Applying Machine Learning to Rank Domain-specific Logical Expressions of Natural Language Descriptions

A Thesis Submitted

in Partial Fulfillment of the Requirements

for the Degree of

Master of Technology

by

Sailesh Kumar Raju Ramayanam

Y8127403

under the guidance of

Dr. Amey Karkare

to the

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

May 2013

CERTIFICATE

It is certified that the work contained in this thesis entitled "**Applying Machine Learning to rank domain-specific logical expressions of Natural Language Descriptions**", by **Mr. Sailesh Kumar Raju Ramayanam (Roll No. Y8127403)**, has been carried out under my supervision and this work has not been submitted elsewhere for a degree.

Amey Karkare 8/5/2013

Dr. Amey Karkare
Assistant Professor,
Department of Computer Science & Engineering,
Indian Institute of Technology, Kanpur
Kanpur, 208016.

ABSTRACT

We address the problem of ranking a set of candidate translations of an English sentence. The ranking reflects the order of probability of correctness of the translations. Such kind of ranking is an important and essential phase in most of the natural language translation systems. Most of the attempts to solve this problem employ adhoc methods for scoring the translations. We propose a systematic two level approach to tackle this problem. For the first level, we identify certain characteristics of a candidate translation, which reflect its possibility of being correct. We present methods to quantify these characteristics. For the second level, we propose a way of combining the component scores from level 1. We present a novel loss function which captures the notion of our success metric (the number of test inputs for which the correct formula is given rank 1). We validate all our ideas using extensive experiments. The results show that most of our ideas indeed work as expected.

Acknowledgements

This work would not have been possible without the invaluable guidance and support of my guide, **Dr. Amey Karkare**. I am truly indebted and express my earnest thanks to him. I also express my sincere thanks to **Dr. Sumit Gulwani**, **Dr. Subhajit Roy** and **Dr. Mark Marron** for providing their valuable time whenever needed and even at very short notices. Many ideas in this work are due to the fruitful discussions we had. I also thank **Aditya Desai**, **Nidhi Jain** and **Vineet Hingorani**, who, besides being brilliant project partners, were very dear friends and made working on this thesis an enjoyable experience. I genuinely feel honored to have worked with so many great minds.

I must surely thank the department of Computer Science and Engineering, IIT Kanpur, for providing me with an excellent environment for research.

I would like to thank my parents, my sisters and all family members for their immense support and trust. Apart from these, there are many others and it is difficult to mention the names of everyone but I must mention Mr. Mohammed Rafi, Sr. Celine, Sr. Basil, Mr. Rama Lingeswara Rao, Mr. N. Chowdary Babu, Mr. Chandra Sekhar Naidu and the Rural Development Trust, without whose support I would not have made such a fruitful journey. I thank all my friends at IIT Kanpur for making my stay here one of the most memorable ones.

Last, but not the least, I thank God Almighty, for showering me with all the above blessings.

Dedicated to my

Father: *Seeta Rama Raju R*

Mother: *Seeta Rama Laskhmi R*

Sister & Brother-in-law: *Hymavathi, Veera*

Sister: *Madhuri*

family, teachers and friends

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Systems that can translate input in natural language to an intermediate logic have numerous applications ranging from providing comfortable querying systems, to intelligent tutoring systems, to program synthesis and many more. It seems natural if end-users, who are not very comfortable with computers, desire to have a natural language interface. Surprisingly, such a desire is equally strong among computer professionals as well. But, the kind of applications desired by the people in these two extremes can be quite unrelated. For example, an end user would wish to have a system that allows to query in English, whereas a data analyst would want a spreadsheet system that accepts commands in English. Thus, the applications are diverse and the target users are also equally diverse.

1.1 Applications of Natural Language Translation Systems

1.1.1 Querying Systems

Most of the querying systems today provide a graphical interface. Drop down menus, text fields, check boxes etc. are among the most common ways of choosing the query attributes. This kind of systems are very easy and comfortable to use when the queries are simple and the number of attributes is small.

Example: *In order to find flights available to go from Denver to Atlanta, one simply has to choose the source and destination cities.*

Websites like MakeMyTrip[12], Yatra[18], ClearTrip[4] have very good user interface for such queries. Now suppose, the user wants to know the flights from Denver to Atlanta which stop

at Pittsburgh. The user has to choose one more query attribute. Another user may want to know the latest such flight, which provides dinner and of a particular airlines. To allow such complicated queries, the interface should have the options to choose many attributes. This is a challenge on the part of the designer, because as the number of possible attributes increases, the interface becomes denser and the user experience might deteriorate. On the other hand, if the attributes are limited, then the queries that can be made also become limited. So a tradeoff has to be made. Note that here the system is capable of executing complex queries and the user might also be wanting to make complex queries. The only hurdle that prevents complex queries is the limitation imposed by the graphical interface.

A natural language interface, on the other hand, has no such limitations because the interface involves giving a natural language description which can be given as plain text.

1.1.2 Intelligent Tutoring Systems

With the advent of websites like KhanAcademy[1], learning has taken a new turn. Opportunity for learning has surpassed many hurdles like location, age etc. and the virtual classroom is available to thousands of users, at the comfort of learning from home. The scale and the variety of courses is so huge that it is impractical to monitor each student manually. Moreover, the question base keeps on expanding and it would be immensely helpful if there is an automated system which can guide the students in understanding and solving the questions. The first step to any such system is to understand the question at hand, which will be in natural language. Therefore, an efficient system for natural language translation would help to build intelligent tutoring systems.

1.1.3 Program Synthesis

Text editing tasks like insertion, deletion, replacement etc. are extremely common and millions of users need them everyday. Sometimes users may need to perform a complicated editing task on a spreadsheet or a text file. For instance, consider the following tasks.

1. Insert a ":" immediately after a sequence of numbers.
2. Insert "*" at the beginning of each line containing "P.O. BOX".
3. Insert "| " after the 1st occurrence of " ".

Each of these tasks is complicated and cannot be performed by normal find and replace action. Systems like Microsoft Excel provide a way to program such complex tasks (Microsoft Excel does this in the form of *macros*). However, there are many users who are not proficient in using such methods and moreover such complex tasks may not arise frequently enough to necessitate the learning of text editing programming. If we can have a system which can take a natural language description of a task and generate a text editing program that can perform the task, text editing can be extremely simplified. Thus, natural language translation can be used for program synthesis as well.

There are numerous other applications for natural language translation. However, natural language translation, in its broadest sense, is a very hard task. Nevertheless, attempts have been made to build efficient systems for translation.

1.2 Our Work

Our work is based on *bag of words* model, which is a very common approach in document classification. In this, a sentence is considered as a collection of words and no importance is given to their arrangement. We then generate all possible expressions using these words and pick up the correct expression using certain methods. The challenge in this approach is that the method to pick up the correct expression must be quick and highly efficient, because the number of possible expressions using a given set of words can be quite high. We use machine learning techniques to achieve this goal.

This thesis is organised as follows. Chapter 2 provides an overview of research done in natural language translation and how our work differs from the existing work. Chapter 3 discusses the methods to generate expressions using bag of words model and also formally defines our problem statement. Chapter 4 discusses our approach to solve the problem of ranking a set of candidate translations. Chapter 5 presents the results of our experiments and an analysis of the results. Chapter 6 presents the conclusions and scope for future work.

Chapter 2

Existing Work and Contributions of Our Work

2.1 Existing Work

Natural Language Translation is not at all a new area ([17]) and a lot of effort has gone into building efficient systems to solve the goal. Initial systems allowed input with restricted vocabulary and structure ([16], [2]), though the restrictions were not as strict as those in a programming language. However, these were very sensitive to issues like grammatical mistakes and were not very robust. Later, several systems were developed which were tolerant to such issues. Still, the amount of success in natural language translation is not very impressive and there is a lot of scope and need for new ideas.

