Semantic Parsing for Automatic Generation of SQL Queries using Adaptive Boosting

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE SCHOOL

OF THE UNIVERSITY OF MINNESOTA

BY

Rajesh Tripurneni

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

Dr. Richard F. Maclin

May 2014

Acknowledgements

I am eternally grateful to Dr. Maclin for all the support that he extended throughout this work and for the enormous trust he had put in me. You are an epitome of the ideal guru Sir and have always lived up to the saying that being nice is never being bad.

I would like to thank all the members of faculty for guiding me in academic, professional pursuits not limiting to Dr. Carolyn, Dr. Pedersen, Dr. Turner and Dr. Willemsen. I would also like to acknowledge the good work by the two lovely ladies of the department, Lori & Clare. Thanks to Jim for his technological prowess and making our computers just a little less annoying.

I also wanted to remind myself that none of this work would have been a reality without the sacrifices of my family. Dad, you've taught me lessons of life that I am just starting to realize and Mom, I wish I could exhibit a finite amount of your perseverance.

Finally to all my friends who had been there whenever I needed them and made my stay memorable, you will be never forgotten.

Abstract

The key step of semantic parsing is to learn a connection between parts of the grammar and parts of the English statement. Many approaches have been generated, but we will be focusing on the mechanism introduced in work by Zettlemoyer (**Artzi and Zettlemoyer, 2011**). This work attempts to learn a probabilistic grammar in a bootstrapping manner, by looking for commonalities in the domain of cricket. For example, if several of the example queries in the game of Cricket have the term "centuries" in it and there is always a corresponding part of the query generated that includes a class such as *give centuries* clause, it might be reasonably concluded that the term "centuries" is a strong predictor of that clause. As more of these connections are made the learner can focus on the remaining words and corresponding parts of the parse tree and attempt to make further connections. This approach is similar, though a different mechanism is used by Kate (2008). Results obtained were promising and proves the efficiency of the model against previously performed work.

Contents

List of Tables

List of Figures

List of Algorithms

1. Introduction

Computers are very useful in helping humans to make effective predictions. For these predictions to hold true, the humans needs to program computers. Computer Programming is a comprehensive strategy that aims at solving certain complex problems such as performing numerical calculations, making predictions and providing communication between users. Programming computers is a skill where humans can easily address problems that are complex to comprehend. For example, it would be useful for humans to program in such a way that it can respond to natural language. But as this task is quite complex, it has so far eluded us in general.

One interesting area, that attempts to tackle this problem, is Question Answering (Q and A). A *Q and A* system can be defined as a programmed system that can handle questions and then suggest answers to questions basing on the question that was posed. Research by **Coleman (1997)** involved a Q and A system that could generate a prediction from a collection of 600 natural language sentences. One area of interest for Q and A system that could help in making effective predictions from natural language by making use of Computers is Semantic Parsing.

Semantic Parsing attempts to find a connection between a natural language sentence and its meaning that is represented in a formal language. For example, Semantic Parsing can be used to learn connection between parts of statement from English language and its meaning. Many approaches have been generated, but we will be focusing on the mechanism introduced in the work by **Artzi and Zettlemoyer (2011)** Previous work in this area had tried to find the connections involved within sentences written from Geographic sets **(Zettlemoyer and Jurafsky, 2011).**

Our work attempts to use semantic parsing, to learn the relations involved within a query written in natural language based on test task using the game of cricket. We try to predict an answer to a query from the associations it developed from a similar query encountered previously. Our method looks for commonalities For example, if several of the example queries from the game of cricket have the term “centuries” in it and there is always a corresponding part of the query generated that includes a clause such as *give centuries scored in a match***,** it might be reasonably concluded that the term “centuries” is a strong predictor of that clause. As more of these connections are made the learner can focus on the remaining words and corresponding parts along with attempting to make further connections.

1.1 Thesis Outline

The details of this thesis are presented in the order given as under. Chapter 2 presents the background for this work. This section broadly covers parsing in general, semantic parsing in particular, logic and natural language. Chapter 3 presents our approach to solving the problem. This chapter deals with generation of derivations, trees and other data structures involved; employing techniques such as semantic decomposition, parse proposals and lexicon expansion. Later, this chapter explains how a learning model for this work has been built and it is compared against similar models. Chapter 4 presents a brief comparison of results that are obtained with this work. The fifth chapter deals with future work that can be carried out and some recommendations, conclusions.

2. Background

This chapter presents the background needed to understand this thesis. The semantic parser built as part of this work will be presented in detail in Section 2.1. Section 2.1 also presents an overview of parsing, parse rules and parse trees. Section 2.2 deals with semantic parsing. Section 2.3 covers logic, natural language and representations used as part of this work. Section 2.4 presents the Combinational Categorical Grammar (CCG) algorithm and its usage in general.

2.1 Parsing, Rules, Derivations and Parse Trees

Natural language can be analyzed in two ways. One of the ways is shallow analysis that analyses text at a lower level and the other way is semantic analysis that examines at a much deeper level than shallow analysis. Shallow analysis embodies tasks such as Information Retrieval (**Zettlemoyer and Collins, 2006**) and Semantic Role Prediction (**Gildea and Jurafsky, 2010**). Shallow analysis fails to capture long distance relationships that can be explored by semantic analysis. Take, For example, the words *vague, unclear* are similar in terms of human language but a machine would not be able to identify the difference Thus there exists a relationship between the two words that is not obvious to a machine and shallow analysis fails in this aspect.

In this work, a semantic parser is tested on a cricket dataset involving queries similar to SQL. SQL stands for Structured Query Language and is a formal mechanism for designing, maintaining and querying standard relational database management systems. An example SQL query might be:

```
select * from odiplayers;
```

where odiplayers refers to a group of players playing ODI matches within the game of Cricket grouped as a table and * refers to the entire group of entities in the table concerned.

The first ten results that can be obtained from this query might be listed as below:

Sachin, Anil Kumble, Glenn McGrath, Ricky Ponting, Adam Gilchrist, Saurav Ganguly, Ishant, Sharma, Ajit, Agarkar. We are interested in SQL for this work because we propose to use SQL as a bridge between natural language and the formal description of a query. A SQL table comprising of all the names of the players might be presented as:

odiplayers
Sachin
Anil Kumble
Glenn McGrath
Ricky Ponting
Adam Gilchrist

A key aspect of *parsing* deals with analyzing a string of symbols to form a meaningful sentence based on a pre-defined set of rules called productions. Parsing involves a formal derivation of any sentence from a language and thereby generates a parse tree exposing the relations involved, based on the grammar. Parsing thus helps in recognizing a string in natural language and breaking it down to a meaningful set of symbols and analyzing each of the symbols against rules of a given formal grammar.

A *grammar* is a set of rules that govern the composition of phrases, production rules in a formal language. A production rule is a rule specifying a substitution that can be performed recursively to generate a new symbol. Production rules are made up of two types of symbols known as terminal symbols and non-terminal symbols. Terminal symbols are the set of symbols that appear in a formal grammar and cannot be changed any further whereas non-terminal symbols can be replaced further and derived. For example, the rule:

$$S \rightarrow N \quad VP$$

helps in rewriting the derivation N into even smaller notion until no such derivations exist any further. For example, VP cannot be replaced with any production further, it can be termed as terminal symbol and since N can be replaced with further productions, it’s a non-terminal symbol. Consider a simple language and a grammar to it; consider two non-terminal variables a, b and a start symbol 'S'. Le the corresponding set of productions be given as:

S→aSb

S→a

S→b

S→☐

Some sample strings that can be generated with the above grammar are given as:

☐, ab, a, b, aabb, aaaaabbbbb, aaaabb and **abbbb.**

Thus, to generate a symbol in a language, we begin with a string known as *start symbol* and successively apply rules of grammar to rewrite the string. A language thus consists of all such strings that can be generated in this manner from the set of grammar rules. Parsing can be broadly classified into two variants: top-down parsing and bottom-up parsing. A top-down parser proceeds by starting with the start symbol first, and iteratively replaces a non-terminal symbol with the right hand side (RHS) of a production until no non-terminals are left, while a bottom-up parser starts with a given input symbol and attempts to replace symbols with left hand side (LHS) of rules to eventually leave only the start symbol. In this thesis, a version of the CYK (Cocke-Younger-Kasami) parsing algorithm that uses a bottom-up strategy similar to the lines of **Kate and Mooney (2006)** is implemented except that dynamic programming is made use of, for this work.

A *Parse Tree* is a tree structure that reveals the syntactic structure of a string basing on rules of a formal grammar. A parse tree structure helps in deciphering out the symbols; strings involved depending on the grammar. Parse tree makes use of non-terminal symbols of grammar, terminal nodes as leaf nodes and terminal nodes in a tree structure. A typical example of a parse tree involving a natural language sentence, its constituent grammatical categories may be denoted as:

Sentence: Sam ate the cake

Here the syntactic categories involved are **S, D, VP** and **N**

The grammar rules corresponding to this parse tree may be presented as below:

$$S \rightarrow N\ VP$$

$$S \rightarrow N\ NP$$

$$VP \rightarrow V\ NP$$

NP → D N

NP → D V

The parse tree is given in Figure 2.1:

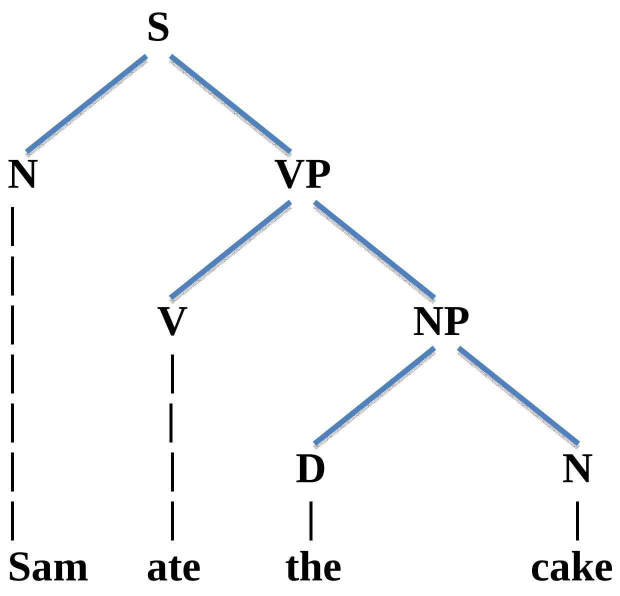


Figure 2.1 A Parse Tree with Derivation

As noted above, production rules are recursively substituted to generate a new symbol. For this grammar and parse tree, as each of the production rules are substituted, the structure of the corresponding language is revealed. The idea of this parsing strategy is to build a semantic parsing technique that maps any sentence from natural language to its meaning representation. An example that underscores this strategy, might be presented as: *How many fours by Sachin?*

The corresponding meaning representation can be presented as: $\lambda x.count(x) \quad \Lambda$ $\lambda x.player\ (Sachin)$

Coupling the original natural language sentence with its meaning denotation helps in making effective prognostications about similar queries. This technique is used in this work to derive effective predictions from the set of productions that have been already defined and derived. Take, For example, a spam filter that detects spam received from similar senders from that was previously encountered thus learning what typically classifies as spam. This involves building an effective learning strategy. Details pertaining to the learning strategy used for our work are presented in Chapter 3.1. A sample query with its meaning representation for another query might be presented as:

Q: *Name the player who hit the maximum runs between India and England at the Lords?*

A: *select* ***player*** *where* ***player.runs*** *=* ***max (runs)*** *and* ***venue*** *= lords where* ***teams.played*** *= India and* ***teams.played*** *= England.* A parse tree for the same is be presented as:

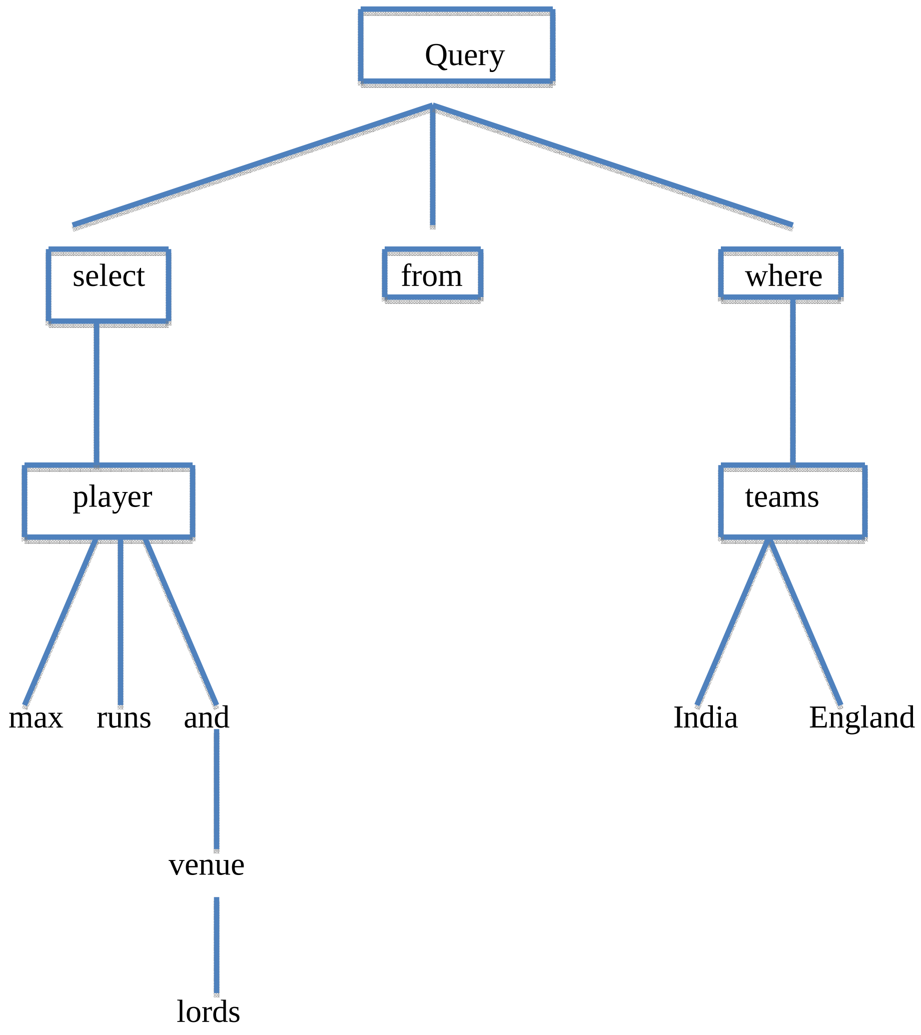


Figure 2.2 A Parse Tree for SQL Query

As shown in figure, the query can be represented in terms of the individual units that represent it in the form of a tree structure. The learning strategy thus plays a crucial role in this work. A novel Cricket dataset developed for this work tests the entire approach.

