They are:

1. **Strongly specific** If the number of values in a user query is more than one, then the user query is said to be strongly specific. For example, *Give me a good American restaurant on Fairgrounds Dr in Sunnyvale?* Here the user is strongly specific as he mentions more than one value (American, Fairgrounds Dr, Sunnyvale) in the query.

2. **Specific** If there is exactly one value in a user query, then the user query is said to be specific. For example, *List the restaurants in San Francisco.* Here the user is specific as he mentions a value (San Francisco) in the query.

3. **Weakly specific** If there are any adjectives modifying an attribute\(^1\) in the user query, then the user query is said to be weakly specific. For example, *List all the good restaurants.* (restaurant/restaurant name is an attribute here and good is the modifier).

4. **Unspecific** If there are no values or modifiers to an attribute in the user query, then the user query is said to be unspecific.

It is important to convert any given user query to a strongly specific query as the size of results would decrease with increase in specificity of the user query, provided that all the details belong to the same topic or same goal as mentioned initially by the user.

---

\(^1\)In the context of a relational database table, a column is a set of data values of a particular simple type, one for each row of the table. In relational database terminology, column’s equivalent is called attribute.
4.3 Modeling User-System Interactions

As mentioned earlier, the system has to interact with the user to collect additional details and convert the given query to a strongly specific query. Depending on the way in which a user becomes specific in a dialogue interaction with the system, at a high level, we propose two models of user-system interactions. They are:

1. **Hierarchical Model:** In this model, the user query becomes specific in a hierarchical manner. In each dialogue, the user adds values of attributes which are related by the sub-field relation. In Figure 1, consider California ($U_1$), San Diego ($U_3$) and Eden Street ($U_3$). Their attributes are Region_name, City_name and Street_name respectively. The sub-field relation between them is:

   \[ \text{Region}_\text{name} > \text{City}_\text{name} > \text{Street}_\text{name}. \]

   This is because a region consists of cities and a city consists of streets. Linguistically, the concept of hierarchy is similar to **Meronymy**\(^2\).

2. **Non-hierarchical Model:** In a Non-hierarchical model of user-system interaction, users add values of attributes which are neither related hierarchically nor have a well defined relation. In Figure 1, the user becomes Non-hierarchical in $U_4$ by adding details like alcohol, outdoor seating whose attributes do not have definite relation with anything previously ($U_1,U_2,U_3$) mentioned by the user. Figure 2 shows an example of a Non-hierarchical interaction.

   \[ U_1: \text{Show all the restaurants in San Francisco.} \]
   \[ S_1: \text{Could you be little specific?} \]
   \[ U_2: \text{Show all the restaurants open till Midnight in San Francisco.} \]
   \[ S_2: \text{There seem to be many restaurants. You'd like to add anything?} \]
   \[ U_3: \text{Good ambience and 5 star rating.} \]
   \[ S_3: \text{Restaurant 1, Restaurant 2,...Restaurant n} \]

   Figure 2: An example of Non-hierarchical interaction.

In Figure 2, new values added by the user are italicized. Though in each $U_i$, the user adds new information, one cannot define a particular relation between them. Thus, the above kind of user-system interaction where a user does not become specific in a well-defined manner is known as a Non-hierarchical user-system interaction. In the next sub-section, we discuss on how the system prompts a user hierarchically, to assemble the details of a user’s query.

---

\(^2\)https://en.wikipedia.org/wiki/Meronymy
4.3.1 Prompting the User Hierarchically

The given user query is processed to retrieve attributes from the values mentioned in the query. This is addressed by Concepts Identification [52]. In Concepts Identification, each token in the user query is mapped to an attribute in the database schema. For example, Concepts Identification for the query *Give me a good eatery in San Francisco for French food* is shown in Table 4.1.

<table>
<thead>
<tr>
<th>Token</th>
<th>Concept/Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Give me a good eatery in San Francisco for French food</td>
<td>O B-restaurant-name O B-city-name I-city-name O B-cuisine-name I-cuisine-name O</td>
</tr>
</tbody>
</table>

Table 4.1 Concepts Identification

For each identified attribute $A_i$, its corresponding hierarchical list $H_i$ is identified. If the position of the attribute $A_i$ in $H_i$ is $i$, the system prompts the user using the attributes from $i+1$ in the list $H_i$. The corresponding hierarchical list $H_i$ of an attribute is maintained by the system in order to prompt the user. Such lists can be easily created using Wordnet\(^3\), since the domain is limited.

Let $K$ be the threshold value of the size (S) of results, based on which the system will decide whether it should try to reduce the size of results or not. If $S > K$, the system would attempt to reduce $S$, else it outputs the results. The system has to prompt the user to reduce $S$ until $S \leq K$. Depending on the number of attributes, we define two ways to prompt a user hierarchically. They are:

1. **Single attribute:** If there is only one value in the user query ($U_1$), the system prompts the user according to the hierarchy of the attribute identified through the value mentioned by the user in $U_1$.

2. **Multiple attributes:** A user query ($U_1$) can have more than one attribute values whose attributes are not related to each other hierarchically. However, these attributes can have their own hierarchical lists. Figure 1 shows an example of such an interaction. Here, [eateries, Italian, Pasta] belong to one hierarchy and [California, San Diego, Eden street] belong to a different hierarchy. In such cases, the system prompts the user with attributes from each list alternatively. This makes the user query specific across a wider scale, thereby yielding more meaningful results. The results in $S_4$ are the restaurants in Eden street, San Diego which serve Italian cuisine, alcohol and have outdoor seating.