Many attempts at tackling natural language translation ([10] [11]) use bag of words model in one form or other. Words from the input description are mapped to certain keywords, which are then combined in a certain way to generate possible translations. The main challenge with bag of words model is that it results in a large number of possible translations, even after eliminating syntactically and semantically ill formed ones. The correct translation must be picked efficiently or else the user experience deteriorates. [10] presents a keyword based programming utility for Java. It uses a score, based on the number of words used from the input sentence. Unused words are penalized by some fixed amount chosen by the authors. [11] presents another keyword based web utility which uses a score, based on the mappings of input words to keywords. Each kind of

mapping has certain predetermined weight which is used to compute the score of a translation. Both these approaches consider a translation as a function tree and attempt to build it top down. In both these utilities, the values used for scoring are based on some heuristics.

PRECISE [15] is a system which translates natural language descriptions of database queries into SQL queries. This also uses a bag of words model, but presents a novel way to produce candidate translations. It models the mappings of input words to keywords as a graph and uses max-flow algorithms to produce valid translations. It, however, does not rank/classify the translations. In case of multiple translations, all of them are presented to the user, who then has to choose a correct one.

All the above approaches either use adhoc (heuristic based) scoring schemes or do not score the translations at all. Thus, there is a need to identify the characteristics of a translation that can reflect the possibility of its correctness. Once it is done, methods to quantify such characteristics should be developed and then the resulting scores can be used to rank the translations. Such an approach will have a well formed theory and can provide more insights into natural language translation. One of the contributions of this work is to define such characteristics and propose methods to compute them.

Once we have certain quantified properties of a translation, the next question is which one to use. If all (or some) of them are to be used, then how to combine them. In any case, the ultimate goal is to rank a given set of translations. Ranking is a very common function performed by many systems. Despite this, the problem of ranking has not received much attention in machine learning community. The problem with ranking is that it almost always is produced by sorting w.r.t. some score and the sort function is inherently non-differentiable w.r.t. the score. Most machine learning algorithms, on the other hand, require a smooth objective function for the purpose of training. Nevertheless, some research has been done on ranking which use a smooth function that can approximate the behavior of ranking function. RankNet[3] and RankingSVM[8] are among the most popular ones. Each of these presents a smooth objective function which is defined in terms of relevance order between the pairs of objects to be ranked. Principle motivation behind both these and most other research in ranking, is the problem of ordering search engine results or documents. In the case of search results, more often than not, there is a relevance order between most of the pairs of retrieved

results. But this is not the case in our context. Although the objective of this work is to *rank* translations, in reality, the relevance order between most of the pairs of candidate translations is undefined. A user will not prefer one incorrect translation over the other. The only relevance is between a correct and an incorrect translation. So, by using the above mentioned ranking approaches and for that matter any approach based on pairwise relevance, we would be trying to do a lot of irrelevant learning. More importantly, we might not be doing the required learning at all, because the number of pairs for which the relevance order is defined is way too small.

RankBoost[6] uses an entirely different approach and addresses a problem highly relevant to our setting. The problem it addresses is combining a set of weak rankings to produce a single strong ranking. It is relevant to our setting because it turns out that the characteristics we identified for the candidate translations are weak ranking functions. RankBoost uses a variation of boosting technique, to learn the strong ranking function. However, it also uses the same pairwise relevance orders for learning the strong ranking. In addition to this, the score computed by the strong ranking function depends only on the relative values of the weak scores and not on absolute values. This dependence on relative values may work well under the assumption that the weak rankings are in fact rankings and not quantified scores. In our case, the weak rankings are indeed quantified scores.

2.2 Our Contributions

Our work is different from the existing work in the following ways.

1. We propose certain concrete characteristics of a translation and also methods to quantify these characteristics.

2. Each of the above characteristics reflects the possibility of correctness of the translation, although in a weak way.

3. We propose a novel approach to rank a set of candidate translations by combining their weak scores.

4. In a general sense, our approach can be used to learn ranking functions for objects where the pairwise relevance orders do not exist between most of the pairs.

Chapter 3

Framework and Problem Statement

In this chapter we shall define the basic framework on which our system is built, discuss its working and at the end, formally state the problem we attempt to solve in the remaining chapters.

3.1 Input

The aim of our system is to translate a given English sentence into an intermediate language. The intermediate language is defined using a context free grammar (CFG) and hence is far easier for a computer to understand. We call the input sentence *a benchmark*.

Example: *I would like to travel from Washington to Pittsburgh on August twentieth*

For any intelligent system to work, relevant knowledge has to be represented in some form. For our system, this translates to knowing how to construct sentences in the intermediate language and also how to deduce the semantics of English w.r.t. the intermediate language. The relevant knowledge for this is provided through the following.

1. Grammar
2. Dictionary

Grammar, G

Grammar is a set of rules to be followed in order to construct a meaningful sentence in the intermediate language. Although it is a context free grammar (CFG), its format is tailored to

suit our application. The productions in the grammar can be of 3 types.

1. Passive Production
2. Active Production *or* Named Production
3. Nameless Production

Definition 1. *Passive Production: A production of type $N \rightarrow T$, where N is a non-terminal in G and T is a terminal in G. Terminals like T are called Passive Terminals.*

FoodType $\rightarrow$ *BREAKFAST* is a passive production. The terminal *BREAKFAST* is a complete attribute in itself. It does not act on any other arguments and hence, it is a *Passive Terminal* in our grammar.

Definition 2. *Active/Named Production: A production of type $N \rightarrow T(N_1, N_2, .., N_k)$, where N, N_i are non-terminals in G and T is a terminal in G. Terminals like T are called Active Terminals.*

Active productions are required when there is a grouping of information possible and this grouping is explicitly indicated by a word in the benchmark. For example, in flight related queries, departure information needs to be grouped together and quite often it is indicated by words like *from, depart, leave* etc. Therefore we have the following named production in the grammar for *ATIS* domain (domains will be discussed shortly).

Departs $\rightarrow$ *EQ_DEPARTS(City, Time, Weekday, Daynum, Month)*

Definition 3. *Nameless Production: A production of type $N \rightarrow (N_1, N_2, .., N_k)$, where N, N_i are non-terminals in G.*

Nameless productions are required when there is a grouping of information possible and this grouping is **not** explicitly indicated by any word in the benchmark.

AtomicRowPred $\rightarrow$ *(Departs, Arrives)* is a nameless production which says that departure and arrival information, together form a predicate. One may say that having a production which allows conjunct (AND operation) can eliminate the need for this nameless production. But, conjunct does not prevent generation of a predicate having only departure information. A translation with only departure information is an ill formed one. But nameless productions

capture the semantic grouping which is specific to the domain at hand and prevent the above kind of ill formed translations.

Dictionary, D

Dictionary is a mapping from English words to terminals in the intermediate language. It can be thought of as a set of ordered pairs (t, W), where t is a terminal in G and W is a set of English words. This means that if a sentence S contains a word $w \in W$, then one possible intention of w in S could be the role associated with t in G.

The grammar as well as the dictionary depend on the domain at hand and are to be provided by domain experts. Except for these two, rest of our system is automated and does not require any external intervention. It however uses the knowledge provided through grammar and dictionary.

In our experiments we considered the following 3 domains.

1. *ATIS* - Queries regarding flight details
2. *Automata* - Undergraduate level problems on finite state automata
3. *Text Editing* - Tasks like insertion, deletion, replacement etc. in text files

3.2 Translation Process

Once a grammar G and a dictionary D are given, the system can be set up for translation. Broadly there are 3 steps in the process of translation.

1. *Generation of initial expressions*
2. *Combination of expressions*
3. *Extracting a correct translation*

3.2.1 Step 1: Generation of initial expressions

We start with an empty list and pick a word w in the sentence S. We find out all the mappings of w in D and add the corresponding terminals to our list. We then repeat this for every other

word in S. Finally, we end up with a list of terminals. But, terminals as such cannot be used for translation because they can have arguments (e.g. Active Terminals) which have to be instantiated. Therefore at the end, we generate one expression for each of the terminals in our list and these expressions are our initial expressions (for active terminals, the arguments are set to null). Here we are considering the sentence as a bag of words and consequently generating a bag of expressions.

```
Data: Sentence S, Dictionary D
Result: List of expressions, L
M ← EmptyList;
L ← EmptyList;
for each word w in S do
    for each mapping w → t in D do
        Add (t, w) to L;
    end
end
for each pair (t, w) in M do
    e ← createExpression(t, w);
    L.Add(e);
end
return L;
```

Algorithm 1: Generation of initial expressions

Example: *I would like to travel from Washington to Pittsburgh on August twentieth*

According to our dictionary, the words *I, would, like, travel, on* map to nothing. So, these words do not generate any terminals. Remaining words generate the following.