2.2 Semantic Parsing

Semantic Parsing is a process of mapping a natural language sentence into its formal representation of its meaning and then associating that meaning with an answer. Semantic parsers can be used in a wide variety of situations that is not just limited to: generating advice-taking learners (**Murzyn, 2009)** helping with formal database queries (**Dorothy, 2008)** developing formal command-based languages. It is a machine learning strategy used to help with a variety of   learning mechanisms.

As part of this work, we employ a learning mechanism to learn a parser that uses a series of sentences of the form (sentence, logical form) similar to the lines work of **Elbridge and Zettlemoyer (2007).** The learning strategy plays a critical role in generating predictions from data that have been seen and learned previously. An example of a sample semantic learner may be geographical datasets is dealt at a later stage in this chapter.

Work by **Zettlemoyer and Collins (2005)** makes use of a dynamically induced semantic parser. Most recent work on semantic parsing explores the use of using shallow semantic parsing. Our work presents a mechanism that tries to integrate both natural, formal languages and tries to bring a language model that exposes the intricacies of both. This work aims at generating such a model that takes the form: sentence, meaning. An example that outlines this representation has been previously presented in Chapter 2.1. The goal is to build and predict such a model similar to a query answering (Q and A) system similar to the one presented in the Introduction section. As part of the work, probabilities for each of the parsed sentences are used in estimating the best answer to a given query. The probabilities associated with any parse may be calculated may be calculated using a mathematical formula:

$$P(s, t) = \prod_{a \to b \in \text{domain}} P(b/a)$$

where ‘t’ is a possible derivation within a sentence, ‘s’ is a sentence and ‘a’, ‘b’ are any possible parses available in the derivation.

When it comes to predicting a given query, the conditional probability of the available parses is chosen. Statistically speaking a *conditional probability* is the probability of an event occurring when another event is known to have occurred. In terms of the model built for this work, it is the amount of certainty a phrase or a word reoccurs after it is witnessed previously. This approach is used in building the semantic parser.

2.3 Logic, Representation, Natural Language and Meaning

One of the goals of natural language is to find an interesting relation between sentences from natural language and tag them with their actual meaning representations. Recent work by **Zettlemoyer and Kwiatkowski (2009)** addresses a part of this issue. In this work, any sentence from natural language is mapped with its unique meaning or representation. As outlined, first order predicate logic is made use of to build the corresponding semantic parser.

First-order logic is a formal system where logical expressions are built from a set of variables, logical constants and non-logical constants. First-order logic differs from predicate logic by its usage of fully quantifiable variables. First-order logic systems also satisfy the condition that all sound systems in it are complete. The logical constants that are commonly used as part of this calculus would be disjunction, conjunction, negation and equality. Some of the non-logical constructs are identifiers, predicates and functions. *Disjunction* is a logical constant that is used synonymously with "or" and is denoted by $\vee$. Say for usage Sachin $\vee$ Ganguly. *Conjunction* is another commonly used logical construct and is denoted by $\wedge$, read as "and". For example, Sachin $\wedge$ Ganguly. *Negation* is another construct denoted by $\neg$ and is used to imply the fact that if variable is true, its negation is false and vice-versa. A *function* maps elements from one set to a different set. A *predicate* may be defined as a Boolean valued function. It is a statement that may be either true or false basing on the value of variables involved. Take, For example; given a variable X, and a predicate, P: X $\rightarrow$ {true, false} can be termed as a predicate on X. Computational formulae are built by employing constructs and by adding terms such as variables that render complete meaning to a representation. A suitable example to present this from Cricket domain:

Runs scored by Sachin

that might in turn be represented in a small meaning representation as follows:

player (Sachin) and count (runs)

In similar lines, object relationships can also be created between persons or entities, that are normally used in natural language. In the next example, establishing a relationship between two object entities and a function has been depicted. For this, quantifiers are used to bind the scope of variables used and make formulae or expressions more suitable to represent. By making use of an existential quantifier to limit scope for our usage, the sentence can be transformed as:

Maximum runs score of Sachin?

the transformed representation for this query might be presented as:

player (Sachin) and max (count (runs))

Lambda Calculus is made use of in deriving the meaning representations of sentences involved. Lambda Calculus is a formal system used along side with mathematical logic for denoting computations by making use of common variables, quantifiers and by associating variables with formal relations. Lambda Calculus systems can be broadly divided into typed and non-typed systems. An example for a formal Lambda Calculus system may be presented as below:

Given a variable x a valid lambda term, Another variable y a valid lambda term, A function f(), with a valid definition, let f (x) = x * y, Given ‘x’ a valid term, λx is called a lambda term or application, thus any usage of λx to the function denotes x * y. Lambda Calculus work from **Turing (1968)** addressed a suitable mechanism that was a simple type-free representation to restrict the range of expressions varying under the finite domain and was in use from then. Lambda Calculus and its representation with a simple atomic grammar here is employed as part of this work. Parsing is thus regarded successful as long as it is within the confines of a previously defined grammar. This approach is used in developing a type-hierarchy that can be later used to create and recognize any domain-specific language. In the semantic parser that was developed as part of this thesis, significant usage of SQL (Structured Query Language) is made to perform testing as part of the result validation and in augmenting the type representations. SQL typically consists of managing a relational database management system and is based on algebraic tuples and relational calculus. A simple representation that has been used from the Cricket domain might be as: *Which teams played the World cup of 2003?* Generating a suitable answer to such a query, requires the semantic parser to identify the underlying grammar and map it to its SQL version as: *Select teams from ODI where TOUR = 'WORLDCUP'*. Reducing and obtaining a query result from any database typically involves data manipulation, data definitions, transaction control and data control.

2.4 **Combinational Categorical Grammar (CCG)**

Parsing natural language has always been a daunting task. Recent work in this area includes Berkeley parser (**Johnson, 1999**), Stanford parser (**Klein, 2010**). Both of the approaches have been successful in developing customized parsers that parse natural language by a pre-defined specification of grammar rules. In this work, we have used a methodology similar to the work of **Zettlemoyer (2009)** for effectively handling CCG. A *Combinational Categorical Grammar (CCG)* is a formal grammar mechanism that is lexicalized and the grammar's rules are entirely based on combinatory logic. Linguistically speaking, a grammar is *lexicalized* when it is formed from individual words, lexical morphemes. A *morpheme* is the smallest grammatical unit in a language. Say, word **cannot** is lexicalized from the two words **can** and **not**. One of the goals of

CCG is to assign natural language a syntactic structure. Say, For example, the sentence: **The book is on the table in the box** can be analyzed and understood in two different connotations. Sentences from natural language are assigned a grammatical structure using CCG to avoid this ambiguity. To understand how CCG forges relations and assigns a structure to sentences from natural language, the underlying components of CCG are presented in detail as below:

- *Categories:* They specify the list of constituents that make up CCG
- *Lexicon:* It assigns lexical categories to words
- *Combinator and rules:* They specify how the individual entities combine and what combinations are valid.

A detailed description of the components that make up CCG follows:

A *category* is a syntactic category in a grammar specification. A lexical category can be either an atomic category such as: **S, NP, PP** or a complex lexical category. An *atomic category* is one that is simple with no usage of operators and slashes. An example of an atomic lexical category is given as: **N, NP, S** and **NNP**. An atomic category is also known as simple category as it does not involve any rules or combinations. A *complex lexical category* is one that is built from basic atomic categories applied along with some rules. An example of a complex lexical category that maps a phrase in natural language is given as: **walked - S\NP**. The backward slash indicates that given an atomic category **S**, atomic category **NP** follows. This notation of using a slash (/ or \) along with syntactic category is termed as *Lambek notation.* By employing Lambek notation, complex categories can be derived. Some examples to illustrate these are given below:

- Intransitive verb: **S\NP**□

- Transitive verb: **(S\NP)/NP**
- Adverb: **(S\NP)\(S\NP)**
- Prepositional phrase: **(NP\NP)/NP**

Interaction between categories, semantic and syntactic types are presented in the next section.

A CCG *lexicon* is defined as an entity that specifies all of the categories that a given word can have. Thus, a lexicon assigns categories to words. Combinatory rules then help in identifying the combinations that are valid and not valid. An important mechanism involving combinators and rules is functional application. An example for a complex CCG lexicon along with a sentence from natural language is given as: ***scored maximum* - (S\NP)/NP/(S\NP)\(S\NP)**.

***F**unctional application* refers to the process of combining function with its argument. An example to illustrate this process is given as:

Which	**player**	**scored …?**
NP	**S**	**S\NP**

Two important rules that are used in function application are forward and backward application. By employing forward and backward application of rules in conjunction with combinators, lexical derivations are obtained. Given A, B two lexical categories is given as below:

A/B	B	$\Rightarrow$	A	-	Forward application (>)
B	B\A	$\Rightarrow$	A	-	Backward application (<)

A '/' indicates that given a category B to the left, category A follows the function application. From the example presented above, forward application can be performed when either atomic or complex lexical categories present in the syntax as discussed. Similarly a '\' indicates that given a category A to the right, category B follows. In function application the direction of the slash (either forward or backward) indicates the position of the argument with respect to the function. A sentence from natural language and its corresponding mapping in atomic and complex lexical categories that illustrate forward and backward slashes is presented as below:

played	*matches*	*at*	*Lords*
(S/NP)\NP	**NP**	**PP**	**(S/NP)**

The rules presented above illustrate how lexical categories are inferred. An *inference* in Machine Learning terms is similar to a conclusion. An inference is drawn when it is nearly certain that an event happens. For a given grammar, a formal system can be deduced based on categorical grammars and thereby, derive an expression through parsing it from natural language and produce a resulting tree structure. A ***formal system*** is a well-defined system of abstract thought that is similar to a mathematical model. A formal system is composed of the following entities:

- A finite set of symbols (Also known as alphabet that help building formulae). E.g.: **S, NP, P, PP**
- A grammar that controls how individual and complex formulae are formed/derived. E.g.: **S→NP, N→NNP NP**
- A set of rules known as inference rules that further helps down with the transformation apart from grammar rules E.g.: Modus Ponens, Contraposition, Modus Tollens

An example from the Cricket domain that works by employing the aforementioned rules with deriving the connection between natural and formal languages is given as:

scored the max runs

The categorical type information for this sentence is presented as:

scored	*the*	*max*	*runs*
(S\NP / NP)	**(NP/N)**	**S/NP**	**NP**

The categorical inference drawn from this sentence in natural language by employing backward application is given as:

```
the           runs
NP/N          NP
      S/NP                 <   max          scored
                               S/NP        (S\NP / NP)
                                      NP
                                                          <
                     S
```

It has been observed from the above derivation that given two lexical categories **NP/N** and **NP**, an inference has been made by utilizing the backward application rule to category **S/NP**. In a similar fashion, a categorical inference of **NP** has been drawn from two individual categories **(S\NP/NP)** and **(S/NP)**. Similarly a final inference is made by utilizing backward application rule on individual entities **NP** and **(S/NP)**. Thus categorical inference for prediction has been made as S, proving the face that the model has made a connection to a word in natural language such as What, Who, Which Whom.