\(^3\)[http://www.nltk.org/howto/wordnet.html]
Algorithm 1 Reducing size of results in a single value [specific query] scenario

1: procedure
2: \( K \leftarrow \text{Threshold value for size of results} \)
3: \( S \leftarrow \text{size of results of } U_1 \)
4: while \( S > K \) do
5: \( \quad \text{Identify attribute } A_1 \text{ of value } V_1 \text{ in user query } U_1 \)
6: \( \quad \text{Identify } A_1 \text{'s hierarchical list } H_1 \)
7: \( \quad \text{prompt with attributes in } H_1 \)
8: \( \quad S \leftarrow \text{update} \)
9: \( \quad \text{if End of list then break} \)
10: output results
11: end procedure

However, it is not necessary that the system should prompt the user with only one attribute. The system can prompt the user with one or more than one attribute. We categorize system prompts based on the number of attributes a system uses in a prompt. They are as follows:

1. **Strong prompt:** If the system prompts a user with two or more than two attributes, the prompt is known as a strong prompt.

2. **Moderate prompt:** If the system prompts a user with only one attribute, the prompt is known as a moderate prompt.

Effect of a prompt on the reduction of \( S \) is:

*Strong prompt* > *Moderate prompt*. This is because a strong prompt gathers more information than a moderate prompt, as it assembles more details.

Depending on the following factors, the system prompts the user accordingly.

a. **Current results size:** If \( S \) is too large, then the system would preferably prompt *strongly*. If \( S \) can be reduced to less than or equal to \( K \) by prompting with only one attribute, then the system would prefer to prompt *moderately*. It is important to note that moderate prompting should be given more priority than strong prompting, as asking the user too many details would make the user boring and also make the process of fetching answers more time consuming.

b. **Position of an attribute in its hierarchy:** If \( S \) is too large and if the position of the attribute in the current \( U_i \) is high in its hierarchy (that is, if there are many attributes after the current attribute in its hierarchical list), then the system would prompt the user *strongly*.

Strong and moderate prompting are important as they help reducing \( S \) in minimum number of steps. Currently, we do not address Non-hierarchical prompting. The next section discusses our experiments.
4.4 Reducing results without prompting a user

We propose few heuristics which can be helpful in reducing the results without prompting a user. They are:

1. Database Knowledge: We also leverage Database Knowledge\(^4\). The system prompts the user with an attribute which gives minimum results. For this, the system follows a simple heuristic using the database.
   \[
   \text{Attribute}_{\text{prompt}} = \arg \min \{ \max(\text{att}[1]), \max(\text{att}[2]), \ldots, \max(\text{att}[n]) \}
   \]
   Here, \(\text{att}[1]\) is the number of results obtained for each value of attribute1. \(\max(\text{att}[1])\) is the worst case number of results obtained for choosing some value of attribute1. \(n\) is the total number of attributes in the ER schema. \(\text{Attribute}_{\text{prompt}}\) is the attribute the system uses to prompt the user. The system chooses an attribute having the minimum of all worst case number of results of all the attributes to present the minimum number of results to the user.

2. Semantic Knowledge: Semantic knowledge reduces size of results by mapping semantics of the given query to database knowledge. For example, in List some good restaurants, tokens like good imply that the attribute rating should be used. So, the system would now list restaurants whose rating is in the range 3-4 (out of 5). Similarly, cheap restaurants would yield restaurants filtered based on attribute rate (cost for 2). Semantic knowledge helps the system in two ways. It helps in reducing the results without prompting the user and at the same time, the system makes sure that these results are more relevant to the user.

3. Scaling of attribute values: If the user asks for a good restaurant (rating between 3 and 4) and if there are too many restaurants in that range, then the system increases the rating to 3.5-4. That is, here the attribute value rating is scaled in order to decrease the size of results.

4.5 Experiments and Analysis

We carried experiments using the state-of-the-art corpus in restaurant domain with 250 queries\(^5\). We extended the corpus size to 300 queries. We conducted the experiment with 100 undergraduate CS students of our university. Each student was given 3 queries and was asked to make each query more specific by adding information. Each query resulted in a dialogue process. This was done under the assumption that there are large number of results for the given query which rightly serves the purpose of the experiment.

We take the threshold value \(K\) as the average number of results produced for a strongly specific query (with only two values) in the database. Ideally, we should output the results once \(S\) is less than or equal

---

\(^4\)Instances/ tuples, values of attributes in various tables in the database.
\(^5\)http://www.cs.utexas.edu/users/ml/nldata.html
to K. However, to gather more data, we did not constrain the users from adding new information to the initially given queries.

Out of the 300 dialogue processes, 55.36% (more than we expected) of the dialogue processes belonged to the Hierarchical model, 25.42% of the dialogue processes belonged to the Non-hierarchical model, 19.2% of the dialogue processes belonged to both Hierarchical and Non-hierarchical models. The observations suggest that prompting a user hierarchically is a promising approach for assembling the additional details of a user query. We observed that the length of a dialogue process was higher when users were given unspecific or weakly specific queries in comparison to strongly specific queries. Theoretically also, this is correct as the scope of adding more information increases with decrease in the specificity of a query. However, this may not be true if a user starts a new goal.

Hierarchical prompting ensures that the final set of results are relevant to the initial query as the system builds information on the lines of values mentioned by the user. The high percentage (55.36%) of the Hierarchical model dialogue processes could be due to the small scale of our experiment. However, in real-time scenarios, as the dialogue size increases, one would observe the increase of Non-Hierarchical patterns, as the system does not impose any constraints for the user to add new information. Irrespective of this observation, it is mainly the relevance factor of hierarchical prompting which makes it reasonable to be used for real-time user-system interactions. Even if the system is not prompting exactly what the user might have added, when asked Can you be more specific or Could you add more details, the relevance factor ensures that the user still benefits from hierarchical prompting. The user is benefited in terms of quality of results, as the query becomes more specific, thereby improving the quality and the size of results. A user could be prompted with any attribute in the database to retrieve details of the user query. We hypothesize that prompting a user with the related attributes would yield meaningful and relevant results (as observed in $U_2, S_2$ in Figure 1). The relation between attributes could be anything. In this work, we take the relation of sub-fields between the attributes.