Word	→	***Terminal***	→	***Expression***
to	→	*EQ_ARRIVES*	→	*EQ_ARRIVES(null, null, null, null, null)*
from	→	*EQ_DEPARTS*	→	*EQ_DEPARTS(null, null, null, null, null)*
Washington	→	*CITY*	→	*CITY*
to	→	*EQ_ARRIVES*	→	*EQ_ARRIVES(null, null, null, null, null)*
Pittsburgh	→	*CITY*	→	*CITY*
August	→	*MONTH*	→	*MONTH*
twentieth	→	*DAYNUM*	→	*DAYNUM*

The initial expressions may not be reachable from the start symbol of G, and they may not

even be complete.

Definition 4. *Hole: A non-terminal position in an expression, which is not yet instantiated. The non-terminal is said to be the type of the hole.*

Definition 5. *a) Partial Expression: An expression with at least one hole or one partial sub-expression.*

b) Complete Expression: An expression with no holes and no partial sub-expressions.

- In the above example, none of the expressions is reachable from the start symbol of G.
- The expression $EQ\_DEPARTS(null, null, null, null, null)$ is a partial expression because the non-terminals $City, Time, Weekday, Daynum, Month$ are not yet filled. These are the holes. The first hole is of type $City$, the second is of type $Time$ and so on.
- The expression $CITY$ is a complete expression.

Each initial expression is annotated with the word that generated it and this annotation is retained throughout. Therefore, one instance of $CITY$ is clearly distinguishable from another instance. Moreover, the words are annotated with their position in the sentence and hence two instances of the same word are also distinguishable.

3.2.2 Step 2: Combination of expressions

We just saw that the first step generates a bag of expressions. In the second step, the initial expressions are combined with each other to generate new expressions (partial/complete). The updated expression set is combined repeatedly until a fixed point is reached. At this point, all the complete expressions which are reachable from the start symbol of G are reported as candidate translations for S.

The combinations, however, are not arbitrary. They are dependent on the grammar G. There are two types of combinations.

Definition 6. *Type 1 combination: Given a partial expression e_1 and another expression e_2, e_2 can combine with e_1 if e_1 has a hole which can be filled with e_2 without violating the type safety. In other words, e_2 must be reachable from the non-terminal present at the concerned hole.*

Example: *CITY(Pittsburgh)* can combine with *EQ_DEPARTS(null, null, null, null, null)* because as per the grammar for ATIS domain, the non-terminal *City* can generate the terminal *CITY.* Consequently, *CITY(Pittsburgh)* can also combine with *EQ_ARRIVES(null, null, null, null, null).*

Definition 7. *Type 2 combination: Given two expressions e_1, e_2, they both can combine, if there is a Nameless Production whose right hand side contains two non-terminals, one of which can generate e_1 and the other e_2.*

Example: *Please tell me the flights between Boston and Philadelphia next Thursday.*

In this benchmark, *Boston* and *Thursday* are departure details, but there is no word which maps to *EQ_DEPARTS*. Still, *Boston* and *Thursday* have to be grouped. To handle this situation, the grammar for *ATIS* domain contains the following *Nameless production.*

Eq_Departs_IMP $\rightarrow$ *(City, Time, Weekday, Daynum, Month)*

This production allows *Boston* and *Thursday* to combine via *type 2 combination.*

Step 2 is formally expressed in the following algorithm. The function $combine(e_1, e_2)$ is assumed to return a set of expressions which is the result of combining e_1 and e_2 via combinations of both types.

This algorithm is only an overview of the combination process. The actual procedure for combination is much more complex and can be found in [7]. However, for understanding this document, this algorithm is sufficient.

Data: Set of initial expressions, E
Result: Set of complete expressions, C
$E' \leftarrow \Phi$;
while $E' \neq E$ **do**
    $E' = E$;
    for *each pair of expressions* e_1, e_2 *in* E' **do**
        if e_1, e_2 *can be combined* **then**
            $E \leftarrow E \cup combine(e_1, e_2)$;
        end
    end
end
$C \leftarrow \Phi$;
for *each expression* $e \in E$ **do**
    if e *is complete* **then**
        $C \leftarrow C \cup \{e\}$;
    end
end
return C;

Algorithm 2: Combination of Expressions

3.2.3 Step 3: Extracting a correct Translation

At the end of *step 2*, we will have a set of candidate translations for the given benchmark S. The next task is to pick up a correct translation from this set of translations. This thesis focuses on this aspect of translation, i.e, *step 3*. We propose an efficient way of identifying a correct translation. It is important to note that the method to pick up a correct translation has to be very efficient, because the number of candidate translations can be very large given that we are using *bag of words* model for translation. The next section formally states the problem.

3.3 Problem Statement: Ranking a set of translations

Given an English sentence S, grammar G, dictionary D and a bag B of candidate translations for S w.r.t. (G, D), rank the translations in B w.r.t. their probabilities of being correct.

To summarise, we consider a sentence as a bag of words, generate all possible translations and from the resultant translations, pick up a correct translation. In the further chapters, we shall describe the approach taken to identify a correct translation and also the results of our experiments on different domains.

Chapter 4

Machine Learning for Ranking

We take a machine learning approach to rank a given set of candidate translations. For this, we collect a large number of benchmarks, produce their candidate translations (using the methods in [7]) and identify the correct translation for each benchmark. This constitutes our training data, using which we build models which can rank a set of candidate translations.

4.1 Two levels of Machine Learning

In our approach, we apply machine learning at two levels. In the first level, a set of component scores is computed for each candidate translation. In the second level, the component scores of each candidate translation are combined to produce a single final score. This final score will be a measure of our confidence in the corresponding translation. So, after the second level, sorting the candidate translations w.r.t. their final scores produces a ranked list, which is the objective of this work.

4.2 Significance of two level approach

The importance of two level approach is the following. In the first level, we identify some simple characteristics of a candidate translation and come up with a method to quantify each of the characteristics. These characteristics must be easy to define and compute and must be useful[1]

[1]Usefulness of a component score for ranking means, a correct translation must highly likely have a high value for this score. In other words, there must be a positive correlation between the score and the correctness of a candidate translation. A negatively correlated component can also be used with appropriate changes in the second level.

for ranking the candidate translations. However, individually they may not be strong enough to be used as final score for ranking. In short, the first level generates a set of *weak scores*. We call these as *component scores*.

In the second level, we come up with a way of combining the weak scores which can result in a strong final score. The two levels can be operated independently. New components can be added to the first level and different methods of combining can be applied at the second level, without affecting one another. The following two sections provide a detailed discussion of these two levels.

4.3 Level 1: Computing Component Scores

We propose to use the following 3 component scores.

1. Fraction of Used Words
2. Word to Terminal Probability
3. Connections Probability

4.3.1 Component 1: Fraction of Used words

For a benchmark S and a candidate translation F, we can define the following.

1. Non-usable words
2. Usable words
3. Used words

Definition 8. *Non-usable words(S): All the words in S that do not map to any terminal in the grammar G.*

For a word w, if *map(w)* represents the set of all terminals that w maps to in the dictionary D, then

$$NonUsableWords(S) = \{w \mid w \in S,\ map(w) = \Phi\}$$

Example: *I would like to travel from Washington to Pittsburgh on August twentieth*

In this example, according to our dictionary, the words *I, would, like, travel* and *on* do not map to any terminal in G. Consequently, they can never be used in the translation of this benchmark. Hence the set of non-usable words for this benchmarks is,

$$\{I,\ would,\ like,\ travel,\ on\}$$

Definition 9. *Usable words(S): All the words in S that map to at least one terminal in the grammar G.*

In contrast to *non-usable words*, *usable words* are those words in S which map to a non-empty set of terminals.

$$UsableWords(S) = \{w \mid w \in S,\ map(w) \neq \Phi\}$$

Alternately, any word w in S which is not a non-usable word is a usable word. Thus the set of *usable words* and the set of *non-usable words* are mutually exclusive and exhaustive.