A *combinator* acts as a basis for design and development of programming languages. A combinator is to Computer Science, what a variable is to Mathematics. Combinators simplify the usage of variables and help with simplification of grammar. Apart from forward and backward functional application, some common combinatory rules are forward type-raising, type-composition. Some complex lexical categories that employ forward and backward slashes and map phrases from natural language sentences may be given as: **respected – (S\NP)/NP**, **gave – ((S\NP)/NP)/NP**. The approach that is followed within this work with CCG allows for capturing certain kinds of relationships that might not be possible to explore with any other approach. An example to present this methodology is given as below:

What is the name of the player who scored…?

Give me the name of the player …?

Who scored …?

Speaking in terms of natural language, all of the questions mentioned above, point to the same answer. For a machine to be able to effectively understand the same requires it to understand how relations are forged in between words in natural language. For this task, CCG is employed. CCG is chosen and it is formalized to this thesis by using the works of **Elbridge (2002) and Zettlemoyer (2009).** For working constraints the lexical items derived from the parsing mechanism and the grammar would be in triplet form presented as:

word |-- syntactic category : logical form

An example to present this can be given as:

SACHIN |-- NP : player

Where SACHIN is the name of a corresponding lexeme, NP is the syntactic category in this case, player denotes the corresponding form, say, For example: odi, test. The major advantage of using a CCG grammar as devised above is that it erases the boundary between syntax and semantic structure of logical items. The idea that drives this approach is that CCG categories coupled with other information may be used as syntactic arguments in a parse. The order that a syntactic category combines with its syntactic arguments is the same as the order that the associated logical-expression combines with its semantic arguments. The **‘λ’** operator is used to define and relate semantic functions in our calculus. Every categorical grammar thus consists of basic syntactic categories some basic and some complex. CCG elucidated as part of this thesis typically consists of basic syntactic entities or atomic categories and interaction amongst these categories by making use of operators. Any number of CCG queries can then be generated basing on these entities and relations among them. An example to present this:

Let the category of individual unit entities, that is, syntactic categories be {S, NP, N, PP, . . . Let the grammatical relation that needs to be generated be one for an ODI player. This is represented as: **player |-- (S\NP)/NP : x, y odiplayer (y, x)**

After building sentences and examining derivations, a mechanism is needed to parse through the derivations. The Cocke Younger Kasami (CKY) algorithm parses through sentence derivations along with meanings. Presented below is a generalized version of the CKY algorithm that parses through categorical grammars whereas the actual CKY algorithm specific to this thesis is presented in the Implementation section.

Let **w[p,d]** be the substring of w of length d starting from the pth symbol of w. Let **P[i,p,d]** be the boolean array where P[i,p,d] means that x_i derives w[p,d]

Input: Sentence [$w_{0:1}$, . . . , $w_{n1:n}$], Lexicon, CCG combinators

Output: A Filled Chart

1. **Initialize all P[i,p,d] to false.**
2. **For all i from r+1 to m, and for all p from 1 to n, if x_i = w[p,1], then assign P[i,p,1] to be true.**
3. **For each production of the form x_j -> x_i, where j is between 1 and p (x_j is a variable) and i is between r+1 and m (x_i is a terminal), and for each p from 1 to n, if P[i,p,1], then assign P[j,p,1] to be true.**
4. **Execute the following for all d, starting with d=2 and ending with d=n.**

    1. **Execute the following for every production of the form x_j -> $x_j x_k$**

```
            for all p from 1 to n-d+1,
            for q from p+1 to p+d-1,
            if P[j,p,q-p] and P[k,q,d+p-q] then assign P[i,p,d] to
            true.
    2. If P[1,1,n] then return Yes else return No.
```

Algorithm 2.1 The Cocke-Younger-Kasami Parsing Algorithm

A major advantage with CYK algorithm is that it not only determines whether a sentence belongs to the denoted grammar but also if its follows the specified syntax as provided by the grammar. With parsing, it builds up a parse tree from the individual nodes thus completing the parse tree.

An illustration of chart filling with CYK algorithm is presented as below:

Given the sample question that is being predicted is: ***Who scored the maximum runs at lords?*** From the initial step of parses and estimates, it can be concluded that the agent needs to answer the aforementioned query. Chart filling proceeds until all of the available parses are accommodated for. Lets say, for example, six different solutions are available for the above query. The process begins by initializing all of 6*6 cells with a value ‘false’. This is illustrated as below:

X11 – false	X12 – false	X13 – false	X14 – false	X15 - false	X16 - false
X21 - false	X22 – false	X23 – false	X24 – false	X25 – false	X26 - false
X31 – false	X32 – false	X33 – false	X34 – false	X35 – false	X36 - false
X41 – false	X42 – false	X43 – false	X44 – false	X45 – false	X46 - false
X51 – false	X52 – false	X53 – false	X54 – false	X55 – false	X56 - false
X61 – false	X62 – false	X63 – false	X64 – false	X65 – false	X66 - false

Each of the candidates in the chart is a probable parse obtained from the previous step. Practically, X13 represents a CCG triplet of the form: **SACHIN |-- NP : player,** whereas X24 represents another set: **Matthew |-- NP : player.** As CKY algorithm proceeds on filling out the chart, parses that either have feature sets similar to the question or parses that have had a higher probability of co-occurence with semantic category of the question are qualified ‘true’. By applying semantic categorical information to the question,

Who	*scored*	*the*	*max*	*runs*	*at*	*the*	*Lords?*
S	**(S\NP / NP)**	**(NP/N)**	**S/NP**	**NP**	**PP**	**NP/N**	**S/NP**

By virtue of categorical inference it can be concluded that the answer to this query possesses a lexical category of ‘S’. Thus, it can be reasonably concluded that the CCG triplet in one of the cells of the chart contains a lexical category ‘S’ or it has correspondingly similar probability of occurrence as the answer of the query does. So lets say, X33 has the form: **Mc Grath |-- NP : player.** Since probability estimates are already drawn from the previous step, they can be compared and the corresponding cell entry in the chart set to ‘true’.

The process of chart filling as the algorithm proceeds is illustrated as below:

		X33 – true			
					X46 - false
	X52 – true				

Step by step, starting from each cell of the chart, the above process is repeated until all the parses in question are accommodated. The actual process of implementing the algorithm and evaluation of answers to the queries is explained in the implementation section.

2.5 Adaptive Boosting

Adaptive Boosting or Ada-Boost (**Freund, 1999**) is a machine learning strategy used along with other algorithms to improve performance in classifier tasks. A *classifier* may be defined, as an adaptive system that learns to predict the best action based on it previous input. AdaBoost works by combining the prediction of multiple classifiers to produce a single classifier known as ensemble. An *ensemble* is defined as a single classifier that is a collection of multiple classifiers that are individually trained and ensemble’s accuracy at making a prediction is generally more than the individual classifiers that make up the ensemble.

Ada-Boost is adaptive in the sense that it uses the classifier based on previously learned classifiers. Learning phase in Ada-Boost proceeds in a round-wise fashion starting with a first round and proceeding till ‘**N**’ number of rounds, for any given N. Ada-Boost improves its overall performance with each round of classifier data.

A key assumption that ensemble approach makes is that all of classifiers are independent in their predictions this means that no two events or classifiers are related to each other and that the occurrence of one event is entirely discreet to any given second event. A diagrammatic illustration of the ensemble approach is presented as below:

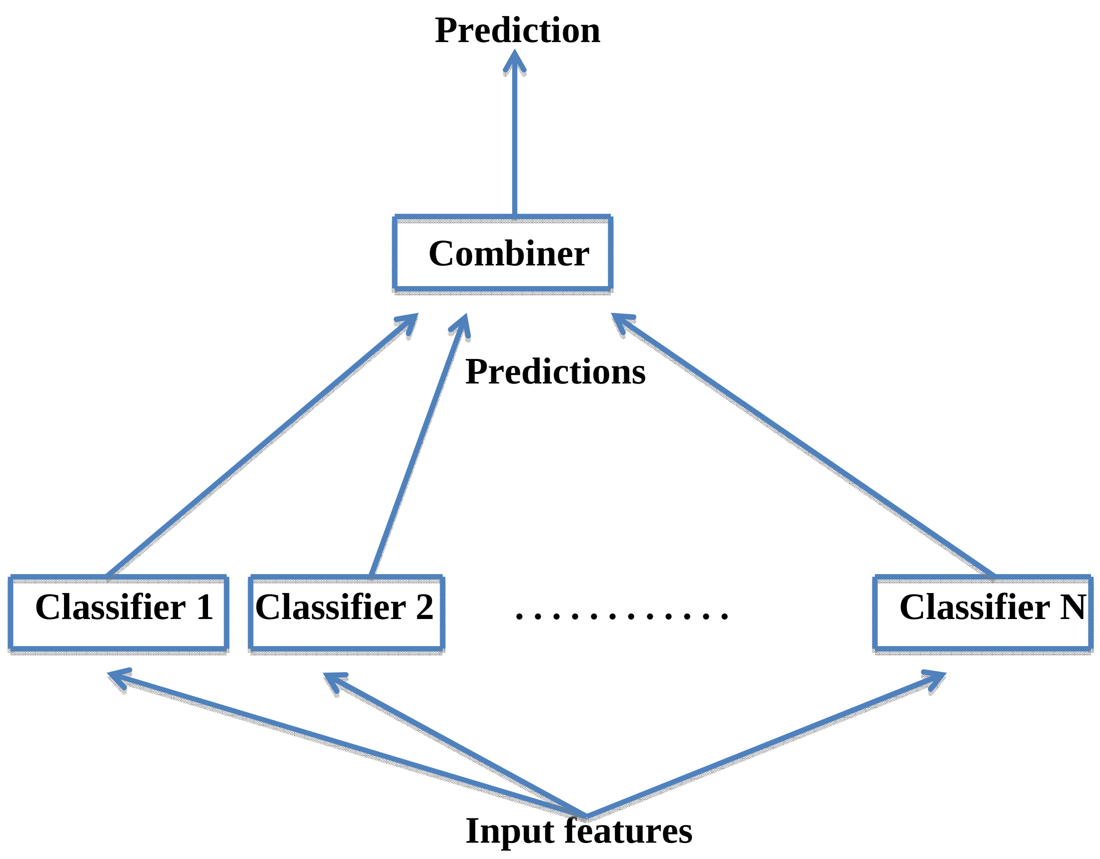


Figure 2.3 Ensemble approach

As illustrated in the figure, ensemble approach uses multiple classifiers to obtain better predictions than each of the classifiers combined together. The advantage of using multiple classifiers is that agent learning that is deprecated from a classifier can be improved via another classifier. Also any given classifier that has shown performance less than its previous round or decrease in its accuracy can be removed. Ada-Boost is utilized when making a prediction involves evaluating more than one model.

Depending on the accuracy required in making predictions, ensemble mechanism consist of a variety of components. Two of the important components are **Learning mechanisms** itself and **Combiners** that are used for including the learning mechanisms. Two important approaches for producing an ensemble are presented as below:

Boosting **(Freund & Schapire, 1996)**

Bagging **(Breiman, 1996**)

A *Boosting* strategy is an amalgamation of a family of learning methods. The main focus of boosting methods is to generate a set of classifiers. The training set used for each of the classifiers is chosen basing on the performance of an earlier classifier in the set. Thus in a boosting approach, examples that have fared poorly by the previous classifiers are picked more often than examples that have fared better. Using this approach, new classifiers are produced to better predict examples that have fared poorly using the current set of classifiers.

For this work a model of Ada-Boost is used that selects an example and weights out error basing on the probability of that example. Thus, in a given set of examples, more weight is associated with examples that have a higher probability of occurrence than that doesn't. The weighing scheme followed and the exact approach is outlined in the form of an algorithm later in this section.

Bagging is machine learning meta-algorithm that is used for improvement in performance within tasks such as prediction and classification. Bagging works by employing a bootstrapping strategy that it uses to generate individuals by randomly picking classifiers from training set. Each of the classifiers training set items is drawn randomly by hand picking them from training set. Bagging is useful when small changes in training can result in large changes while making predictions. This happens when a variety of classifiers that has absolutely no data in common are combined together to form an ensemble. In such a case, the difference in their training data can accommodate for a good prediction, that is, a combination almost always results in a prediction that is greater than the sum of individual components.

A brief description of the overall approach for this work by utilizing AdaBoost is presented below:

In the first step of implementing the ensemble approach, a key task is to provide the model with a weighted distribution, that is, probability distribution estimates. *Probability distribution estimates* are models that give the exact number of times a particular classifier appeared in the data versus the count of the entire datum. In this task, weight distribution is provided and each set is updated for all datasets during classification. With each passing round, the weights connected with the incorrectly classified examples are increased and weights of the correct items are kept the same or sometimes decreased, so as to keep the concentration on the pool of corrected items. Employing such an approach makes it is easy to identify the incorrectly classified items.