### 4.6 Conclusion

In this preliminary work, we proposed a system for reducing the size of results of a user query in an NLIDB system. User queries were categorized into four types, based on number of values mentioned by a user. At a high level, we proposed two types of user-system interactions based on how a user interacts with the system. User responses can be related in many ways. Our work focuses on user responses which are hierarchically related, based on the sub-field relations between attributes of the values mentioned in the queries. Experiments suggested that prompting users in a hierarchical fashion is a reasonable strategy to congregate details of a user query. We claim that, instead of prompting the user with Can you add any details?, a system could prompt the user hierarchically. This would make the conversation between the user and a system more interactive, human-like and simultaneously aid in minimizing the size of results. We focused on two factors to alleviate problems during real-time user-system interactions in NLIDB systems. They are:
• **Relevance:** Hierarchical prompting ensures that filtered results are relevant to the user. This is because the user query becomes specific on the lines of the details initially mentioned by the user in his query.

• **Minimum prompts:** The system prompts the user differently based on various factors as discussed in Section 3.1, ensuring that the user gets a compact set of results in minimum number of steps/prompts.

As our system is only concerned with the results of an NLIDB system, it can easily be coupled with any NLIDB system, irrespective of the approach used inside the NLIDB system to generate results for a query. Currently, we are exploring other relevant ways of prompting the user, to assemble more details of a user query. In future, we look to address the Non-Hierarchical model of user-system interaction. We would like to make the corpus available to the society in the hope that others would download and use it.
Lexical Simplification aims at improving the readability and comprehensibility of text by transforming complex text into simple text. Lexical Simplification [51], [9], [27] is the process of replacing a word in a given context with its simplest substitute to enhance the readability of the text. The process should make sure that while replacing words with other variants, the meaning of the text is preserved. Lexical Simplification [50] is useful to a wide variety of target audience like people with aphasia, children and also non-native speakers. Complex Word Identification [49], [37] is considered to be the first step in the pipeline of Lexical Simplification. Lexical Simplification can be divided into four sub-tasks:
• **Complex Word Identification**: This is the first step in the pipeline (Paetzold, 2015) of Lexical Simplification. It is the process of identifying complex words in a sentence which are to be simplified.

• **Substitute Generation**: Substitute generation is the process of producing the variants of the words identified in the previous step.

• **Substitute Selection**: Selecting the right variants which fit the context of the sentence is the main aim of this sub-task to preserve the meaning of the sentence.

• **Substitute Ranking**: Substitute ranking picks the simplest word by ranking words identified in the previous step and finally replaces the complex words identified in the first step.

The overall performance of a Lexical Simplification system is thus crucially dependent upon Complex Word Identification. The problem of Complex Word Identification is relatively new in the field of Natural Language Processing. However, a few approaches have been previously proposed for this task. The simplicity score [26] of a word is computed by integrating both, frequency and length of a word. They consider a threshold value and simplify words only if the word’s frequency is lower than the fixed threshold. Matthew Shardlow [49] explores the frequency thresholding to differentiate between simple and complex words by experimenting with each threshold value on a particular corpus. However, this approach is not practically convincing. The same author also frames the problem as a machine learning classification problem by designing a few features. We approach the problem at hand on similar lines.

The Complex Word Identification (CWI) task is framed as a binary classification problem. Given a word in a sentence, the task is to predict whether the word is simple or complex. A word is tagged with 0 if it is simple and 1 if the word is found to be complex.  

$$c(w) = \begin{cases} 1 & \text{if } w \in C \\ 0 & \text{if } w \in S \end{cases}$$

$C$ is the set of complex words, $S$ is the set of simple words and $c(w)$ represents the class of the word. Here is an example of a sentence taken from the training dataset provided by the organizers.

diamond A frenulum is a small fold of tissue that secures or restricts the motion of a mobile organ in the body.

In the above example, the task requires a system to space the words in bold as complex. The remainder of this chapter is structured as follows. In Section 2, we describe our systems and Section 3 discusses experiments and results. We conclude in Section 4.

---

1Complex: In the context of this shared task, complex words are the words which are difficult to understand for a non-native English speaker.
5.1 System Description

We use the Nearest Centroid Classification technique [34] for the classification of words using Manhattan and Standardised Euclidean distance metrics. This classification method is widely used in Information Retrieval tasks. In this method, each class is represented by the mean of all the training samples belonging to that class in the training data. A new observation is assigned a class label, whose mean is closest to the observation. Given below are the labeled training samples:

\[(\overrightarrow{x}_1, y_1), \ldots, (\overrightarrow{x}_n, y_n), y_i \in Y\]

\[\overline{\mu}_c = \frac{1}{|N_c|} \sum_{i \in N_c} \overrightarrow{x}_i\]

\[\hat{y} = \text{arg min}_{c \in Y} ||\overline{\mu}_c - \overrightarrow{x}_i||\]

where \(Y\) is the set of classes,

\(|N_c|\) is the number of samples in class \(c \in Y\),

\(\mu_c\) is the centroid of all samples belonging to class \(c\),

\(\hat{y}\) is the class assigned to the new observation.

We submitted two systems for this shared task. Our first system uses the Manhattan distance metric for the Nearest Centroid Classification. The Manhattan distance function computes the distance to be travelled to get from one data point to another point in a grid-like path. The Manhattan distance between two points is the sum of the differences of their corresponding components. Manhattan distance between two points \(A(x,y)\) and \(B(x,y)\) is defined as:

\[d(A, B) \equiv |A_x - B_x| + |A_y - B_y|\]

The distance metric used by System 2 in the training algorithm of the Nearest Centroid Classification was Standardised Euclidean, which is a slight variant of Euclidean distance. Euclidean distance between two points is defined as the sum of the squares of the differences between the corresponding components of the points. The Standardised Euclidean distance between two points \(A(x,y)\) and \(B(x,y)\) is defined as:

\[d(A, B) \equiv \sqrt{\frac{(A_x - B_x)^2 + (A_y - B_y)^2}{V}}\]

where, \(V\) is the 1-D array of component variances and the numerator is the Euclidean distance between two points \(A(x,y)\) and \(B(x,y)\).