In the above example, the set[2] of usable words is,

$$\{to,\ from,\ Washington,\ to,\ Pittsburgh,\ August,\ twentieth\}$$

Definition 10. *UsedWords(F, S): The set of words in S that have actually been used to generate F.*

UsedWords(F, S) is always a subset of *UsableWords(S)*. A correct translation F_c for our running example uses the words *from, Washington, to, Pittsburgh, August* and *twentieth*. So, the set of used words for F_c is,

$$\{from,\ Washington,\ to,\ Pittsburgh,\ August,\ twentieth\}$$

Definition 11. *Fraction of Used Words (F, S): The ratio of the number of used words to the number of usable words is defined as the fraction of used words.*

$$Score1(F) = f_{used}(F, S) = \frac{\text{number of words used in F}}{\text{number of usable words in S}}$$

For the correct translation F_c of our running example,

$$f_{used}(F_c) = \frac{6}{7}$$

[2]One may feel that technically the sets of usable words and used words are multi-sets, because a word can appear more than once in the same sentence. But, as mentioned at the end of section 3.2.1, every word is uniquely identifiable using its position.

Why Fraction of Used words ?

Clearly, the fraction of used words is very simple and easy to compute. But, how can it help in ranking?

The set, *UsableWords(S)*, can be thought of as the total useful information (*useful* for translation) available in S. Similarly, the set, *UsedWords(F, S)*, can be thought of as the total information which has actually been used in the process of translation. Consequently, the quantity *fraction of used words of F* denotes the fraction of available information that has been used for translation. A correct translation should utilise maximum amount of information presented in S, or rather, a translation which uses very less information from S is more likely to be incomplete (it can still be correct). One can note that the previous statement makes an implicit assumption that there is very little redundancy in S.

Consider the following translations for our running example.

F1 *AtomicRowPredSet(AtomicRowPred(EQ_DEPARTS(CITY(Washington), ANY(), ANY(), DAYNUM (twentieth), MONTH(August)), EQ_ARRIVES(CITY(Pittsburgh), ANY(), ANY(), ANY(), ANY())))*

F2 *AtomicRowPredSet(AtomicRowPred(EQ_DEPARTS(CITY(Washington), ANY(), ANY(), ANY(), ANY()), EQ_ARRIVES(CITY(Pittsburgh), ANY(), ANY(), ANY(), ANY())))*

The corresponding sets of used words are,

U1 {*from, Washington, to, Pittsburgh, August, twentieth*}

$f_{used}(F1) = \frac{6}{7}$

U2 {*from, Washington, to, Pittsburgh*}

$f_{used}(F2) = \frac{4}{7}$

One can verify that *F1* is the correct translation. Although *F2* is also correct, it is not complete. For almost every benchmark in our training data, the correct translation is found to have a higher value for the quantity f_{used}, which supports our assumption of very little redundancy in S. So, a higher value for f_{used} is an indicator of completeness and hence can be used to identify correct translations. However, f_{used} alone cannot be used to rank the candidate translations. The following section discusses the reasons for this.

Weakness of Fraction of Used Words

It is easy to see that f_{used} can have a maximum value of 1. However, it need not always be 1. Our running example is one such case, although in this case the correct translation has the highest value of f_{used} among all the candidate translations. Use of two *EQ_ARRIVE*s (generated by two instances of the word *to*) is not allowed by the grammar itself. Hence there is no translation with $f_{used} = 1$. However, *F1* is not the only translation with the highest f_{used}. The following translation also has the same $f_{used}(= \frac{6}{7})$.

F3 *AtomicRowPredSet(AtomicRowPred(EQ_DEPARTS(CITY(Pittsburgh), ANY(), ANY(), DAYNUM (twentieth), MONTH(August)), EQ_ARRIVES(CITY(Washington), ANY(), ANY(), ANY(), ANY())))*

Unlike *F2* which is correct (although not complete), *F3* is not correct. It considers departure city as arrival city and vice-a-versa. However, there is no way to distinguish between *F1* and *F3* using only f_{used}. Such ties among candidate translations (w.r.t. f_{used}) are present in most of the benchmarks which is one reason why f_{used} alone cannot be used as final score for ranking.

Moreover, there are benchmarks in which an incorrect translation has a higher value for f_{used} than the correct translation. Consider the following example.

Example: *I need information on flights leaving Dallas arriving in Boston leaving Dallas early in the morning.*

In this benchmark, the word *Dallas* has been used twice and the second use is actually redundant. So, the correct translation for this benchmark will have $f_{used} < 1$. However, the grammar allows for a translation which uses the second *Dallas* also, in addition to all other words used by the correct translation. Consequently, the correct translation will not occur in the first position if the ranking is done based on f_{used} alone.

Redundancy in the benchmark causes the correct translation to have a low value for f_{used}. Since we are dealing with natural language, some amount of redundancy is expected and this is another reason for f_{used} being a weak score.

4.3.2 Component 2: Word to Terminal Probability

For any candidate translation F, there is an implicit set of terminals, S_T, associated with it. This is the set of all terminals and only those that occur in F. Each of these terminals is due to a word from the corresponding sentence S. Consequently, there is an implicit set of words, S_W, associated with F. The set S_W is nothing but the set of used words. Every word w in S_W can be associated with exactly one terminal in S_T (the one that is due to w) and vice-a-versa. This defines a mapping from the set of used words, to the set of terminals in F. One should notice that it is this mapping that is generated first and the translation, F, is a result of the combination of the terminals in S_T. In fact, combination of the terminals in S_T results in a list of candidate translations and F is only one of them.

Now consider the set of terminals S_{T_c} associated with the correct translation T_c. For the translation T to be the correct translation, S_T must be same as S_{T_c}. So, for a candidate translation to be correct, it is absolutely necessary (although not sufficient) that the associated word to terminal mapping be correct.

Example: *I am looking for a flight leaving Denver traveling to Atlanta with a stop at Pittsburgh.*

Consider the following mappings for the above example.

Word	**Mapping 1**	**Mapping 2**
leaving	EQ_DEPARTS	EQ_DEPARTS
Denver	CITY	CITY
to	EQ_ARRIVES	EQ_ARRIVES
Atlanta	CITY	CITY
stop	EQ_STOPS	COL_STOPS
Pittsburgh	CITY	CITY

The only difference between the two sets of mappings is the mapping of the word *stop*. The intent of this word in the sentence is that there must be a stop at Pittsburgh. The terminal *EQ_STOPS* in our grammar is a function that takes a *CITY* as an argument and returns whether a particular flight stops at the particular *CITY* or not. On the other hand, the terminal *COL_STOPS* is a flag which projects all the stops of a particular flight. Clearly, *mapping 1* is the correct mapping. Moreover, *mapping 2* can never produce a correct translation, or rather,

any translation whose set of terminals does not include the terminals under *mapping 1* cannot be a correct translation.

From the above discussion we can conclude that the correctness of word to terminal mapping is a necessary condition for the correctness of a candidate translation. However, in reality, we do not know the list of terminals associated with the correct translation. So, instead of labeling a set of mappings as correct or incorrect, we propose to predict the probability of it being correct. This can be further simplified by looking at individual mappings instead of the set in an entirety. We extend our notion of *correct set of mappings* to *correct mapping*, by saying that each word in the input sentence must map to the correct terminal. Strictly speaking, this is not necessary, because exchanging the mappings[3] of two words can still result in the correct translation. However, in this case, the intent of the words will have been wrongly identified and more importantly, translations with such mappings may have lower value for component 3 (to be discussed later in section 4.3.3).

Once we compute the probabilities of individual mappings, the probability score of the mappings associated with a candidate translation, F, is taken to be the product of probabilities of individual mappings in F.

$$\textit{Score2(F)} = \prod_{M \in F} \textit{probability}(M)$$

where F is a translation and M is a word to terminal mapping.