AdaBoost also involves a weighting scheme to be followed for every implementation. A *weighing scheme* is a strategy that helps in simplifying the learning strategy by evaluating the classifiers that are strong and the ones that are weak. Ensemble approach in general, helps in converting a weak learner strong. Ensemble approach works this by coupling the performance of the weak classifiers into a strong learner so that re-weighting is performed for compensating the weak learner in the pool of available classifiers. This way, the weak learners concentrate more on the problems that were previously wrong or the ones that the classifier has performed incorrectly. In the approach that was employed for this work, the probability distribution estimates that were provided to the agent, act as the basis of the weighing scheme. The corresponding algorithm that was implemented for improvement in the performance of this work is presented below:

Weighting scheme:

The initial weight considerations for most Ada-Boost approaches start from **1/N**, where **N** is the total number of classifiers. It remains true for this work except that the weight measure changes with every round. Given a nth member of a pool of classifiers,

$$^{dE}{}_{= -W_c e^{-\alpha_m} + W_e e^{\alpha_m}}\ {}_{d\alpha_m}.$$

Equating this expression to zero and multiplying by e^{α_m} we obtain $-W_c+W_e e^{2\alpha_m} =0$

The optimal α_m is thus:

$$\alpha_m={}_{1/2}{}^{\ln}\ {}_{W_e}$$

Remembering that W is the total sum of weights, we can rewrite the expression

above as:

$$\alpha_m = {}_{1/2}{}^{\ln}\ {}_{W} = 1/{}_{2}{}^{\ln}\ {}_{e}$$

Where $e_m = W_e/W$, is the percentage rate of error given the weights of the data points

Now the overall algorithm for this work might be presented as:

Training set: $(x1, y1), (x2, y2) \ldots (xn, yn)$ where $xi \in X_i$, $y_i \in Yi$. and $(X_i, Yi) \in$ domain

Number of iterations T

Initialize D1 (i) = 1/ m; I = 1, 2, 3 … m;

For: t = 1, 2, 3 … T DO:

From the family of classifiers, find the classifier h_t that maximizes the absolute value of the difference of the corresponding weighted error rate $\in_t$ and 0.5 with respect to the distribution D_t :

$$H_t = \underset{ht \in H}{\operatorname{argmax}} \left(|0.5 - \in| \right)$$

$$\in_t = \sum_{I=1}^{m} D_t(i) * I\,(y_i \cong h_t(x_i))$$

where

I is the indicator function

D is the given distribution

If $|0.5 - \in| \geq \beta$ where β is a previously chosen threshold, then stop

```
Choose $\in_t \in$ R, typically $\in_t$ = ½ ln (1- $\in_t$) / $\in_t$

Update – $D_{t+1} = D_t$ (i) * exp (-$\in_t$ * h * β) / Z where i = 1, 2, 3, … m;

where Z is normalization factor and ensures that D will be
a probability function (sum over all x equals one)
```

Algorithm 2.2 Adaptive Boosting

An example to illustrate the above-presented algorithm in the context of a **Q and A** system is given as below:

Assume the following probability distribution is presented at the first step for evaluating the correct answer to a query, that is, *Which team won the world cup of 1992?*

The entities that are presented for evaluation are given out as:

- 'The' with a probability of 0.02
- 'Kenyan' with a probability of 0.04
- 'England' with a probability of 0.20
- 'India' with a probability of 0.34
- 'team' with a probability of 0.45

Its evident that words 'team' and 'The' are not a solution to the query. AdaBoost aims at improving the performance of words like 'India', 'England' and 'Kenyan'. Lets say, to identify the above word fragment from weighted distribution is nearly impossible for a machine. To help it with the judgement process, AdaBoost helps in identifying the correct classifiers and figure out a solution from the given choices. Thus, Ada-Boost can be termed as a meta-algorithm. A *meta algorithm* is an algorithm that acts along with other algorithms in conjunction with them towards improving the performance. In the example presented above, depending on the learning algorithm that is followed and information obtained from further estimates, weights for each of the words may be decreased or increased for further processing. The implementation details for this work follow in the next chapter.

3. Implementation

This chapter covers the implementation details for this work. The first section covers the overall approach used for this work. The second section explains how semantic decomposition is used in this thesis. The third section covers how sentence derivations are generated using semantic parsing and how parse trees are filled. The third and fourth sections deal with how these parses are evaluated. The fifth section explains how lexicon expansion is proposed within this work. The sixth section deals with building a learning model for this work.

3.1 Overall Algorithm Used

A step-by-step procedure of the method to this work is presented as below, along with a detailed example of each step following that:

Input:

- A training set consisting of formal sentences from the cricket domain paired with its logical/meaning representation

- Lexical parameters such learning rate constant, cooling rate parameter

Output:

- Updated lexicon
- Linear modeled parameter vector
- Calculated scores: Recall, precision and F1 (Arithmetic mean of recall and precision)

Algorithm:

1. Extract lexical features that score individual items
2. Perform initial parse for all sentence logical-form pairs
3. Obtain semantic features that contain the logical form from initial parse
4. Initialize the parameters based on the set of obtained lexicon from parsing and initial scores
5. For all of the sets of features sentence pairs perform: □

    5.1 Expand the lexicon by adding lexical items to the current vector□

    5.2 Update the current lexicon from the set of added items □

    5.3 Update parameters from evaluated entries

6. Using the previous steps calculate the gradient-based parameters such as learning rate,

cooling rate

7. Repeat steps 6 & 7 for entire set

8.1 Update received parameters that maximizes the conditional likelihood in the given space

8. From initial parameter weights and co-occurrence counts obtained from the previous steps, match each word pair to fit a semantic entity returned from step 5.3

9. Select the predicate from the predicate variables that maximizes the log likelihood for all of the states, types present

10. Eliminate the negative outlier data from the pool of classifier data using Adaptive Boosting

11. Calculate the final parameters recall, precision and the arithmetic mean of recall and precision (F1)

An example to illustrate this algorithm step-by-step is presented as below:

Step 1: Starting with step 1, the following query is subjected to processing as.

Which team won the most number of matches at the Lords?

The query is one of its kind and was not encountered previously by the agent. Prior to this, the following items have been added for learning purposes so as to allow it to learn and make good predictions:

- the Indian team

- the English team
- the South African team
- the Australian cricket team
- Adam Gilchrist
- team
- bowler

All of the above items have varying levels of probability and occurrence counts. The initial step makes an estimate of how many times each of the core words occur. A *core word* is an important keyword or term that can be very useful and one that carries considerable amount of semantic detail with it. This is also known as a feature in Machine Learning terms. Repetitive terms such as *the, an, a,* and *or* are skipped and assigned no weight.

Step 1: After these words are extracted, an estimate of the counts of these individual words is made, that is, to give an idea of how many times these terms have occurred in the entire training corpus. It is then subjected to a broad variety of strategies collectively called as decomposition. To be able to effectively answer a query we need to know what the question is about. Question Decomposition helps us in this aspect. *Question Decomposition* breaks the question into smaller and smaller chunks so as to understand it better. In context to the query that is being evaluated, Question Decomposition helps in extracting a core word from the question as:

Which team won . . . . . ?

Step 2: Purging out all words except the core words that is,

- Indian team
- English team
- South African team
- Australian team
- Adam Gilchrist
- team
- Bowler

From step 1, it can be inferred that the lexical item in question here is **S**.

Step 3: With each of the chunks from the question extracted, we have more information about the question that is being posed. At this stage, the individual counts corresponding to each of the items are obtained. These counts are also known as *corpus counts*. This is a measure of how many times the aforementioned core words occur with respect to the entire training data. The counts for the corpus presented within this example may be given as:

English team	-	0.008
Indian team	-	0.0003

Australian team	-	0.00070
South African Cricket team	-	0.000909

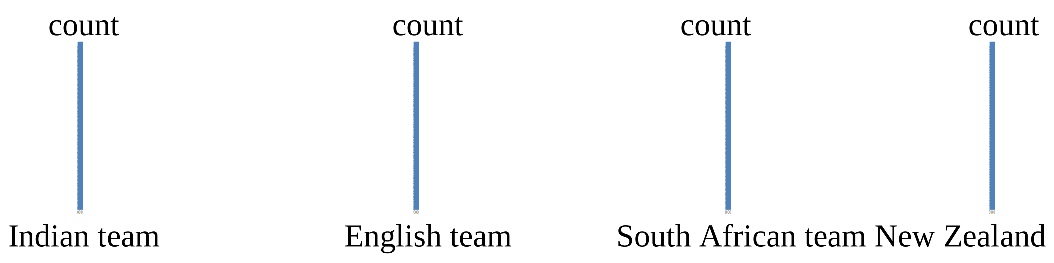
By employing lexical categorization from the previous step, the lexical category in question has been identified as ‘**S**’

Step 4: Proceeding further, we have a few nodes of the initial parse tree filled and lexicon expansion needs to be performed for evaluating all of the available parses. Some of the available items that are obtained at this step are:

New Zealand team, Irish Cricket team, Kenyan Cricket team, Rwandan team that are all possible answers to the lexical item that is being evaluated, that is, S. All possible lexeme extension is performed at this step so as to evaluate the best answer.

Step 5: From query decomposition, it is learnt that we are interested in a team that has won the highest number of matches at the Lords stadium in England. This confirms that the subject in question corresponds to a lexical category of **S.** Steps further help in generating the corresponding SQL denotation for this query.

Step 6: At this stage, sub trees corresponding to derivations and denotations are all filled with respective counts and we have a clue as to what the expected answers syntactic category is. Filling out of the sub trees and merging of all of the sub trees is presented as:



Step 7: At this stage, the words and phrases obtained are evaluated as to match against their respective categorical type. Thus eliminating most of obvious choices, we arrive at: the Indian team, the English team, the South African team, the Australian cricket team, New Zealand team, Irish Cricket team, Kenyan Cricket team, Rwandan team all of the possible teams playing matches. Starting with the pruned nodes in the sub tree, an inference is made.

Parses are processed further under two instances: a lexicon has a similar co-occurrence count as the category itself. The second instance is when the categorical information extracted from the lexicon stand as a reasonable proof that the two words are similar. This is further explained in Chapter 3.2, 3.3

Step 8: Again corpus counts for the core words are obtained, this time from the entire training data. A probability distribution is drawn similar to Bayesian inference and is retained for further processing. Meta-algorithms are utilized to improve the performance of the existing classifiers further more.

Steps 9,10,11: From the counts obtained and features selected, the correct answer is chosen at his juncture and other parametric data such as learning rate, cooling rate are calculated to evaluate the learning. Semantic Decomposition that forms a major aspect in question decomposition is presented in Chapter 3.2.

3.2 Semantic Decomposition

Semantic Decomposition is an indexing and retrieval strategy that is used in identifying and understanding relationships within an unstructured collection of data. An underlying notion, that drives semantic decomposition, is the fact that words are used in a similar context to their meanings. Thus semantic decomposition helps in extracting and mining information, conceptual context from a given stream of text by associating it with words, terms that occur in a similar sense. Techniques such as Semantic Decomposition, Singular-Value-Decomposition form a broad category of techniques known as *Latent Semantic Analysis*. Semantic decomposition can be useful to correlate semantic relations within a pool of unclassified and semi-structured datum.

As part of this work, propositional logic is made use of to build a system that is similar to **Zettlemoyer and Collins (2005),** Semantic decomposition is utilized in building up a chart of filled trees that can be further divided into respective nodes filled out using parses. A filled tree is an entity such that a tree structure is laid and lexemes are filled as tree nodes and leaves. A *lexeme* is a word extracted from a sentence and can be a part of it only a single word. This work begins with an initial lexicon and a probabilistic model with initial round of parses obtained with the training data. Derivations of the data take up a standard form throughout this work: (sentence, logical-form). To begin with, a probabilistic model is estimated with counts obtained from the training set. All of this input is fed to a generator function **G (s, f)** can be mathematically expressed as:

Given the next lexicon to be calculated as D (derivation), For any given sentence, logical form pair s, f respectively,

$$D \quad = \quad G(s, f)$$

The generator function repetitively proposes for all possible parses while estimates are made to calculate how good the parse can be. An incremental approach is followed thus for making the first set of parses and making predictions. As described previously, all of the combinators use function calculus to help with semantics of the given input categories. Each of the parses obtained from this phase can be referred to as a single split or parse. Given a query:

Who scored the maximum runs in world cup of 1996?

Generator function helps in estimating all of the possible splits for a query. Logical forms corresponding to the given query are brought into picture that may be presented as:

Count (Max (runs)) and match (world cup (1996))

Such representations allow for simulating a structured query language in similar lines to a formal language such as SQL. Then each of the individual items is subjected to further processing by the probabilistic engine and further estimates are made on the obtained logical forms. As discussed earlier semantic decomposition is used along side with logic induction for building and filling up parse trees will be discussed in the coming sections. Semantic decomposition is thus used in this along side with generating tree derivations and filling out the nodes. The primary objective here is to be able to substitute the role of filling for the sub trees present.