Both, System 1 and System 2 rely on 5 features to classify the target word as simple or complex. An almost similar set of features was previously employed by Matthew Shardlow [49] to identify complex words using Support Vector Machines. The features we considered for the classification of words are:
• **Unigram Word Probability:** We used the Google Books Ngram Viewer\(^2\) to obtain the word probability of the target word in the sentence. This information was considered only for the year 2000.

• **Length:** We considered length of the target word as a feature because longer words are likely to be complex.

• **Number of Senses:** A word with higher number of senses is relatively more ambiguous in comparison to a word with fewer senses. Number of senses of a word was obtained using WordNet\(^3\) from NLTK\(^4\) package.

• **Syllable Count:** A word with higher number of syllables\(^5\) is likely to be more difficult to be read.

• **CD Count:** The number of films in which the target word had appeared was obtained from the SUBTLEX\(^6\) corpus.

### 5.2 Experiments and Discussions

#### 5.2.1 Data

We used the joint dataset provided by the task organizers for training. The training and test data consisted of 2,237 and 88,221 instances, respectively. Also, the training and test data comprised 200 and 8,929 unique sentences respectively.

#### 5.2.2 Discussions

We discuss our experiments with respect to the training dataset. Experiments were run with a variety of machine learning algorithms using the Scikit-learn toolkit \(^{42}\). However, the Nearest Centroid Classification algorithm was found to outperform other algorithms like Random Forest and Support Vector Machines significantly over 5-fold cross-validation of the training dataset. We tried numerous combinations of features and finalized on the feature set described in the previous section.

There are a few features that we tried during our experimentation but did not include in the final system, as the results turned comparatively lower with their inclusion during cross-validation. We believe that the features are still worth discussing. They are:

1. **Average Word Length of Synonyms:** If the length of the given word is greater than the average length of all its synonyms, then the word is tagged as 1, else 0. This gives us a relative estimate

\(^2\)https://books.google.com/ngrams
\(^3\)http://www.nltk.org/howto/wordnet.html
\(^4\)http://www.nltk.org/
\(^5\)http://www.syllablecount.com/syllables.
\(^6\)http://zipf.ugent.be/open-lexicons/interfaces/subtlex-uk/
on whether people would prefer to use a word more regularly in comparison to its synonyms. A similar feature could be tried using frequency or syllables in addition to length of the word. But for our experiments, we only used the length feature.

2. **Rank of a Sense:** We use the Lesk algorithm from Wordnet to find the sense of a word in the given sentence. We find the corresponding sense and the rank of the sense based on frequency using Wordnet. A lower rank of a sense suggests higher frequency of usage of that sense [12]. A higher frequency of a sense indicates that the word is likely to be inferred as simple. The rank of a sense is divided by the total number of senses of the word to get a normalized measure of the feature.

3. **Number of Synonyms:** Number of synonyms of a given word was considered. This information was obtained using Wordnet.

4. **Automated Readability Index of a Sentence:** This feature provides a measure of how easily a user can comprehend the given text. Automated Readability Index (ARI) of a given text is defined as

\[
4.71 \times \frac{C}{W} + 0.5 \times \frac{W}{S} - 21.43
\]

where \( C \) is number of characters

\( W \) is number of words

\( S \) is number of sentences in the given text

A lower ARI score indicates that the given text is easier to comprehend. ARI scores should be used more innovatively for better results in Complex Word Identification.

5. **Collocation Score:** Collocations are a set of words which occur together frequently. We find collocations for the target word in each sentence. We calculate a collocation score (C-score) defined as

\[
C\text{-score} = \frac{C}{N}
\]

where, \( C \) is the number of collocations matched in the sentence and \( N \) is the total number of tokens in the sentence. A higher C-score value indicates that the word usage is less ambiguous. Collocations\(^7\) include Noun, Verb, Adjective, Adverb and Conjunction collocations.

We believe that the first two features are very critical in differentiating simple words from complex ones as words should be inspected in a relative frame with respect to their synonyms and senses. *A complex word may have higher number of senses but it does not necessarily imply that a word with higher number of senses is always complex.* It is fairly possible that a word (with many senses) in the given sentence may turn out to be the most frequently used sense and hence appear to be simple.

Thresholding based on frequency of a word is a common and effective way to identify complex words. The PLUJAGH team submitted a frequency threshold-based system and it stood first in this

---

\(^7\)Collocations were collected from http://prowritingaid.com/free-online-collocations-dictionary.aspx
shared task when evaluated on F-score. In addition to simple thresholding, it is essential to consider relative frequencies of a word with respect to their synonyms. For example, consider the words *eldest* and *oldest*. From a manual experiment, we found that *eldest* is a complex word and *oldest* is a simple word. The frequency of *oldest* is higher than the frequency of *eldest*. However, the word *unused* (from the training dataset provided by organizers) which is less frequent (google ngrams) than *eldest* is a simple word. We found 110 words in the training dataset whose frequency is lower than *eldest* and are still simple. We claim that this may be due to the relatively higher frequency/usage of *oldest* with respect to the frequency/usage of *eldest*.