In the example discussed above, we have seen that a correct translation should use 6 words from the input sentence. Consequently, there will be 6 terms in the product of its *Score2*. Now consider a translation which uses only 4 words and hence has only 4 terms in the product of its *Score2*. Directly comparing the *Score2* of these two translations may not be the right thing to do, because they involve different number of terms. However, all through our discussion we have been missing a latent mapping present in many translations. If a translation F does not use a usable word w, then it means that F considers w to be of not being informative and so is ignoring it. Alternately, we can say that F maps w to a stop word. So, in reality, the overall word to terminal mapping should consist of the mapping of unused usable words to stopwords, in addition to the mappings of used words. With this modified definition of word to terminal

[3]The exchange of mappings is possible only if both the words map to both the terminals in the given dictionary D

mappings, any candidate translation (for a given sentence) will contain same number of terms in the product of *score2*.

Computation of Score 2

Given a grammar G and dictionary D, let T be the set of all the terminals in G. In order to consider words mapping to stop words, we shall use a dummy terminal t_{stop}. For the grammar G and dictionary D, our task is the following.

Given a sentence S, a word w in S and a terminal t in $T \cup \{t_{stop}\}$, predict the probability that w maps to t in the correct translation of S, w.r.t. G and D.

In our approach, we consider the word w and its part of speech, $POS(w)$ as the features and the corresponding terminal t as a class label. In this set up, our task can be framed as,

Given a feature vector $\vec{f}$ and a class label c, predict the probability of $\vec{f}$ belonging to c.

Algorithms for training and testing are presented below. Part of speech of a word is extracted using Stanford NLP parser[13].

Data: Sentence S, Translation F
Result: Training data extracted from S and F
$D \leftarrow \Phi$;
for *each usable word* $w \in S$ **do**
  Feature Vector $\vec{f} \leftarrow \langle w, POS(w) \rangle$;
  Class $c \leftarrow t_{stop}$;
  if *w has been used in F* **then**
    Let w map to a terminal t in F;
    $c \leftarrow t$;
  end
  $D \leftarrow D \cup \{(\vec{f}, c)\}$;
end
return D;

Algorithm 3: getTrainingData(Sentence S, Translation F)

Data: TrainingSet B
Result: A classifier to predict the probability of a word to terminal mapping
$D \leftarrow \Phi$;
for *each training instance* $b \in B$ **do**
    $D \leftarrow D \cup$ getTrainingData($b.Sentence,\ b.correctTranslation$);
end
C = Classifier.Train(D);
return C;

Algorithm 4: Train a classifier for Component 2

Data: Sentence S, Translation F, Classifier C
Result: Score2 of F
score $\leftarrow$ 1;
for *each usable word* $w \in S$ **do**
    Feature Vector $\vec{f} \leftarrow \langle w, POS(w) \rangle$;
    Class $c \leftarrow t_{stop}$;
    if *w has been used in F* **then**
        Let w map to a terminal t in F;
        $c \leftarrow t$;
    end
    score $\leftarrow$ score * C.predict($(\vec{f}, c)$);
end
return score;

Algorithm 5: Compute the Score2 of a translation of the given sentence

Weakness of word to terminal probability score

We mentioned earlier that for a candidate translation to be correct, the correctness of word to terminal mapping is necessary but not sufficient. To see why it is not sufficient, let us reconsider the example discussed earlier.

Example: *I am looking for a flight leaving Denver traveling to Atlanta with a stop at Pittsburgh.*

We already discussed that for a candidate translation to be correct, it should have the following mapping.

Word	Correct Mapping
leaving	EQ_DEPARTS
Denver	CITY
to	EQ_ARRIVES
Atlanta	CITY
stop	EQ_STOPS
Pittsburgh	CITY

Now consider the following two translations for the above sentence.

F1 *AtomicRowPredSet(AtomicRowPred(EQ_DEPARTS(CITY(Denver), ANY(), ANY(), ANY(), ANY()), EQ_ARRIVES(CITY(Atlanta), ANY(), ANY(), ANY(), ANY())), EQ_STOPS(CITY(Pittsburgh)))*

F2 *AtomicRowPredSet(AtomicRowPred(EQ_DEPARTS(CITY(Atlanta), ANY(), ANY(), ANY(), ANY()), EQ_ARRIVES(CITY(Denver), ANY(), ANY(), ANY(), ANY())), EQ_STOPS(CITY(Pittsburgh)))*

F2 is same as F1 except that the arrival and departing cities have been interchanged. Clearly, F1 is a correct translation and F2 is an incorrect translation. However, both have the same word to terminal mapping. Thus, a correct translation not only has the correct word to terminal mapping, but also the correct placement of terminals. Our *score2* captures only the correctness of mappings and is blind to the placement of terminals. Therefore, *score2* alone cannot be used to distinguish among the translations having the same word to terminal mappings, which makes it a weak score.

4.3.3 Component 3: Connections Probability

We have just seen that in addition to having a correct word to terminal mapping, the placement of terminals is also important to decide whether a candidate translation is correct or not. The third component captures this notion of placement of terminals.

Definition 12. *Connection: For any production p in the grammar G, where $p : N \rightarrow a_1 a_2..a_i..a_j..a_n$, the tuple (p, i, j) is called a connection. For a named production (section 3.1), a_1 will be the terminal T.*

As discussed in section 3.2.2, translation starts and proceeds by combining different expressions. Any combination of two expressions is an instantiation of a connection in the grammar.

Definition 13. *Combination: Given two expressions e_1, e_2 and a connection $C : (p, i, j)$, if e_1 and e_2 are used to fill the i^{th} and j^{th} holes of p respectively, then e_1 and e_2 are said to combine via C and the combination is expressed as $C(e_1, e_2)$.*

The definitions of connection and combination formally express the notion of the placement of terminals (note that terminals generate initial expressions, see section 3.2.2). Using the definition of connection, we can say that the translations F1 and F2 in the previous section are equal in every respect, except for connections among the following words (strictly speaking, the connections are among the expressions generated by these words).

leaving, Denver, to, Atlanta

In F1, *leaving* is connected with *Denver* and *to* is connected with *Atlanta* whereas in F2, *leaving* is connected with *Atlanta* and *to* is connected with *Denver*. Clearly, in F2, the connections are wrong. So, for a translation to be correct, in addition to having the correct word to terminal mappings, the connections among the terminals must also be correct.

Like in the case of component 2, instead of labelling the connections as correct/incorrect, we shall compute the probability of a connection being correct and take the product of individual connections as score3.

$$Score3(F) = \prod_{C \in F} probability(C)$$

where F is a translation and C is a connection. However, different translations can have different number of connections and consequently, different number of terms in the above product. In order to have a common ground for comparison, we take the geometric mean of the above product as our score3. So, the modified definition for score3 is,

$$Score3(F) = GeometricMean(\prod_{C \in F} probability(C))$$

Feature Selection for Component 3

The discussion that follows is in the context of an input sentence S, grammar G, dictionary D, a candidate translation T and a correct translation T_c. Our task is the following.

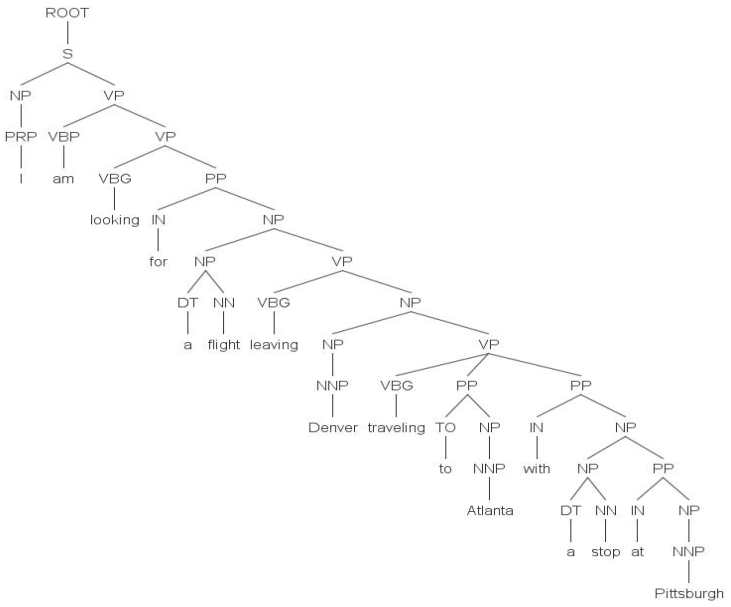


Figure 4.1: A parse tree from Stanford Parser

Given two expressions e_1, e_2 and a connection C, predict the probability that T_c contains the combination $C(e_1, e_2)$.