An illustration for the tree filling might be presented as below:

Consider the tree node to be filled out as $T_{i,}$ Picking an example from the above cricket query as max (runs) **and** match (world cup (1996)), T_j, T_k are two sub trees filling out from the nodes. Initially the nodes of the tree are worked on, one at a time starting with the root node that is, T_i. The operation performed can be represented as:

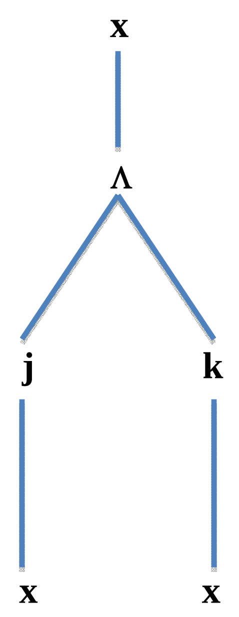


Figure 3.1 Tree Representation of a Logical Form

The above figure presents how a logical form that supports, a specific domain based query cricket can be tailored to fit into a tree structure. Any higher order logical representation can be turned to resemble this. In the diagram referred to above, the leaf node of the tree contains a relation directly to its upper node.

The above diagrams illustrate how query processing is handled part-by-part. Initially the node is empty and by making use of previously obtained derivations, counts a decision is made for each of the individual nodes. Then each of the obtained sub trees is merged until all parses are completed. An algorithm may be presented to illustrate this process as below:

Algorithm: Build and fill sub trees

Input: A single node with n number of children in the original form

```
Output: A new sub tree with substituted nodes

1. N  =  Ψ where Ψ is the old sub tree node with no or empty parses

2. N = { }; Initialize begin for all parses. Set parameters: cooling rate,
   n.o. of runs

2.1 for 0 to (n-1):  (a, b) = Ψ where a, b represent the logical  form

    2.2 if Match (a.N, a. Ψ)  and (a.N == b. Ψ):

            set-parameters (a, b)

            N.a = N.a ∪ a. Ψ

            N.b = N.b ∪ b. Ψ

    2.3 else: N = { }; k = (n-1)

    2.4 for 0 to (k-1):

            for all children in parse k:

            set-parameters (Ψ(k-1), Ψ(k-2))

            N.a = N.a ∪ Ψ(k-1)
```

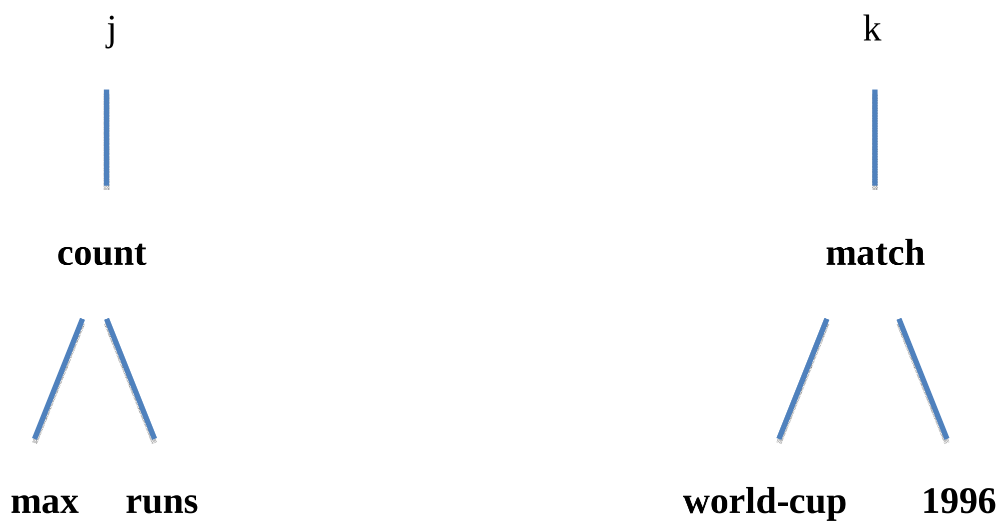
```
                    N.b = N.b ∪ Ψ(k-2)

    3. Repeat step 2 for all parses available

    4. Return ( N.a, N.b )
```

Algorithm 3.1 Build Sub Trees

The above algorithm depicts how sub trees are merged from individual parses and how a decision is made in choosing them. An example to present this approach is given below. Given a number of splits, nodes corresponding to the tree are chosen as:



From the input that is obtained after dividing the question into smaller chunks, we have obtained the above sub trees. In the next step the initial lexicon is chosen and the union set of lexicon with all available parses is considered. In the example presented above, count **(max (runs)), max (runs)** is deemed as one of the probable parses. In a similar fashion some examples include **min (runs), num (players), avg (runs).** Thus given an initial lexicon and a sample-weighted distribution corresponding to it, semantic decomposition proceeds by examining and predicting each probable parse similar to the initial lexicon.

The results of this step that have a similarity to the initial lexicon are thereby carried forwarded to the next step of building sub trees trees. Evaluations are made again while filling out the sub tree so as to ensure that all probable parses are thereby represented as nodes in the sub trees. Later all of the obtained sub trees are merged into forming a tree with all of the nodes filled.

3.2 Generating Derivations from Semantic Parsing

As part of this work, we are interested in learning and implementing probabilistic grammars that could parse sentences onto their logical forms. As outlined in section 2.2, logical forms consist of a CCG lexicon, CCG schema necessary for driving this and some derivations that could result from it. Different parses are then chosen basing on the sentence. Making use of these different parses thus completes the sentences. The entire learning mechanism will be dealt with by using a series of sentences of the form (sentence, logical form) similar to the lines work by **Elbridge and Zettlemoyer (2009)** Thus grammars are induced as part of the learning strategy. Then a probabilistic model is built from a large set of these input derivations by taking all allowable pairs of the input. The mechanism that is proposed here is known as the Inverted CCG hierarchy and is one that is used by **Hillrenger (2002).** The operations used may be mentioned as below:

Table 3.1 Inverted CCG Combinators

Inverted Forward Application □X : f(g) => X/Y:f Y: g

Inverted Backward Application □X : f(g) => Y:g X \ Y: f

Inverted Forward Composition □X / Z : f(g(x)) => X / Y: f Y / Z:

Inverted Backward Composition □X \ Z : f(g(x)) => Y \ Z: g X \ Y

Often times Inverted Forward and Backward application are made use of for generating the forms. Each of the combinators mentioned in above correspond to recombine and factoring both the syntactic category and logical expression involved.. We propose to do this operation using higher order unification. The idea of employing higher order unification has been brought forth from the CCG derivations, was started by **Zettlemoyer and Huningghake (2006).** An individual semantic decomposition while obtaining derivations may be referred to as a split. After generating individual splits from the sentence and other input parameters, all of the generated splits are integrated and a sub tree is built. Later many of these sub trees are clustered together and a tree structure is formed.

As part of deriving the right form of the proof for statements we make use of various propositional forms in our derivation. Some of the various valid propositional forms may be given with a brief description as below:

Modus Ponens: If A, then B

A

Therefore, B

Apart from the valid proposition forms that were described above, we also base our derivations and formal proofs for it on rules of inference**.** Any formal system or propositional axiomatic schema must make use of these rules of inference for driving the derivations. The above system makes use of a rule of logic called modus ponens. In relation to our work we concentrate our attention on systems that are related to a question and answer (Q&A). By making use of the above said derivations and logical forms, a formal learning mechanism is developed in similar lines to a **Brody (1987)** system is developed. A structure similar to a tree is then built making use of the Q&A system mechanism. Starting at the root node, the data structures are filled with the logical forms and parses obtained from the CCG. Initially root node is first filled with the logical form and the syntactic categories. Filling out of this tree plays a crucial role in generating the required forms and this process continues till all of the nodes present in the tree are accommodated for. This process might be depicted as below:

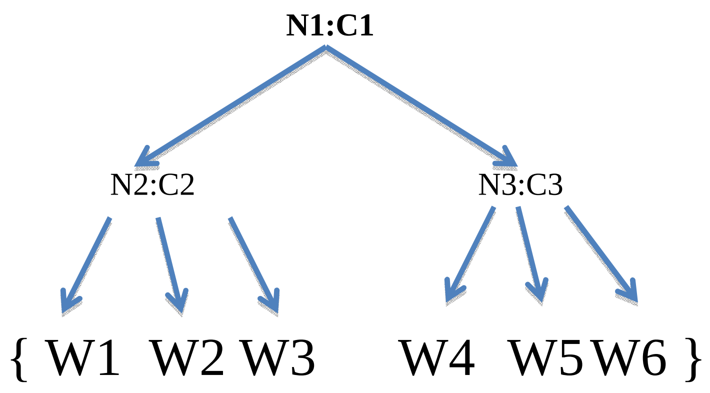
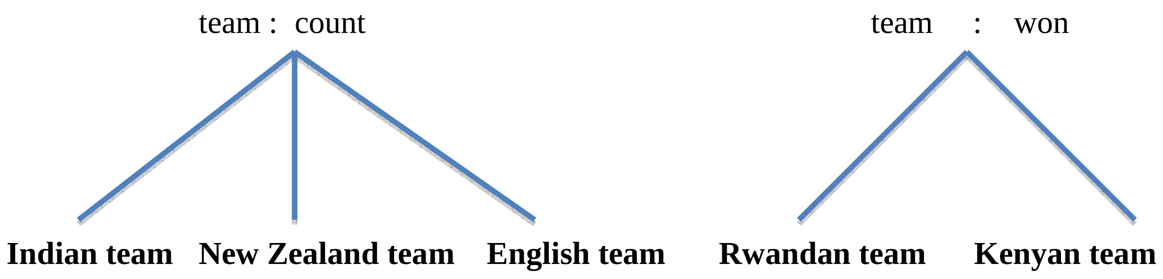


Figure 3.2 Parse Tree with Word Predictions

As depicted in the figure, this work proposes a method to map semantic type to syntactic category, that is, word predictions are made basing on this split. Essentially while filling out all of the tree nodes, a label is made use of for each possible node in the tree. This mapping with an example is presented as:



With each of the word splits obtained from the sets, a solution for the parse is proposed by recursively splitting all of the nodes in the tree. Splitting continues until the best solution is reached. When finding out the best solution from the obtained parses, work carried out by **Zettlemoyer (2009)** proposes to consider only a single split of the obtained nodes. This work instead chooses splits from all the possible nodes in the parse tree. Thus splits for words are obtained by generating all splits and by using Semantic

Decomposition. The necessary function mappings for obtaining this in this work may be given out as follows:

$$\{ (N1 : C1), (N2 : C2), (N3 : C3), \ldots \} \; =\!\sim \; \mathbf{S}\,(N : C)$$

The mapping function here is the same approach that was used to fill in parse nodes. This approach thus helps in making out the correct predictions.

3.3 Generating Sentence and Logical Form Derivations

As outlined in the previous chapter, within this work, semantic parsing is used to generalize a specific domain related query base to a predictor upon learning from related domain. In this chapter the (sentence, form) pairs obtained from the previous step, more predictions are made and non-logical relations are made between the involved items. Logical expressions and relational items obtained through parsing are made use of to create a simulation model for filling out further of the tree nodes. The nodes obtained from the parses are again purged after checking with relational constraints, that is, how closely related in a particular context is the obtained parse to the initial lexicon. Thus a node-by-node substitution takes place starting with the root node. In this process, only the items that are really close to the actual meaning of the obtained parse are retained eventually leaving the bad pool of classifier data out. An algorithm for the same might be presented as:

Algorithm: Substitute tree nodes

Input: A given node α in a tree with n children

Output: Set of pairs N to be filled out into new nodes represented as

```
        (ai, bi) ∀ i = 0, 1, 2, 3, . . . .n .,

        // (a1, b1), (a2, b2)
```

Algorithm:

```
1. Initialize step N = { } ;

2. if n = 0:    ai = a; a.targetnode = ai;  N    =    N ∪  { ai };

   bi = b; b.targetnode = b;         N    =    N ∪  { ai };

3. else:

    A                         =                    { [1], [2], [3], . . . [n]  };

4. for i = 0 to (n-1):

      N[i].targetnode = NULL;

        //for each node starting with root

5.      A                     =                Build and fill sub trees (N[i])

6. for j = 0 to (i-1):

        setarguements (count(ai), count (bi));
        //Augmenting the additional parameters within lexicon

        $N = N \cup N[i].targetnode;$

        Return (ai, bi);

    7. End
```

Algorithm 3.2 Substitute Tree Nodes

As depicted in the algorithm above, the substitute nodes algorithm is used to purge out unnecessary denotations. The set arguments are used to obtain the number of available nodes before hand so that they may be used to create node levels for the data structure before hand. The processed output obtained from building sub trees part is made use within this algorithm. Instead of doing a complex task altogether, it is advisable that a divide and conquer strategy is employed and each individual problem is handled effectively. The filled out tree structure obtained from the previous step is then sorted. These denotations thereby are subject to further processing and estimates. An important thing here is that the order in that the denotations are fed to the build sub trees step is retained and the same order is used while purging out outlier data. An example is presented as below:

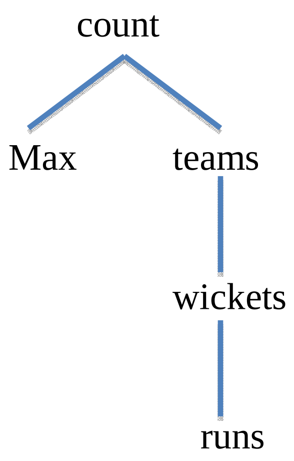
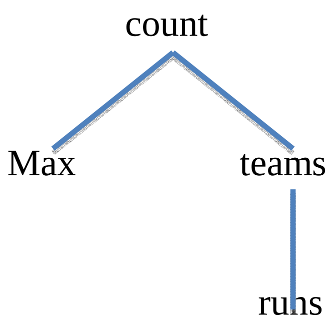
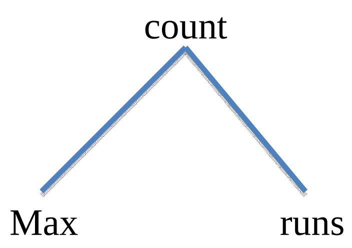


Figure 3.3 Node Prediction Within a Single Parse

As illustrated in the figure, a number of leaf nodes exist at a given level within the sub tree. The tree depicted in the figure was obtained directly from the build sub trees part and was never subjected to any substitution or node retrieval mechanism before. Basing on the previous estimates and a set of initially chosen feature sets, one of the leaf nodes needs to be selected as final node and rest of them eliminated. After a round of initial parameter initiation, the node prediction algorithm presents itself on the sub tree and reduces one of the levels of the given tree as below:



The algorithm proceeds further and the next level of the sub tree is worked upon that reveals its default structure as below:



As understated above, the algorithm proceeds for each of the nodes and each of the sub trees until all of the grammatical denotations are accounted for. In certain cases, all of the nodes need not be replaced, thus retaining possibly some of the lexemes. If any of the children nodes pertaining to any of the sub trees are empty, node replacement uses the lexeme it has obtained from the previous denotation. This step is especially useful when the logical form of one of the grammatical denotations is missing and just the sentence in an original query language is presented as input.