Here’s another interesting example. Consider the synonyms *mourning* and *grieving*. $f(w)$ indicates the frequency of the word $w$. According to google ngrams, $f(\text{grieving}) < f(\text{oldest}) < f(\text{mourning})$. However, both *mourning* and *grieving* are simple words according to the test data gold labels released by the organizers. Now, a general frequency threshold model would output both *oldest* and *mourning* as simple words because they are highly frequent and may output *grieving* as a complex word because of its low frequency. We suggest that, though $f(\text{grieving}) < f(\text{oldest})$ and *eldest* is a complex word, *grieving* could be a simple word because

$$|f(\text{mourning}) - f(\text{grieving})| \ll |f(\text{oldest}) - f(\text{eldest})|$$

$c(\text{mourning}) \in 0$

$c(\text{oldest}) \in 0$

That is, *oldest* is more widely used with respect to *eldest* than how *mourning* is used with respect to *grieving*. The likelihood of knowing *grieving*, given that *mourning* is known, is more than the likelihood of knowing *eldest*, given that *oldest* is known. This notion can be particularly useful to identify less frequent words which are simple, with the help of synonyms. A constraint that emerges with the above hypothesis is that atleast one word amongst the synonyms must be highly frequent. In conclusion, it becomes important to speculate words in a relative frame (of frequency/senses/length) with respect to their synonyms, to improve complex word identification. The effectiveness of this hypothesis could have been empirically better visible with the availability of a larger training dataset.

### 5.2.3 Evaluation Metric

The official evaluation metric of the task is G-score. It is the harmonic mean of Accuracy (A) and Recall (R).

$$G\text{-score} = \frac{2 \times A \times R}{A + R}$$

### 5.2.4 Results

Table 5.1 shows the average G-score obtained using different classifiers for 5-fold cross-validation on the training data. For experiments using the Nearest Centroid Classification, we explored 24 different
distance metrics, of which Manhattan and Standardised Euclidean metrics performed the best. Based on G-score, our systems were ranked 15th and 23rd in the task. Our first system was able to beat all the baseline systems including the threshold-based and the lexicon-based systems. Table 5.2 shows the performance of System 1 and System 2 on the test data.

<table>
<thead>
<tr>
<th>System</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>System 1</td>
<td>0.65</td>
</tr>
<tr>
<td>System 2</td>
<td>0.63</td>
</tr>
<tr>
<td>Decision Tree</td>
<td>0.54</td>
</tr>
<tr>
<td>Naive Bayes</td>
<td>0.61</td>
</tr>
<tr>
<td>AdaBoost</td>
<td>0.58</td>
</tr>
<tr>
<td>Gradient Boost</td>
<td>0.54</td>
</tr>
<tr>
<td>SVM</td>
<td>0.44</td>
</tr>
<tr>
<td>Random Forest</td>
<td>0.53</td>
</tr>
</tbody>
</table>

**Table 5.1** Cross-validation results on Training Data

<table>
<thead>
<tr>
<th>System</th>
<th>A</th>
<th>R</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>System 1</td>
<td>0.546</td>
<td>0.879</td>
<td>0.674</td>
</tr>
<tr>
<td>System 2</td>
<td>0.465</td>
<td>0.860</td>
<td>0.603</td>
</tr>
<tr>
<td>Decision Tree</td>
<td>0.734</td>
<td>0.613</td>
<td>0.668</td>
</tr>
<tr>
<td>Naive Bayes</td>
<td>0.404</td>
<td>0.924</td>
<td>0.562</td>
</tr>
<tr>
<td>AdaBoost</td>
<td>0.767</td>
<td>0.777</td>
<td>0.772</td>
</tr>
<tr>
<td>Gradient Boost</td>
<td>0.826</td>
<td>0.705</td>
<td>0.761</td>
</tr>
<tr>
<td>SVM</td>
<td>0.837</td>
<td>0.546</td>
<td>0.661</td>
</tr>
<tr>
<td>Random Forest</td>
<td>0.794</td>
<td>0.641</td>
<td>0.709</td>
</tr>
</tbody>
</table>

**Table 5.2** Results on Test Data

The reason for choosing the Nearest Centroid Classifier was its consistent and comprehensive dominance over other classifiers tuned over various parameters and different feature combinations during the 5-fold cross-validation on the joint dataset. The results shown in Table 5.2 pertain only to the finalized feature set discussed in Section 3. Table 5.2 also shows the results obtained on the test data using different classifiers with the same set of features employed for System 1 and System 2. Table 5.3 shows the results obtained after adding features discussed in section 5.2.2. At this point of time, we are not sure about why few classifiers like AdaBoost and Random Forest performed way better than System 1 and System 2 on the test data. A possible reason for the poor performance of the Nearest Centroid Classifier in comparison to other systems could be the imbalance between the training and the test data.
size. The test data being highly skewed could probably be another reason. AdaBoost\textsuperscript{8} classifier (Table 5.2) was found to achieve a G-score of 0.772 on test data. This suggests that both, the proposed feature set and the approach presented are competent.

### 5.3 Complex Word Identification for NLIDB

On the first look, one may consider that Complex Word Identification and NLIDB systems are independent of each other. Leonardo da Vinci was indeed right when he said, *Everything connects to everything else.* CWI and NLIDB share a very critical relation with respect to tokens in the natural language query. One of the pre-processing modules of the CPG NLIDB (Gupta et al., 2012) addresses synonym identification. Synonym Identification is done to map tokens present in a natural language to tokens of the ER schema, also known as concepts (discussed in Chapter 3 of this thesis). Consider the following example,

*Who are the faculty teaching UG3 ?* Here, *faculty* should be mapped to *professors* if the ER schema has *professor* instead of *faculty* as the concept.

Consider a second example, *Which subjects does Krishna Reddy teach ?* Here, *subjects* should be mapped to *courses* if the ER schema has *course* instead of the term *subject* as the concept.