Now the question is, what features of e_1 and e_2 can help us learn and predict their connection probability?

A translation can also be viewed as an expression tree where the leaf nodes are initial expressions generated from the words in input sentence and any internal node is also an expression which is the combination of the expressions at its child nodes. In a natural language, it is the parse structure of a sentence that helps to interpret its meaning. For a translation, it is the expression tree that helps to interpret the translation. So, expression tree of a translation is the counterpart of the parse tree of the input sentence S. Therefore, for constructing a correct expression tree, we can use the properties of the parse tree of S and hence we define most of the properties for component 3 in terms of the parse tree of S. We use Stanford NLP parser [13] to obtain the parse tree. An example parse tree is shown in figure 4.1.

First we shall define certain properties [5] of an expression e and then proceed to define our features. In the following definitions, e represents an expression (rather a sub-expression) in the translation T. Any reference to subtree means a subtree in the parse tree of S.

Definition 14. *Words(e) = set of all the words used in e*

This is similar to the set S_W defined in section 4.3.2. S_W is defined for a translation, whereas $Words(e)$ is defined for any expression e (a translation is also an expression).

Definition 15. *a) minSubTree(e) = smallest subtree in parsetree that contains all the words from Words(e)*

b) SubTree(e)

$$= \begin{cases} minSubTree(e), \textit{if minSubTree(e) contains any word} \in \textit{Words(T) - Words(e)} \\ \textit{largest subtree containing minSubTree(e) and no word} \in \textit{Words(T) - Words(e)} \end{cases}$$

Definition 16. *Window(e) = [start, end], where*

start = min {pos | pos = position of w in S, w ∈ Words(e)}

end = max {pos | pos = position of w in S, w ∈ Words(e)}

Definition 17. *a) LCA(R1, R2) = Least Common Ancestor of r1 and r2*

b) Order(R1, R2) $= \begin{cases} 1 & \textit{if R1 lies to the left of R2} \\ -1 & \textit{otherwise} \end{cases}$

where R1, R2 are two subtrees and r1, r2 are the respective root nodes

Definition 18. $Overlap(e_1, e_2) = \begin{cases} 1 & \textit{if w1.end} < \textit{w2.start} \\ -1 & \textit{if w2.end} < \textit{w1.start} \\ 0 & \textit{otherwise} \end{cases}$

where, w1=Window(e_1) and w2=Window(e_2)

Definition 19. $Distance(e_1, e_2) = \begin{cases} w2.start - w1.end & \textit{if Overlap}(e_1, e_2)\textit{==1} \\ w1.start - w2.end & \textit{if Overlap}(e_1, e_2)\textit{==-1} \\ \perp & \textit{otherwise} \end{cases}$

where, w1=Window(e_1) and w2=Window(e_2)

The above definitions capture the properties required to define our features. Given two sub-expressions e_1, e_2, let

- R1 = SubTree(e_1)
- R2 = SubTree(e_2)
- r1 = root(R1)

- r2 = root(R2)

We define the following 7 features, of which first 5 are parse tree based and the last 2 are sentence based.

f1 Part of Speech of (r1)

f2 Part of Speech of (r2)

f3 distance(LCA(R1, R2), r1)

f4 distance(LCA(R1, R2), r2)

f5 Order(R1, R2)

f6 Overlap(e_1, e_2)

f7 Distance(e_1, e_2)

The idea behind features $f1$ and $f2$ is to learn which kind of phrases can combine using a particular connection. For example, in the ATIS domain, when phrases related to departure and arrival combine to give a predicate, sometimes departure phrase happens to be a prepositional phrase (PP) while arrival phrase happens to be a verb phrase (VP). Although this does not hold always, the idea is that such properties can be learnt using $f1$ and $f2$.

Features $f3, f4$ and $f5$ are motivated from the fact that we want to maintain the structure of translation as much similar to the parse tree as possible. The distance and order quantities can help us do that. However, one must remember that the structure of the translation depends on the structure of the grammar G and hence one should try to make it as close to the natural language as possible.

Features $f6$ and $f7$ are based on the belief that closely related information often occurs together in the input sentence.

Computation of Score 3

Given a grammar G, the number of possible connections is fixed. We learn one classifier for each connection. The task of each of the classifiers is to predict the probability that a correct

translation contains the combination of the given two expressions. The algorithms are presented below.

Data: Sentence S, CandidateTranslations T, CorrectTranslation t_c, TrainingData[ ] D
Result: Training data extracted from S, T and t_c
$C_{corr} \leftarrow$ All combinations in t_c;
for *each translation* $t \in T$ **do**
  for *each combination* $comb \in t$ **do**
    Feature Vector $\vec{f} \leftarrow$ features of $comb$ w.r.t. S;
    $occurs \leftarrow False$;
    if $comb \in C_{corr}$ **then**
      $occurs \leftarrow True$;
    end
    D[comb.connection] $\leftarrow$ D[comb.connection] $\cup \{\langle \vec{f}, occurs \rangle\}$;
  end
end

Algorithm 6: getTrainingData(Sentence S, CandidateTranslations T, CorrectTranslation t_c, TrainingData[ ] D)

Data: TrainingSet B, Grammar G
Result: A list of classifiers which can be used to compute Score3
$D[G.numConnections]$;
$C[G.numConnections]$;
for *each connection* $c \in G$ **do**
  $D[c] \leftarrow \Phi$;
  $C[c] \leftarrow null$;
end
for *each training instance* $b \in B$ **do**
  getTrainingData($b.Sentence$, $b.candidateTranslations$, $b.correctTranslation$, D);
end
for *each connection* $c \in G$ **do**
  $C[c] \leftarrow Classifier.Train(D[c])$;
end
return C;

Algorithm 7: Train classifiers for Component 3

Data: Sentence S, Translation F, Classifier[ ] C
Result: Score3 of F
score $\leftarrow$ 1;
for *each combination comb* $\in F$ **do**
    Feature Vector $\vec{f} \leftarrow$ features of *comb* w.r.t. S;
    score $\leftarrow$ score * C[comb.connection].predict($\vec{f}$);
end
return GeometricMean(score);

Algorithm 8: Compute the score 3 of a translation of the given sentence

Weakness of connections probability score

The weakness of Score3 stems from the fact that it is blind to word to terminal mappings. A translation which uses less number of words, but uses them correctly will have a high value for Score3, because everything that is used has been connected correctly.

Example: *I am looking for a flight leaving Denver traveling to Atlanta with a stop at Pittsburgh.*

Consider the following two translations for the above sentence.

F1 *AtomicRowPredSet(AtomicRowPred(EQ_DEPARTS(CITY(Denver), ANY(), ANY(), ANY(), ANY()), EQ_ARRIVES(CITY(Atlanta), ANY(), ANY(), ANY(), ANY())), EQ_STOPS(CITY(Pittsburgh)))*

F2 *AtomicRowPredSet(AtomicRowPred(EQ_DEPARTS(CITY(Denver), ANY(), ANY(), ANY(), ANY()), EQ_ARRIVES(CITY(Atlanta), ANY(), ANY(), ANY(), ANY())))*

F1 is the correct translation. F2 is also correct, but not complete. However, F2 gets a higher value for Score3 than F1. Using only Score3, in this case, will rank F2 as better than F1 whereas the combined score ranks F1 at rank 1.

4.4 Level 2: Combining Component scores

Once we compute the component scores, the next task is to combine them into a single *final score*. A simple way to combine the component scores is to take their sum or product. In either case, all the component scores get equal weight. But, in reality, all the component scores may not be equally important. So, a more general combination would be to take a weighted sum.

$$\textit{Final score, } s = \sum_{i=1}^{k} w_i.c_i$$

where c_i is the i^{th} component score, w_i is the corresponding weight and k is the number of components ($k = 3$ in our case). The weights have to be chosen systematically and preferably in an automated way.