3.4 Lexicon Expansion via CCG Category

Based on the output obtained from various phases, this step proceeds towards proposing an expansion for a given lexicon at a given time. The processed output obtained from the previous steps is made use of for generating complete sentences in this step. Lexical entries for individual meanings and forms are proposed by mapping each of the categorical entities obtained with its original meaning forms, thereby representing the best possible node in the parse tree. Take, For example, the following set of queries:

1. *Who is the captain of the Australian cricket team?*

the agent is fed with the correct answer, say, Ian Bobell. Again starting off,

2. *Who is the coach of the South African cricket team?*

At this stage from the input the agent received from the two sets and parses, it can co-ordinate the difference between the South African and Australian cricket teams. Supposing the answer for the second query to be Sean Pollock, the prediction parameters for individual words arrived from breaking down sentences are calculated. This process of breaking down these full natural language sentences may be termed as the *tokenization* step. After the tokenization step, chunk of individual character stream making up the sentence complete is obtained. The mathematical probability estimate necessary for driving this work is based on Bayes law of conditional probability. Generalizing this work it may be presented as:

$$P(S, \Phi/D) = \frac{P(S, D/\Phi) * P(\Phi)}{P(D)}$$

where S denotes the set of all productions, D is the training data, Φ is the model parameters used as feature sets for the calculation

An example to describe this process of mathematical predictions and estimates obtained at this step may be presented as follows:

the Australian team	~	Ian Bobwell
the South African team	~	Sean Pollock

At the second stage of the model after the two items have been fed and successfully parsed, estimated concerning the same may be presented as bellows:

Ian Bobwell	~	0.049

Sean Pollock	~	0.0071

From this step, the second stage of parsing and pruning begins for making further estimates. All of the obtained lexemes are processed again in a bottom up fashion. From this step the pruned parse trees are again filled making use of the parse strategy. As part of this step CYK **(Cocke-Younger-Kasami)** algorithm is used. Parsing proceeds by recursively filling out a chart data-structure in all cells. Thus a word-parse structure for each entity is obtained. A pseudo-code algorithm that illustrates this process in its entirety may be given below:

Input: Sentences along with denotations s = {w1,w2, w3, . . . wn}; along with category: cn

Output: A filled chart along with final lexicon from prediction model, ∇

Algorithm:

```
1. Initialize step:    Chart    =    {}{};
2. Chart [1] [n-1]   =    cn
3. ∇    =    ∇    X    {w1, w2, w3 . . . wn}    ⊥  {cn};
  //Generating a cartesian cross product of sentences forms and initial lexicon
4. for i    =    0 to (n-2):
        for j = 1 to (n-i):
            for x $\in$ Chart [j] [i]:
                Chart [j] [k] = Chart [j] [k] $\cup$ {ci}
                Chart [j+i] [k+1] = Chart [j+i] [k+1] $\cup$ {ci}
                $\nabla$ = $\nabla$ $\cup$ Chart [j] [k]
                $\nabla$ = $\nabla$ $\cup$ Chart [j+i] [k+1]
                // Do until all combinators are used
5. return $\nabla$
```

Algorithm 3.3 CYK Algorithm

In the above algorithm, initially the chart is initialized as to accommodate for the forthcoming predictions. Initially the beginning sentence along with all of he words comprising it is fed as an input to the algorithm. The major step in this algorithm is obtaining the Cartesian product of the sentences as to form more sentences and individual words. The Cartesian product is obtained in such a way that all possible combinations of the word predictions are made. Only after an exhaustive prediction is made from the available word sets, the best possible lexeme as obtained form the algorithm is passed on to the next stage for further processing.

As outlined previously the training and testing methodology is carried out on a uniquely developed cricket database. This database is composed of 26 sets of queries that consist of a sentence from natural language coupled with its meaning representation from SQL language. To kick-off the training process, this model is fed with a query along with its SQL representation. Two sets are present within the database, one of the sets form the training set and the other set is used for testing. Type representations are made use of, to test the effectiveness of the queries. This means a single query in the SQL formal language may be expressed in different forms in the natural language. For example,

Who scored the highest runs in the match between India and England held at the Lords?

is just the same as:

Give me the team who scored the maximum runs between India and England at the Lords?

is also similar to:

Name the player who hit the maximum runs between India and England at the Lords?

is again similar to:

Obtain the name of the player who scored the maximum number of runs in all of the matches played between India and England at the Lords?

For the above three queries corresponding SQL representation may be the same that is presented as:

select **player** where **player.runs** = **max (runs)** and **venue** = lords where **teams.played** = India and **teams.played** = England.

Thus type representations can be useful in capturing all possible meaning representation available within a given parse. Query sets include queries with varying levels of complexity. Making use of certain primary attributes and associating them with some primary relations, a formal language is generated in similar lines to SQL, predicate logic and so forth. For example, in the above SQL notation, '**venue**' is a secondary attribute used to augment the primary attributes. The SQL representation of the natural language sentence can be inferred to be consisting of both terminal and non-terminal symbols. The non-terminal symbols in the statement represent the attributes that lead to the correct answer. Consider a natural language query:

How many fours did Saurav hit?

The corresponding SQL notation for it may be presented as:

Select ***fours*** *from* ***odiplayers*** *where* ***team*** *= India and* ***player*** *= Saurav*

A simple illustration of a parse tree built from this work may be given as follows:

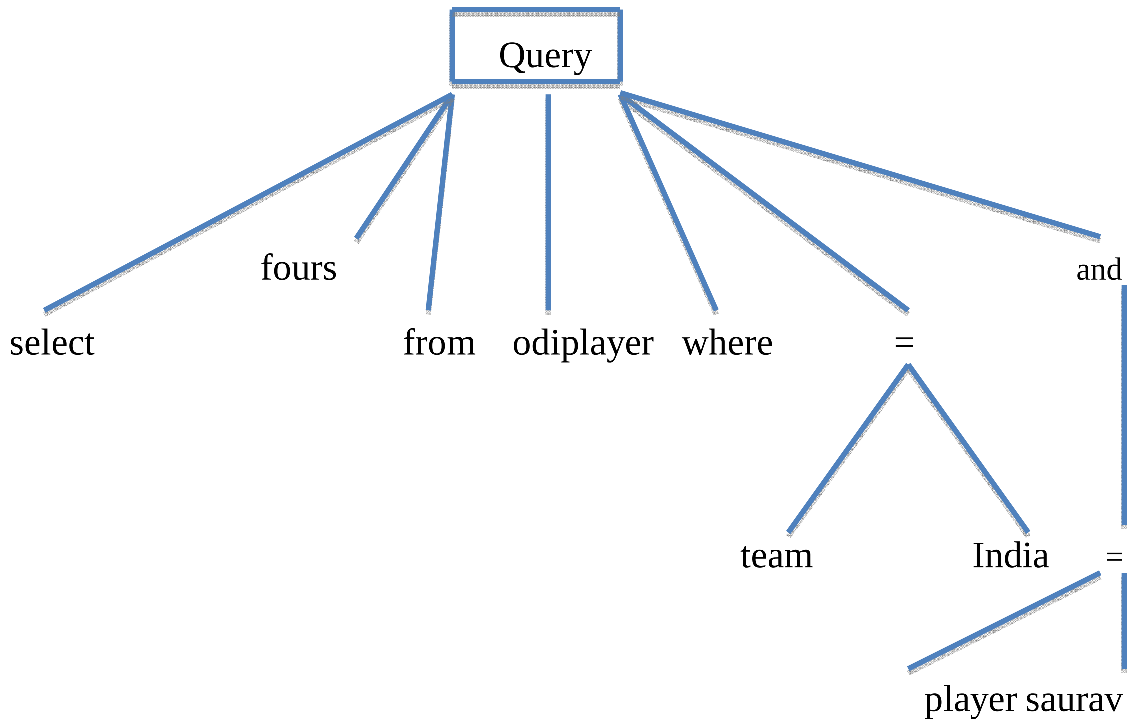


Figure 3.4 Parse Tree for a Cricket Query

As depicted in the figure above, the parse tree depicts terminals, non-terminals and leaf nodes. Basing on the relations exhibited by secondary and primary attributes, a SQL statement might be sub divided into many more possible levels. A generalized SQL statement might be formalized thus as below:

*select **SB** from **FB** where **WB,***

where **SB** denotes select body**, FB** denotes from body**, WB** denotes where body

3.5 Building the Learning Model

The previous chapters give an insight into how knowledge representations are built, how these representations are used along with logical denotations, how the task of building an agent capable of making predictions. This chapter presents the ways and means to integrate this work. Coupling most of the work done so far, this section presents the strategy of how the learning model is built for this work. To understand this, a step-by-step procedure is given regarding CCG and how it is used as part of this work. This may be presented as:

Input: A dataset consisting of (sentence, logical-form) pairs along with a mechanism for bootstrapping this.

Output: A lexicon and a vector with due weights calculated

Algorithm:

1. Update initial lexicon from a pair of sentences coupled along with its logical forms.

2. For all the set of sentences in the given training files, build lexicon from the updated set and identify the best parse.

3. Repeat until all parameters in the set stand updated.

4. Basing on the number of iterations performed, learning rate and □cooling

rate (tentatively set to 10), estimate new parameters.

5. Calculate feature vectors basing on these estimates and update the parameters with obtained values.

6. Improve the lexicon prediction by making use of all parameters and by increasing the conditional likelihood.

7. For ambiguous models, update parameters to check improvement in likelihoods.

8. Output the respective lexicons and parameter vectors basing on the estimates.

Algorithm 3.3 CCG Algorithm

As described in the algorithm, combinatorial categorical grammar plays a key role in building a working model for the problem to be solved within this work. From the estimates made using all of the previous steps, using the lexicon probabilities, the best possible answer to a query is obtained. As understated previously, feature sets used for building this mathematical model are dynamic, that is, at each step of processing, processed output is fed as input to the next level. As the output from this stage is obtained from the feature sets extracted in the previous step, these results are fed into the next step. At this stage, a technique known as adaptive boosting is employed to remove any of the outlier data present in the denotations. The lexicon prediction model that was built till now is again subjected to usual outlier removal technique

4. Results

This chapter presents the results obtained by testing on a custom cricket domain based database queries. The chapter is further divided into three subsections. The first section covers an aspect of the database built for this work and how it is employed in testing. The second section gives a brief comparison of results obtained from CCG with this work. The third section gives out the actual results obtained with this work.

4.1 Query Predictions and Evaluation

This work proceeds with testing by employing a novel cricket set. The database consists of 26 sets of queries with some original grammatical denotations for the same. Initially this database is divided into a training set and a testing set. In connection with this work, we have tried to address the following series of questions:

How effective would an agent work in addressing the problem of question answer prediction within a finite domain?

How to improve the performance of agent in making effective predictions?

How good is the rate of improvement in performance with effective training?

What is the most feasible training paradigm for such a work?

What is the baseline threshold for this learning strategy?

How does the agent respond under really tough situations?

How much data is needed during the training phase to achieve significant performance?

With an increase in training data, how much performance improvement can be expected?