Note that in both of the above examples, *professors* and *faculty* as well as *courses* and *subjects* are synonyms. Thus, synonym identification is important for the SQL query formation given that tokens have to be mapped to their corresponding concepts using synonyms. Consider another example, *Who are the students studying NLP ?*

Here, the user means who are the students *registered* for NLP ? Thus, in the above query, the token *studying* has to be mapped to the concept *registered* or *register* in order to form an SQL query. However, this mapping of *studying/study* to *registered/register* cannot be resolved straightforward through synonyms as *study* and *register* are not direct synonyms.

\textsuperscript{8}For the AdaBoost classifier, the number of estimators used was 10.
Thus, Concepts Identification or synonym identification becomes challenging when users use different words or tokens that are not synonym-related or tokens that have not been covered while training, in a natural language query. Though this problem cannot be completely resolved, the problem can be minimized via right usage of terms or concepts in the ER schema. That is, the percentage of mismatches between the token used by a user in the NL query and the concept present in the ER schema can be reduced if one can make sure that the concept used in the ER schema is a direct match of the tokens used by the user (of course, not at the lemma level) in the NL query, most of the times. To make this possible, one needs to have an idea about vocabulary knowledge of the users. Therefore, knowledge of a user’s token usage or a target group’s (non-native speakers or native speakers or children etc.,) token usage should be taken into account in order to create an ER schema which can reduce the mismatch of tokens to concepts. Considering vocabulary knowledge of each user is practically not possible. Therefore, considering vocabulary knowledge of a group can be a reasonable idea. This is what CWI does and can thus help us. Earlier we have addressed CWI for non-native speakers (could be our target group). This can help in building the right (simplest) ER schema by distinguishing complex words and simple words and using only the simple words to build the ER schema. But what if we have multiple simple words? Which word do we finalize in the ER schema? For this, we can use Substitution Ranking [see Figure 5.1] and use the word which is ranked as the simplest one. Three reasons why this hypothesis can be helpful:

- There is no definite hypothesis proposed until now on how concepts in the ER schema are selected. This can be a promising start.
- Choosing the simplest words can help in minimizing the mismatches. [see the examples discussed below]
- This can help in automatic generation of concepts for an ER schema. No manual effort is needed.
- This approach of building an ER schema is also domain portable as in a new domain also we can find simple words.

Considering one of the above examples, Who are the faculty teaching UG3?, the user could have used faculty instead of professor because the length of the word faculty is shorter than the length of the word professors. As discussed earlier in this chapter of Complex Word Identification, it is said that words with shorter length are more frequent and thus simpler [33] for people to use in comparison to words with a longer length.

Likewise, in the example, Which subjects does Krishna Reddy teach ?, the term subject had been used by the user instead of courses. We checked the unigram frequency of the words subjects and courses using google ngram viewer \(^9\), it showed that subjects was way more frequent than courses, which could possibly be a reason for the user’s usage of the word subjects over the word courses. More frequent words are generally simpler than less frequent words [33]. Had the ER schema considered

---

\(^9\)https://books.google.com/ngrams
such factors like word length and frequency and many other features discussed earlier in this chapter in
its construction, one could have possibly reduced the mismatch between tokens and concepts in the ER
schema. A much more careful analysis is required to build such a proper ER schema. However, we do
not further explore this problem as it is out of scope for this thesis. This can be taken up as future work.

Apart from building an ER schema that can reduce the mismatches (between a token and a concept) as
discussed above, Lexical Simplification can be helpful in disambiguating a verb concept or relation con-
cept. Considering the examples,
Who are the students studying NLP?
Who are the students registered for NLP?
Both the above examples mean the same. Here, we need to map studying to register, because the ER
schema has register as a concept and not study. Now, a lot of Lexical Simplification techniques deal
with meaning preservation. Meaning preservation is necessary when a sentence is simplified to ensure
that after simplification, the new simplified sentence has the meaning preserved of the original sentence.
Thus, in the above example, when we identify the verb studying and also find that it has to be mapped to
a concept in ER schema, here register, one can use meaning preservation techniques [53] to map study
to register.
That is, given a query, Who are the students studying NLP?, one has to map studying to register. This
can be done as:

Say we have n verb concepts in our ER schema. Now replace the verb studying with all the verbs
in the ER schema, taking care of verb tenses. Since we have n verbs, we now have n new sentences.
We can then measure meaning-preservation scores [53] between the original sentence and all of these n
sentences one by one. From this, we can get the sentence which has the highest meaning-preservation
score with the original sentence. The verb in the sentence with maximum meaning-preservation score
would be the required concept. Say we have two verb concepts teach and register in the ER schema.
Let the original sentence be Who are the students studying NLP?
We identify the verb studying in the above query and form two queries (since we have two verbs in the
ER schema) by replacing the verb studying with the two verbs. The two queries are:
Who are the students registering NLP? (or) Who are the students registering for NLP?
Who are the students teaching NLP?

From the above two examples, it is very clear that, the first query preserves more meaning than the
second query, thereby, the verb or the concept, that is, register, is the required concept. However, this
should be rectified with various meaning preservation algorithms currently being used in the community.
Thus, one can reasonably use Lexical Simplification techniques to tackle problems in NLIDB systems.
This (NLIDB and Lexical Simplification) would be an interesting intersection that can exclusively be
taken up in future.
5.3.1 How else can Lexical Simplification help NLIDB?

As discussed earlier in this chapter, lexical simplification is the process where first complex words are identified, then potential substitutes are generated, only a few substitutes are selected from the previously generated ones and finally, substitutes are ranked. The highest ranked substitute is deemed to be the most simple one and is considered as the best surrogate for the complex word identified in the first step.

When an NLIDB system outputs an answer, it is important that the answer is generated in the form of a sentence. This is a common routine for NLIDB systems used in a dialogue process [1]. The sentence generation is often taken care by Natural Language Generation (NLG), where the sentence is generated considering various factors like context [1], type of question etc. Here, Lexical Simplification can be helpful to generate the most simple sentence to ensure that even non-native speakers can easily understand what the system is talking about. This would instigate a better and an active conversation among the user and the system. This can be considered as a post-processing module in the framework of an NLIDB system, similar to the pre-processing module where the natural language query is cleaned (spell-checking, auto-correction etc..). Chapter 6 illustrates how a sentence from an NLG can be passed to a lexical simplification component and then be presented to the user.