4.4.1 Setup for Combining Component Scores

Our aim is to learn a weight vector for combining component scores. The weight vector is not any arbitrary vector but should combine the component scores in such a way that, ideally for any benchmark, the correct translation should get the highest final score. So, we want to find a weight vector which optimizes some metric. The metric we shall be using is the number of test benchmarks for which the correct translation is assigned rank 1. Optimization is a very common machine learning approach and several optimization techniques exist. Almost every technique requires a loss function (or alternately a gain function) defined whose minimum (alternately maximum) point corresponds to the optimal point of the underlying metric. Here we want to learn a weight vector and hence the domain for the loss function is a $k-$dimensional vector space, where k is the number of component scores ($k = 3$ in our case).

Before we can define a loss function, we should understand what does *loss* mean in our setting. The task at hand is to rank a set of candidate translations and ideally we wish the correct translation to be ranked at the top of the ranked list (i.e. to get rank 1). So, we fail (or lose) if the translation at rank 1 is incorrect. Given a weight vector $\vec{w}$, let us define the following quantities for any benchmark S.

$$\begin{aligned} s_{corr} &= \textit{final score of the correct translation for } S \\ s_{wrong} &= max\{\textit{final scores of the remaining translations for } S\} \end{aligned}$$

Let F_c and F_w be the corresponding translations. For F_c to be ranked first, we need

$$s_{corr} > s_{wrong} \Rightarrow \frac{s_{corr}}{s_{wrong}} > 1$$

Thus, our failure depends on the quantity $\frac{s_{corr}}{s_{wrong}}$ and hence we shall define our loss function in terms of $\frac{s_{corr}}{s_{wrong}}$. s_{corr} and s_{wrong} themselves are defined in terms of a weight vector $\vec{w}$. So, effectively f is a function of the weight vector $\vec{w}$.

One can notice that the above loss function is concerned with one benchmark only. But for training, we want a loss function defined for the entire training data. This part is easy. We define a loss function for each benchmark and the overall loss function for training will be the sum of the individual loss functions. Therefore, it suffices to characterise the individual loss function. So, the next question is what characteristics do we want the loss function to exhibit?

As discussed earlier, we fail if $\frac{s_{corr}}{s_{wrong}} \leq 1$, i.e., we suffer loss if $\frac{s_{corr}}{s_{wrong}} \leq 1$ and no loss otherwise.

$$\frac{s_{corr}}{s_{wrong}} > 1 \Rightarrow \textit{Loss must be small}$$
$$\frac{s_{corr}}{s_{wrong}} \leq 1 \Rightarrow \textit{Loss must be large}$$

Alternately,

$$\frac{s_{wrong}}{s_{corr}} < 1 \Rightarrow \textit{Loss must be small}$$
$$\frac{s_{wrong}}{s_{corr}} \geq 1 \Rightarrow \textit{Loss must be large}$$

At this point, one can think of many loss functions with the above characteristics. For e.g.,

1. $\frac{s_{wrong}}{s_{corr}}$
2. $-\frac{s_{corr}}{s_{wrong}}$
3. $s_{wrong} - s_{corr}$
4. $(\frac{s_{wrong}}{s_{corr}})^p, p \in \{1, 2, 3, ...\}$
5. $e^{\frac{s_{wrong}}{s_{corr}}}$ and so on

However, there is something more to the nature of the loss function in our setting. Remember that our overall loss function is the sum of individual loss functions. Suppose that we have a benchmark S which has $\frac{s_{corr}}{s_{wrong}} > 1$ and every other benchmark has $\frac{s_{corr}}{s_{wrong}} \leq 1$. Consider the following two ways (among others) to decrease the loss.

1. Increase the ratio $\frac{s_{corr}}{s_{wrong}}$ of S much beyond 1, with little focus on the remaining benchmarks

2. Increase the ratio $\frac{s_{corr}}{s_{wrong}}$ of other benchmarks to move just beyond 1

Although our aim is to minimise loss, the value of loss is not really our success metric. Our success metric is the number of benchmarks for which we can rank the correct translation at rank 1. So, we really never want to follow the first way to minimise loss.

For any benchmark, all that we want is $\frac{s_{corr}}{s_{wrong}} > 1$. We are not interested in exactly by how much s_{corr} is greater than s_{wrong}, i.e., it does not matter if s_{corr} is 2 times or 20 times or only 1.1 times larger than s_{wrong}. So we want the loss to saturate in the region $\frac{s_{corr}}{s_{wrong}} > 1$. This is important because, otherwise optimization process might proceed to minimise loss by increasing the ratio $\frac{s_{corr}}{s_{wrong}}$ much beyond the value of 1 rather than improving those benchmarks which have $\frac{s_{corr}}{s_{wrong}} < 1$. Therefore, another desirable characteristic of the loss function is,

$$\frac{s_{wrong}}{s_{corr}} << 1 \Rightarrow \textit{Gradient should quickly fall to } 0$$

In addition to this, we also want the loss function to saturate for very small values of $\frac{s_{corr}}{s_{wrong}}$. To see why, suppose that one benchmark, S, has $\frac{s_{corr}}{s_{wrong}} << 1$ and every other benchmark has $\frac{s_{corr}}{s_{wrong}} > 1$. If the loss function does not saturate for small values of $\frac{s_{corr}}{s_{wrong}}$ and if gradient due to S is very large (for e.g. $e^{\frac{s_{wrong}}{s_{corr}}}$), then it can disturb the status of other benchmarks because the resultant change in the weight vector $\vec{w}$ can be very large. Moreover, a very small value for $\frac{s_{corr}}{s_{wrong}}$ might mean that the benchmark S is inherently difficult to rank. For example, if there is an incorrect translation F_i, which has a higher value for every component score than that of the correct translation, F_c, then there is no way for F_c to get a higher final score than that of F_i [4]. This situation is possible if there a lot of redundancy in the benchmark text. In any case, we want the loss to saturate for very small values of $\frac{s_{corr}}{s_{wrong}}$.

$$\frac{s_{wrong}}{s_{corr}} >> 1 \Rightarrow \textit{Gradient should quickly fall to } 0$$

Following is a complete characterisation of the loss function.

[4] In principle, negative weights can result in F_c being ranked higher than F_i, but in our experiments the weight vector has always ended up being positive. Also, using negative weights for any of the 3 components contradicts the definition of the component itself.

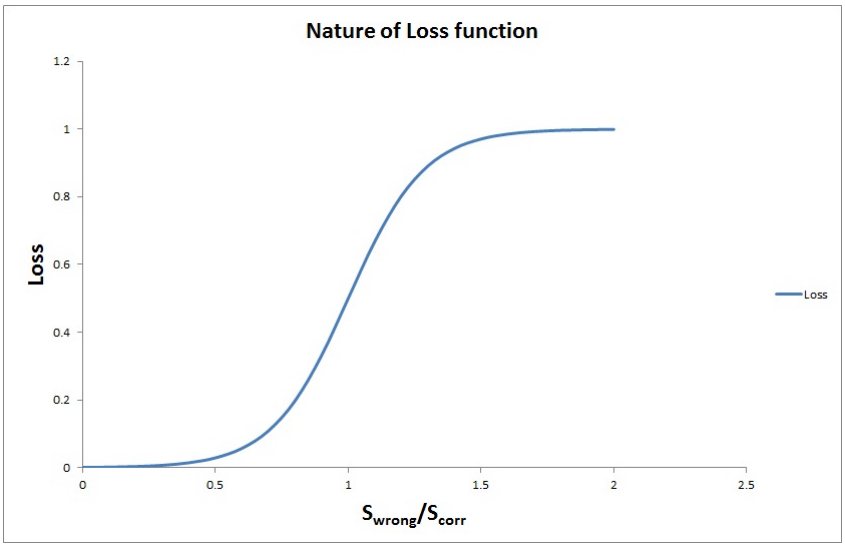


Figure 4.2: Nature of loss function

$$\frac{s_{wrong}}{s_{corr}} < 1 \quad \Rightarrow \quad \textit{Loss must be small}$$
$$\frac{s_{wrong}}{s_{corr}} \geq 1 \quad \Rightarrow \quad \textit{Loss must be large}$$
$$\frac{s_{wrong}}{s_{corr}} << 1 \quad \Rightarrow \quad \textit{Gradient should quickly fall to } 0$$
$$\frac{s_{wrong}}{s_{corr}} >> 1 \quad \Rightarrow \quad \textit{Gradient should quickly fall to } 0$$