Starting off with the training set, each of the queries is fed as input for this model to begin the training process. After learning from the training sets, the agent is then subjected to testing set for accuracy. For testing the accuracy of this work, N-folded cross validation is employed. The idea with cross validation is that it gives out an unbiased estimate of generalizations made in any model. Thus it is used as a model selection strategy. Here the variable N is specific to the work being carried out. Cross validation randomizes the entity to be selected for any particular iteration. The entire experiment is then repeated for N iterations. Each run of this strategy may be termed as an individual fold. As outlined previously the entire dataset in each of the given folds is sub-divided into a training set and testing set. Firstly the training set is fed as input to the model for enabling it to make estimates on its own from the calculations. Thereby the training set helps in making out its own predictions from the estimations drawn from previous sets. The interesting thing about employing this strategy is that the accuracies of the prediction model increases manifold with respect to the amount of training data it is given.

By the end of this stage, the model has a handful of individual sentences and then a stream of words making up the same. As previously stated, the accuracy for models tends to increase with an increase in the amount of training received. A plot is presented below to indicate the improvement in percentage while increasing the number of folds:

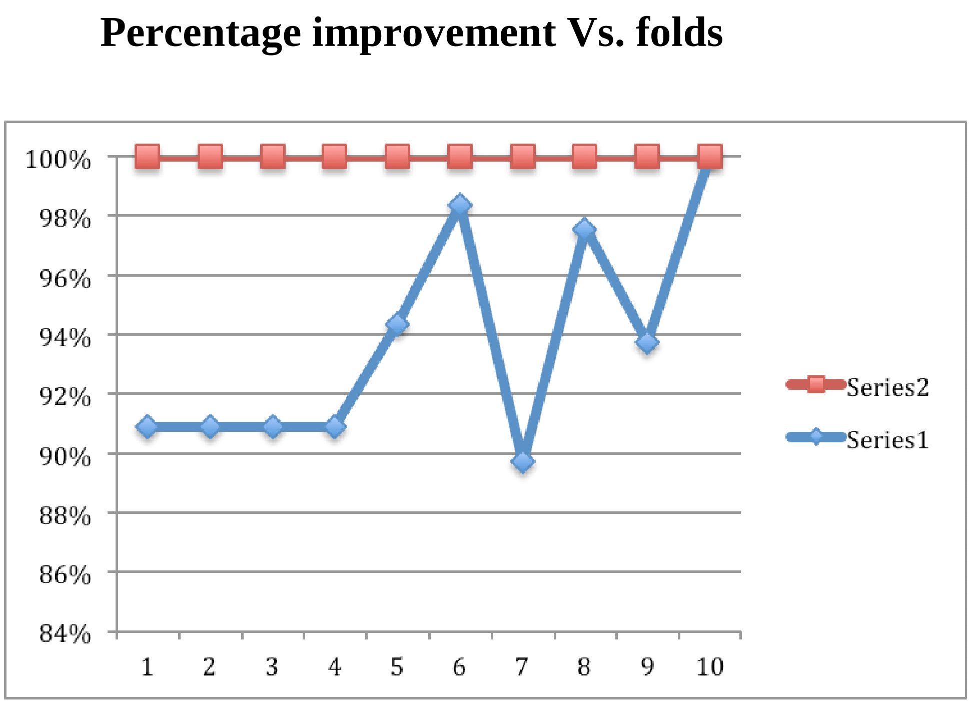


Figure 4.1 Accuracy of N-folded Validation

The plot indicates how the learning strategy devised by this work performs against a series of ten datasets. It also presents a comparison of improvement in accuracy with N-folded validation. It makes the fact clear that with every other iteration, the accuracy improves and in some cases the performance does decrease steeply. After attaining a baseline threshold, it may be seen that the accuracy reaches a near top 100 percentile. Although this is technically not possible, the assumption here is that it attains a near 100 percent improvement. Recent work by **Zettlemoyer (2009)** has been successful in predicting answers to queries basing on geographic sets known as Geo sets. For this work, each of the cricket sets that were built are sorted into 26 queries of 10 sets each and

26 sets of 10 queries each and the marginal median performance of each individual sets starting with 10 set based evaluation may be presented as under:

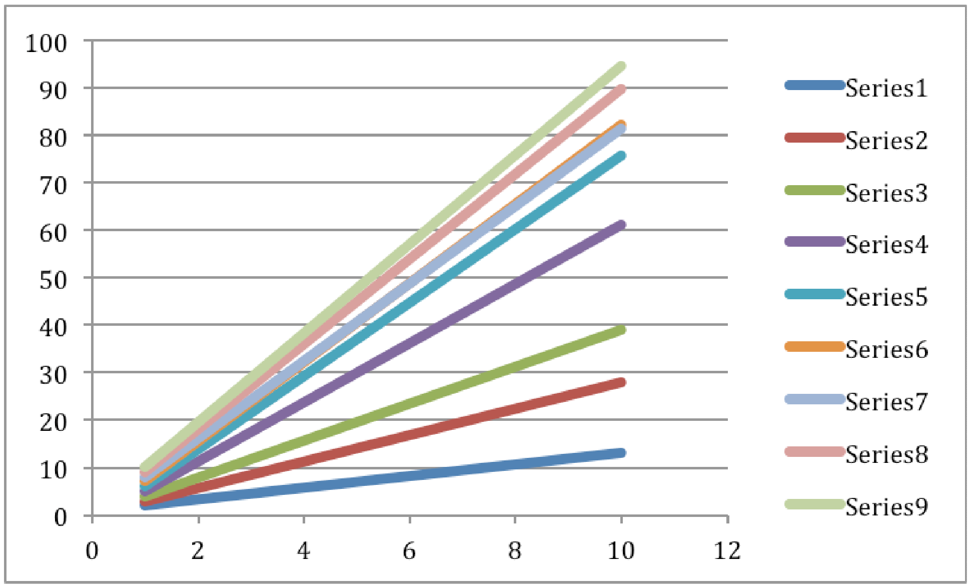


Figure 4.2 Percentage Improvement vs. Sets

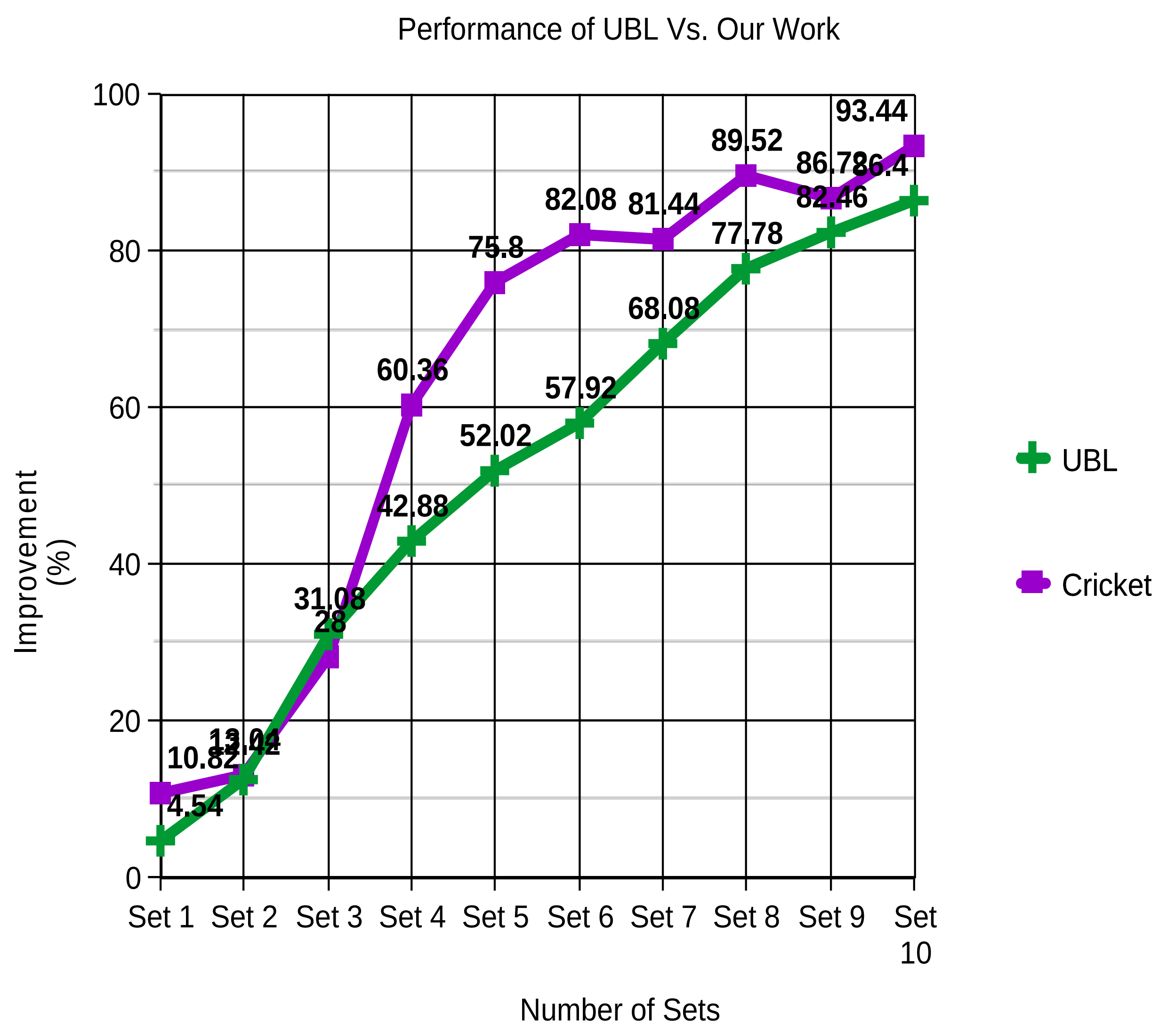
In the figure as shown, there is a consistent increase in the performance of the model. With each increase in set, it can be seen that there is a consistent increase in the accuracy levels. The corresponding numerical data may be presented as under:

Table 4.1 Improvement in Performance

Number of Sets Trained (1~10)	Percentage Improvement (%)
1	10.82
2	13.04

3	28.00
4	60.36
5	75.80
6	82.08
7	81.44
8	89.52
9	86.72
10	93.44

The baseline performance that was obtained here compared to that of **Zettlemoyer (2009)** is presented below:



As depicted in the plot as above, the agent does a fairly good job when compared with the previous works. It should also be noted that after a certain baseline performance is attained, there can be a slump in performance as well. In the plot this can be depicted at datasets 2, 6 and 8. The numerical data corresponding to this plot may be presented as follows:

UBL	This work	Improvement in %

4.54	10.82	6.28
12.42	13.04	0.62
26.48	31.08	4.6
42.88	60.36	17.48
56.80	76.10	19.30
57.92	82.00	24.08
68.08	81.44	13.36
77.78	89.52	11.78
82.60	82.46	- 0.14
84.42	90.00	5.58
85.80	93.44	7.64

The results clearly show the disparity in performance between UBL and this work. It is also witnessed that maximum rise in the learning curve happens when the data set has reached to set 4 in this iteration. The average set of improvements over time is noted to be 11.20 %.

5. Conclusions and Future Work

Chapter 5.1 presents the conclusions drawn from this work and chapter 5.2 briefs on potential future work.

5.1 Conclusions

As part of this work, a semantic parser is tested by employing a healthy marriage between unification part of work from **Zettlemoyer (2009)** and adaptive boosting. The dataset used to test this work is from a game known as Cricket. Training is performed initially on the cricket queries and then extensive testing is performed. It has been concluded with this work that as the number of iterations pertaining to a given training phenomenon increase, so does the performance. Also it can be observed that the performance of the system is directly proportional to the amount of training it received. Thus more the number of situations the agent is trained on, the better the performance. This work simulates a language acquisition mechanism, that is, the way a foreigner learns a new language. It can be also concluded that employing an ensemble approach proves favorable in a learning domain. Since outlier data is purged out by this technique, it proves helpful in solving classifier problems and categorization tasks.

5.2 Future Work

Language acquisition and induction via a semantic parser is studied with the help of this work and the performed work is tested using a novel cricket dataset. For complexity reasons, the query set that was built for this work is small. Future work may proceed at expanding this work and test it on a much larger scale and with multi domain data. Language models built for this work is entirely based upon the English language. Future work may extend it to some other languages where the game of cricket is played. Some of linguistic detail cannot be differentiated and generalized with that of regular English prose. For normality reasons this work bases on plain English text rather than acerbic and flowery use of language, where it fails miserably. This might be one other area of future interest. Future work should also use efficient Machine Learning algorithms that do not consider all of the available parses for a given sentence and arrive at the correct response in a reasonable amount of time. Certain other deficiencies may be tackled that can further enhance performance.

6. Bibliography

Rohit J. Kate. Learning for Semantic Parsing with Kernels under Various Forms of Supervision. Ph.D. Thesis, Department of Computer Sciences, University of Texas at Austin, August 2007.

Rohit J. Kate. Transforming Meaning Representation Grammars to Improve Semantic Parsing. In Proceedings of the Twelfth Conference on Computational Natural Language Learning (CoNLL 2008), pp. 33-40, Manchester, UK, August 2008.

Rohit J. Kate. Learning for Semantic Parsing with Kernels under Various Forms of Supervision. Ph.D. Thesis, Department of Computer Sciences, University of Texas at Austin, August 2007.