5.4 Conclusion and Future Work

In this chapter, we described a promising approach for identifying complex words for non-native English speakers using Nearest Centroid Classification technique. Our approach is simple in terms of both, features and the learning algorithms.

We emphasized that words should be inspected in a relative frame with respect to their synonyms and senses. Testing this supposition in depth will be the subject of future work. We further look to improve the system by incorporating phonetic and semantic features. We also look to explore the problem at sentence level, as the complexity of the sentence can influence a person in comprehending a word’s meaning in that sentence. We also discussed about how Complex Word Identification and Lexical Simplification can help NLIDB sytems.
Chapter 6

Evolution of the NLIDB

This chapter looks back at how the IIIT-NLIDB fared as years went by.

- The CPG NLIDB started in 2012 with three main stages, namely, syntactic, semantic and the query processing stage. [20].

![Diagram of NLIDB-2012]

**Figure 6.1** NLIDB-2012

- In 2013, Arjun R. Akula addressed the Contextual Processing Module that uses information from previous dialogues to answer user queries. [1].

![Diagram of NLIDB-2013]

**Figure 6.2** NLIDB-2013
• Aggregates are very important to be identified when NL queries are related to numeric outputs. [21] conducts a systematic study on how aggregates have to be handled in an NLIDB system.

![Figure 6.3 NLIDB-2013](image)

**Figure 6.3 NLIDB-2013**

• Statistical Concepts Identification was introduced in 2014 [52], showing that it performs better than the previously proposed semantic frame approach to identify concepts in an NL query.

![Figure 6.4 NLIDB-2014](image)

**Figure 6.4 NLIDB-2014**
• Statistical Karaka mapper that maps Stanford grammatical relation to karaka relations was proposed [19] in 2014, showing that it performs better than previously proposed rule based mapper.

Figure 6.5 NLIDB-2014

• In 2015, [39] explore how attributes in an NL query can be utilized to build an SQL query which is formed with the help of graph processing stage in [20].

Figure 6.6 NLIDB-2015
• Natural Language Generation (NLG) module that outputs answers in the form of a sentence was addressed by Ravi Chandibhamar, M.Tech report (2014).

![Diagram](Figure 6.7 NLIDB-2015)

• The Compact Result Processor interacts with the user and filters the results according to the user specification, thereby improving the quality of the results. The filtered results are sent back to the NLG module.

![Diagram](Figure 6.8 NLIDB-2015)
• Complex Word Identification, the first step of Lexical Simplification that is important to produce simplified sentence was addressed in 2016 as a part of SemEval-2016 shared task. More importantly, its relation with NLIDB system has been discussed, particularly at the level of Entity Relation schema.

![Diagram of NLIDB-2016](image)

**Figure 6.9 NLIDB-2016**

• The work in this thesis contributes to:
  - Concepts Identification ([52], Chapter 3 of this thesis).
– Statistical Karaka Mapper\(^1\) ([19])
– Attribute Classifier ([39], Chapter 2 of this thesis).
– Compact Result Processor (Chapter 4 of this thesis).
– Complex Word Identification ([38], Chapter 5 of this thesis).

\(^1\)Though the author of this thesis has contributed towards the Statistical Karaka Mapper [19], this has not been discussed in this thesis to avoid redundancy, as Sai Kiran Gorthi has already discussed about it in his thesis.
Chapter 7

Future Directions

The future directions of each chapter of this work can be summarized as follows:

1. Attribute Classification:
   - Addressing nested SQL queries can be explored. Additional tags have to be introduced. Classification has to be done further between whether an attribute belongs to the normal clause or the nested clause.
   - In addition to using surface level features, semantics of the natural language query can be utilized to improve the reasoning capabilities of an NLIDB system.
   - It would be interesting to see if parsing techniques can be merged with our system that focuses on the context of a token.

2. Concepts Identification:
   - We show better results than Gupta et al., 2012 using sequence learning for classifying tokens into their corresponding concepts. We then improve our method using Stanford dependencies which are close cousins of Karakas proposed by Gupta et al., 2012. As mentioned by Gupta et al., 2012, Karaka relations are syntactico-semantic and perform effectively. However, the reason we merged sequence learning with Stanford dependencies is the unavailability of a system which gives reliable Karaka relations in English sentences. If tools that can generate Karaka relations in English are developed, then the Stanford relations in our work can possibly be replaced with Karaka relations. However, to develop a reliable tool, one needs to train it using large Karaka annotated data (unavailable).

3. Classification of Concepts:
   - In attribute classification, attributes were identified with human supervision. We did not directly classify the concepts into different SQL clauses, as the errors in concepts identification would propagate into SQL query. Classification of concepts will be an ideal way to form an SQL query rather than considering direct tokens as attributes and then constructing the SQL query.
4. Compact Results:

- We modeled user-system interactions on just one pattern of how a user may become specific while interacting with a system. Many other patterns can be explored and new models can be proposed.
- WordNet provides many relations between words. It would be interesting to see if these relations can be explored between words of different user responses in a user-system interaction to craft a new method of system prompting. In our work, meronymy relation was focused upon.
- We consider the threshold value K as the average size of results for a specific query in a database. This depends on the database and can differ from one database to another. It would be interesting to see if new factors can be utilized to compute the value of threshold.
- We use database knowledge to choose

5. Complex Word Identification:

- Sentence structure can be utilized (apart from sentence length, number of syllables in a sentence, readability scores, grammatical score that have been explore in the community) to identify how the word is used in the sentence and predict if it is a complex word.
- Relative theory proposed in chapter 5 is one thing which we are currently working on.
- Our work uses only the joint dataset provided by the organizers of SemEval-’16 task-11 organizers for Complex Word Identification. Most systems in the shared task used the joint dataset. Can the disjoint dataset be put to use for achieving better results? That would be interesting.
Appendix A

Examples of Complex Words

The following are examples of sentences with complex words marked in bold.

1. In 1832 his family emigrated *thence* to Belleville, Ontario, where he *apprenticed* with the printer at the town newspaper, The Belleville Intelligencer.

2. Leo, on December 23, took an *oath* of *purification* concerning the *charges* brought against him, and his opponents were *exiled*.

3. Their *thick, shimmery fur sparkles* with tiny lights; their Kewpie doll style heads are hairless and feature big eyes with *rows* of lights for eyebrows.

4. The structure is known as the middle *ear*, and is made up of the *incus*, *stapes*, *malleus*, and *tympanic membrane*.

5. Twenty-six of these 0-6-0 *locomotives were* ordered in January 1963, to be built at British Railways Swindon Works.

6. The United States *convened* a 13-nation conference of the International Opium Commission in 1909 in Shanghai, China in response to increasing *criticism* of the *opium* trade.

7. Hanna *meandered* around the southeastern Bahamas, weakening to a *tropical* storm while also *dumping* heavy rain on *already-devastated* Haiti.

8. By 1960 he had developed the short story into a *screenplay*, and *envisaged* it as containing a *suitable* role for Monroe.

9. The last event was *held* on June 11, 2000, not to be held again due to the *acquisition* of WCW by World Wrestling Federation.

10. The *purpose* of public speaking can range from simply transmitting information, to motivating people to act, to simply telling a story.
Appendix B

Examples of User queries for Hierarchical and Non-Hierarchical Models

1. Hierarchical

   • – Where is a restaurant in Sunnyvale that serves good American food?
   • – Location of a restaurant in Sunnyvale that serves good American and Chinese food?
   • – Location of a restaurant in Sunnyvale which serves gelatos and good American food?
   • – Where is a good Arabic restaurant in the Bay Area?
   • – Where is a good Arabic restaurant in the north bay area?
   • – Where is good Arabic restaurant for dinner in the bay area?

2. Non-Hierarchical

   • – Where are some good cafes on Webster St in Alameda?
   • – Best coffee cafe on webster in Alameda.
   • – List cheapest cafe on Webster st in alameda.
   • – Give me some good places for ice cream in Alameda?
   • – Give me some good places for ice cream in Alameda at night?
   • – Give me some good places for couples to have ice-cream in Alameda at night?
   • – Give me some good, cheap places for couples to have ice cream in Alameda at night?

3. Mixed

   • – Where is a good bakery on Shattuck ave in Berkeley?
   • – Where is a bakery that serves delicious pastries on Shattuck ave in Berkeley?
   • – Serves customized cakes?
   • – Affordable?
   • – Within 5km of Shattuck ave in Berkeley?
Related Publications

1. **Title:** Classification of Attributes in a Natural Language Query into Different SQL Clauses.  
   **Authors:** Ashish Palakurthi, Ruthu S M, Arjun R. Akula and Radhika Mamidi.  
   *In Proceedings of Recent Advances in Natural Language Processing (RANLP)*, pages 497-506, Hissar, Bulgaria, Sep 7-9, 2015.  
   **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 an 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.

2. **Title:** Concepts identification of an NL query in NLIDB systems.  
   **Authors:** Saikrishna Srirampur, Ravi Chandibhamar, Ashish Palakurthi and Radhika Mamidi.  
   **Abstract:** This paper proposes a novel approach to capture the concepts of an NL query. Given an NL query, the query is mapped to a tagset, which carries the concepts information. The tagset was created by mapping every noun chunk to the attribute of a table (tableName.attributeName) and every verb chunk to a relation in the ER schema. The approach is discussed using the Courses Management domain of a University and can be extended to other domains. The tagset here was formed using the ER-schema of the Courses Management Portal of our university. We used the statistical approach to identify the concepts. We ourselves formed a tagged corpus with different types of NL queries. Conditional Random Field algorithm was used for the classification. The results are very promising and are compared to the rule based approach seen in Gupta et al. (2012)

\(^1\)Structured Query Language is a specialized language used for relational database management and data manipulation.
3. **Title:** IIIT at SemEval-2016 Task 11: Complex Word Identification using Nearest Centroid Classification.

**Authors:** Ashish Palakurthi and Radhika Mamidi.

*In Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval 2016).*

**Abstract:** This paper describes the system that was submitted to SemEval2016 Task 11: *Complex Word Identification.* It presents a preliminary investigation into exploring word difficulty for non-native English speakers. We developed two systems using Nearest Centroid Classification technique to distinguish complex words from simple words. Optimized over G-score, the presented solution obtained a G-score of 0.67, while the winner achieved a G-score of 0.77 and the average G-score of all the submitted systems in the task was 0.56.

4. **Title:** Identification of Karaka relations in an English sentence.\(^2\)

**Authors:** Sai Kiran Gorthi, Ashish Palakurthi, Radhika Mamidi, Dipti Misra Sharma.

*In Proceedings of ICON-2014, Goa, India*

**Abstract:** In this paper we explain the identification of karaka relations in an English sentence. We explain the genesis of the problem and present two different approaches, rule based and statistical. We briefly describe about rule based and focus more on statistical approach. We process a sentence through various stages and extract features at each stage. We train our data and identify Karaka relations using Support Vector Machines (SVM). We also explain the impact of our work on Natural Language Interfaces for Database systems.

---

\(^2\)This publication has not been discussed in this thesis to avoid redundancy. Sai Kiran Gorthi has already discussed about it in his thesis. Note that this publication also discusses a critical problem in the CPG NLIDB system.
Bibliography