Knight, K. (1989). Unification: A multidisciplinary survey. *ACM Computing Surveys*, 21:93–124.

Kubat, R., DeCamp, P., Roy, B., and Roy, D. (2007). Totalrecall: Visualization and semi-automatic annotation of very large audio-visual corpora. In *Proceedings of International Conference on Multimodal Interfaces*.

Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. Inducing Probabilistic CCG Grammars from Logical Form with Higher-order Unification. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing* (EMNLP), 2010.

Tom Kwiatkowski, Sharon Goldwater, Luke Zettlemoyer, and Mark Steedman. A Probabilistic Model of Syntactic and Semantic Acquisition from Child-Directed Utterances and their Meanings. In *Proceedings of the Conference of the European Chapter of the Association for Computational Linguistics* (EACL), 2009.

Murzyn D. (2003). A* parsing: Fast exact viterbi parse selection. In *Proceedings of the Human Language Technology Conference of the NAACL*.

Cynthia Matuszek, Evan Herbst, Luke Zettlemoyer, and Dieter Fox. Learning to Parse Natural Language Commands to a Robot Control System. In *Proceedings of the International Symposium on Experimental Robotics* (ISER), 2006.

Yoav Artzi and Luke Zettlemoyer. Weakly Supervised Learning of Semantic Parsers for Mapping Instructions to Actions. *Transactions of the Association for Computational Linguistics Vol. 1*, (TACL), 2008

7. Appendix

This chapter presents the grammar denotations used as part of this work and the set of database queries used to test this work.

7.1 Grammar Denotations Used

The following representations are used for convenience. **'e' –** entity, **'s'** - select, **'f' –** from, **'w'** - where. The entire set of grammar demotions used is presented as below:

select f w

select f w:e

select f (select w) w

select w (select w) w:e

from i a

where i a

select f select f w (select w) f w

select w (select w) w) f w:e

select f select f (select w) w) f w f w

select f select f (select f select f (select w)) f w f w f w f w select f select f w (select w) w:e a ((where i a))

select f select f (select w) w (select f select f w (select w) f w) a (select w (select w) w) f w:e)

select f select f w (select w) a (from i a)

select f select f w (select w) a (where i a)

select f (select w) w) :e a (select f select f w (select w) a (from i a)) select f select f (select f select f (select w select f select f w (select w) a (where i a)

)) f w f w f w f w

select f select f w (select w) w:e a ((where i a)):e select f select f (select f select f (select

w)) f w f w f w f w

select f select f (select f select f (select w select f select f (select w) w) f w f w

)) f w f w f w f w:e

select f select f (select w select f select f (select w) w (select f select f w (select w) f w) a

(select w (select w) w) f w:e)) w) f w f w :e

select f select f (select f select f (select w)) f w:e f w:e f w f w:e select f select f (select w) w:e (select f select f w (select w) f w) a (select w (select w) w) f w:e)

select f select f (select w:e) w (select f select f w:e (select w) f w:e) a (select w (select w) w) f w:e)

select f (select w) w) :e a (select f select f w:e (select w:e) a (from i a))

select f (select w) w) :e a (select f select f select f select f (select f select f (select w)) f w:e f w:e f w f w:e w (select w) a (from i a))

select f (select w) w:e) :e a (select f select f w:e (select w) a (from i a))

7.2 Database Query set

The set of 262 data base queries (extracted from 26 query sets each containing 8 ~10 queries) that was used to test this work have been presented as under:

1. Who won the world cup of 2003?

2. Give me the test captain of Australia during the World cup of 1992?

3. Find the names of Australia players who can bowl□?

4. Who is the highest Run getter in the year 2001 played at the lords?

5. Give me the winner of ashes series of 2009?

6. How many runs did Ganguly score in the Sri Lanka cup held in 2010 at the Dhaka stadium?

7. How many matches have been played at the Eden gardens in the year 1995?

8. List the number of wickets taken by all members of the Indian team in 1997 at the Chinnappa stadium?

9. What is the Odi rank of England?

10. List all of the players who have hit at least one six at the Chinnappa stadium?

11. How many runs did Yuvraj score at the Firoz shah kotla in the year 2011?

12. Give me the total number of Odi matches played by Sachin?

13. What is the highest score in all of the Odi matches that India has played at the Eden gardens?

14. Who is the Odi coach of South Africa team?

15. How many times did Anil Kumble pick 4 wickets in the World cup of 2007?

16. Who is the Test captain of New Zealand?

17. Who won the match held between South Africa and Pakistan in the world cup at the wanderers?

18. Who is the ODI captain of South Africa?

19. Give me the player who picked most of the wickets at the match held between India and Pakistan in the World cup 1992?

20. Name the player with the highest average in ODI’s?

21. Who hit the most number of fours in the match held at The Wanderers between West Indies and South Africa?

22. Which test captain of India has the highest batting average?

23. Give me the top three players who scored the highest in all of the matches held between New Zealand and England?

24. Which team won most in all of the test and ODI matches held at the Eden Gardens?

25. Name the test captain of Pakistan?

26. Which player holds the best figures in the ICC trophy between teams Pakistan and West Indies?

27. What is the all time best of Sachin Tendulkar?

28. How many matches have been played between India and South Africa at The Wanderers?

29. What is the highest score by any team in the Ashes series of 1987?

30. Who won most of the matches between New Zealand and England?

31. What are the best bowling figures of Dylan Ashcraft when he played against West Indies in the match held between England and West Indies?

32. How many times did Anil Kumble pick all of the wickets in the World cup of 1996?

33. List the coach for Pakistan cricket team?

34. What is the average of the England’s captain?

35. Which South African captain holds the highest score till date in all of the world cup matches?

36. Which Sri Lankan player took the most number of wickets in all of the matches held between India and Srilanka?

37. List the number of wickets taken by all members of the Indian team in 1997 at the chinnappa stadium?

38. What is the odirank of England?

39. □Give the names of srilanka players who can bowl?

40. List all of the players who have hit at least one six at the chinnappa stadium?

41. How many runs did Yuvraj score at the Firoz Shah Kotla in the year 2011?

42. Give me the total number of ODI matches played by Sachin?

43. What is the highest score in all of the ODI matches that India has played at the Eden gardens?

44. How many times did Anil kumble pick four wickets in the world cup of 2007?

45. Who is the odi coach of South Africa team?

46. □Which teams are playing in border-gavaskar trophy?

47. List the names of the players who have taken 2 or 3 wickets at the wanderers in the year 1990?

48. Which teams are playing in champions trophy□?

49. List out teams participating in world cup□of 1986?

50. How many times did Pakistan win a match in the 2004 World cup against West Indies?

51. What is the highest score of Brian Lara in an ODI match?

52. Give me the number of times Harbhajan Singh took 3 or more wickets in any match against Australia?

53. List the coach for South Africa cricket team□?

54. Who took the highest number of wickets in the match between India and Pakistan at the Lords in 2008?

55. What are the names of westindies players who can just bowl□?

56. Find the names of Pakistani players who can bowl□?

57. Give the name of odi player who scored the maximum runs in the match between India and England at the champion’s trophy in 1992?

58. □

59.

60. □

61. List the teams playing the world cup□in the year 2007?

62. Who won the semi-final of 1992 World cup?

63. What is the highest score of Vinod?

64. Which team won the highest matches in the world cup of 2007?

65. How many wickets did Anil Kumble take in the champions trophy final of 1992 against Pakistan?

66. Who is the captain of the South African cricket team?

67. Give me the number of fours hit by Saurav Ganguly?

68. Who is the coach of the Indian cricket team?

69. How many runs did Glenn McGrath score in the Ashes series of 2009?

70. What is the highest score of Sean Pollock?

71. How many wickets did Shane Warne take in the champions trophy of 2006?

72. Who is the highest run getter in the Ashes series of 2009?

73. What is the test rank of Australia?

74. Who is the captain of the South African cricket team?

75. How many matches did Sachin play against England?

76. Give me the highest run getter in the match played between Pakistan and Australia in 2001 at the Wanderers?

77. How many matches did Anil Kumble play against Australia?

78. Who is the coach of Australian cricket team?

79. Give me the player who scored the second highest number of runs in the match between England and South Africa at the Lords in 1992?

80. Who picked the maximum wickets in the world cup of 2007?

81. What is the highest score of Shaun Pollock?

82. Who took the leading number of wickets in the champions series of 1008?

83. Who is the best bowler of 2004 Ashes series

84. What is the lowest score of Adam Gilchrist?

85. Who is the skipper of the Australian cricket team?

86. Name the best bowler of Ashes series of 2004?

87. Who scored the highest in the T20 series between India and Kenya?

88. Outline all the names of players of the English cricket team?

89. How many matches did India win in the world cup of 1992?

90. Who is the captain of the New Zealand cricket team?

91. How many wickets did Glenn McGrath take against England in the Ashes series of 2009?

92. What is the best score of the Australian cricket team's captain?

93. How many wickets did Jhonty Rhodes take in the match against Kenya at the Wandereers?

94. What the best performance of Glenn McGrath?

95. How many matches did Sean Pollock play at the Wanderers?

96. How many matches did South Africa win at the Wanderers?

97. Who is the captain of the Indian cricket team?

98. How many times did South Africa lose to India at the Wanderers?

99. How many times did harbhajan Singh take 5 or more wickets in a match against Pakistan?

100. How many runs did Ricky Ponting score in the match against India at the Lords?

101. What is the best score of the Indian captain at the world cup of 2004

against Australia?

102. How many wickets did Jonty Rhodes take in the match against Pakistan at the Lords held in 1998?

103. Give me the name of the Srilankan player who took the maximum number of wickets in the match between India and Srilanka at the chnnaswamy stadium?

104. Which captain of the Indian cricket team was the most successful in all o

the matches against England?

105. Who is the most successful bowler of the Indian Cricket team?

106. Who took the most number of wickets in the Champions trophy final held at 2004 between India and Australia?

107. Give me the captain of New Zealand?

108. What is the highest score of Inzamam in the match against South Africa at the Lords in 2000?

109. Who picked the most number of wickets in the final of the World cup between England and Australia?

110. Give me the highest run getter in the final of the ICC T20 on the year 2008?

111. How many wickets did Anil Kumble pick in the match against England at the Wankhade stadium in 1998?

112. Name the player who scored the maximum runs in the match held between South Africa and Kenya?

113. Give me the captain of the Australian cricket team?

114. Who is the lead run getter in the ICC Champions trophy of 2004?

115. Give me the captain of the Sri Lankan cricket team?

116. How many wickets did Sachin take in his career?

117. What is the best performance of Sachin in the match against Australia held at the Lords?

118. How many matches were played between India and England at the Lords?

119. How many matches did India win against England?

120. What is the number of times that India won against England at the Lords?

121. Who is the captain of the South African cricket team?

122. Give me the name of the captain of the Aussie cricket team?

123. Name the lead run getter of the Indian team?

124. How many times did Anil Kumble pick 5 wickets against the English cricket team?

125. Who is the lead run getter in the Champions trophy held at Lords in the year 2004 between India and England?

126. Name the captain of the Aussie cricket team?

127. Who is the best captain of the Aussie cricket team?

128. Name the highest run getter in the Champions trophy held in 2004?

129. Who won the world cup of 2008?

130. Give me the name of the Aussie cricket team?

131. Name the lead bowler of the Aussie cricket team?

132. Who is the best bowler of the Indian cricket team?

133. Who is the highest run getter in the match between Australia and New Zealand at the Lords?

134. Give me the skipper of the Aussie cricket team?

135. Who won the most number of matches between India and England?

136. Who won the most number of matches between teams South Africa and Australia?

137. How many matches did England win against New Zealand?

138. Who is the skipper of the Indian Cricket team?

139. Who won the most number of matches between India and New Zealand?

140. How many wickets did Anil Kumble claim in the match against Australia at Eden gardens?

141. Who is the most successful skipper of the Aussie cricket team?

142. How many matches did South Africa win against England in the year 2009?

143. Give me the captain of the South African cricket team?

144. In which year did Sachin score the highest number of runs against England?

145. Who is the most successful skipper of the New Zealand Cricket team?

146. Name the most successful skipper of the Aussie cricket team?

147. What is the highest score of Saun Pollock in the final match between India and South Africa?

148. Name the player with the highest number of runs in all of the matches played between England and Netherlands?

149. Who is the skipper of the Netherlands cricket team?

150. How many wickets did Sunil Gavaskar pick in all the matches held against Pakistan?

151. Who is the highest run getter in the match between Australia and England in 2007 at the wanderers?

152. Which team won the world cup of 1978?

153. Who is the skipper of the New Zealand cricket team?

154. How many matches did Sachin play against West Indies?

155. What is the score of Sachin the match held at the wanderers against West Indies?

156. How many matches did Ireland win against England?

157. How many wickets did Anil Kumble pick against Kenya in the match held at the centurion club in 1997?

158. What is the highest score of Russell martin against Srilanka?

159. Name the player with the highest number of runs in the world cup of 2004?

160. Give me the player with the maximum number of runs in the world cup of

1987?