A function with these characteristics looks like the plot in figure 4.2. This plot is similar to a shifted *S-curve*. So, we can use a shifted form of logistic function as our loss function. In short, for a benchmark B, we shall use the following as our loss function.

$$f_B(\vec{w}) = \frac{1}{1+e^{-c(x-1)}}, \text{ where}$$
$$x = \frac{s_{wrong}(B)}{s_{corr}(B)} \text{ and } c \text{ is some positive constant}$$

Consequently, the overall loss function will be,

$$f(\vec{w}) = \sum_{\forall benchmark\ B} f_B(\vec{w})$$

A careful observation will reveal that the loss function defined this way is not really differentiable. The reason is that *max* function (which defines s_{wrong}) itself is not smooth in our setting. The incorrect translation with maximum score, i.e. F_w, may not remain constant over the entire domain and whenever F_w changes, the corresponding change in f_B is abrupt, rendering the loss function non-differentiable at the point of change. However, f_B remains differentiable as long as F_w does not change. In other words, our loss function is continuous and piecewise differentiable.

But almost every optimization technique requires a smooth loss function. Nevertheless, we tried using our loss function by using an adhoc modification. At the points of non-differentiability, we define the gradient to be one of the gradients in its neighborhood (the choice is made arbitrarily). Surprisingly, the optimization worked remarkably well and that too across domains. We shall discuss more about this in the results section.

We propose to use gradient descent technique for choosing the weights.

4.4.2 Gradient Descent

Gradient Descent is a very simple and well known machine learning technique that is used to optimize a given function. It is an iterative technique. Given a smooth multivariate function f and a starting point $\vec{x_0}$, the update rule is the following.

$$\vec{x_{n+1}} = \vec{x_n} - \gamma \vec{\bigtriangledown} f(\vec{x_n}) \quad n = 0, 1, 2, ..$$

where $\vec{\bigtriangledown}$ denotes the gradient and γ is a positive constant. At each step, $\vec{x}$ moves in the opposite direction of the gradient, i.e. in the direction in which the value of f decreases. The process is stopped when the change in the function value in successive steps is not significant, i.e.,

$$|f(\vec{x_{n+1}}) - f(\vec{x_n})| \leq \epsilon$$

where ϵ is a small positive constant. The function f is referred to as *loss function*. Effectively, gradient descent attempts to reach a minimum value of the loss function.

4.5 Summary

In this chapter, we proposed to follow a two level approach for ranking. We defined 3 component scores which are computed at level 1. In level 2, we discussed how the problem of combining the component scores can be cast as an optimization problem. We analysed the nature of loss function which suits our needs and finally decided to use a shifted form of logistic function as the loss function. Once training is completed at both the levels, the system will be ready for translating a given sentence. The next chapter discusses the experiments we performed and their results.

Chapter 5

Experiments and Results

5.1 Experimental Setup

5.1.1 Machine Learning

In the implementation of the first level of our approach, we used NaiveBayes classifier for learning the components 2 and 3. We used MATLAB implementation of the NaiveBayes classifier for our experiments.

5.1.2 Benchmarks

We chose the following 3 domains for evaluating our system.

1. *ATIS* - Queries regarding flight details
2. *Automata* - Undergraduate level problems on finite state automata
3. *Text Editing* - Tasks like insertion, deletion, replacement etc. in text files

A large number of benchmarks have been collected for each domain and candidate translations for each benchmarks have been generated. Benchmarks for all the domains were collected through various means which are described in [7].

5.1.3 Ranking Scheme

Recall that the input to our system is a set of candidate translations for a benchmark and the output is a ranked list of these translations. The ranking is done based on the *final score*s

(defined in section 4.4) of the candidate translations. Several ranking schemes exist and we use two different ranking schemes.

1. *1334 ranking*

2. *1224 ranking*

1334 ranking

Suppose there are 4 candidate translations T_1, T_2, T_3 and T_4, whose final scores are $2, 9, 15$ and 9 respectively. As per *1334 ranking*, T_3 gets rank 1, T_2 and T_4 each gets rank 3 and T_1 gets rank 4. In general, under this ranking scheme, if a translation T, with final score f, gets rank r, then there are exactly r translations, including T, with final score greater than or equal to f.

1224 ranking

Under this ranking scheme, T_3 gets rank 1, T_2 and T_4 each gets rank 2 and T_1 gets rank 4. In general, under this ranking scheme, if a translation T, with final score f, gets rank r, then there are exactly $r - 1$ translations with final score strictly greater than f.

Each ranking scheme helps to evaluate certain specific aspects of our system. To see what these aspects are, imagine our system as providing the user with a series of translations, one at a time. Our system first suggests its most confident translation and waits for the userś response. If it is rejected, our system then suggests the next most confident translation and the process continues until either the user accepts a translation or the candidate translations get exhausted. Therefore, the success of our system depends on the number of rejections faced before suggesting a correct translation and consequently on the rank of the correct translation.

If every candidate translation has a unique final score, then the ranking is unambiguous. But, in reality, this may not always be the case. Among the translations with same final score, the order of their suggestion is not clear. However, the maximum and minimum number of rejections is well defined. The maximum number of rejections corresponds to the worst case whereas the minimum number corresponds to the best case. It is easy to notice that *1334 ranking* helps to compute the worst case number of rejections and *1224 ranking* helps to compute the best case number of rejections.

Domain	Total benchmarks	Top	% of top	Top 3	% of top 3
ATIS	535	473	88.41	499	93.27
Automata	245	208	84.89	223	91.02
Text editing	492	405	82.31	467	94.91

Table 5.1: Overall Performance

We mostly talk about the worst case scenario in our experiments and hence unless otherwise stated, the ranking scheme remains *1334*, by default.

5.1.4 Cross Validation

We do a 10 fold cross validation using a random permutation of the benchmarks. The training and test sets are disjoint in all the experiments.

Following sections describe the overall performance of our system, experimental demonstration of our theory (described in chapter 4) and also the effectiveness and generality of our approach. Our major metric for evaluation is the number of benchmarks for which the correct translation gets placed at the top of the ranked list (i.e., $rank = 1$). In addition to this, we also present the number of benchmarks for which the correct translation gets placed in top 3 (i.e., $rank \leq 3$).

5.2 Overall Result

Statistics of overall performance of our system have been presented in table 5.1. The results show that the performance of our system is more or less uniform across domains and the uniformity is more pronounced in top 3 ranks, with maximum difference being less than 4%. State-of-the-art results for ATIS domain have been reported as 85%[19], 84%[14] and 83%[9]. However, these results are over a set of more than 5000 benchmarks as opposed to 535 in our case. But, we do believe that our approach will work well on the larger set as well, the only hurdle is to go through a large number of benchmarks, expand the grammar and dictionary files and prepare the training data. Since the datasets for Automata and Text Editing domains have been collected by us, there is no state-of-the-art work to compare the results in these domains.

Domain	Total benchmarks	1334 ranking Top	1334 ranking % of top	1224 ranking Top	1224 ranking % of top
ATIS	535	30	5.60	487	91.02
Automata	245	44	17.95	214	87.34
Text editing	492	40	8.13	425	86.38

Using only Component 1 (Fraction of Used Words)

Domain	Total benchmarks	1334 ranking Top	1334 ranking % of top	1224 ranking Top	1224 ranking % of top
ATIS	535	10	1.86	287	53.64
Automata	245	76	31.02	130	53.06
Text editing	492	44	8.94	303	61.58

Using only Component 2 (Word to Terminal probability)

Domain	Total benchmarks	1334 ranking Top	1334 ranking % of top	1224 ranking Top	1224 ranking % of top
ATIS	535	108	20.18	109	20.37
Automata	245	126	51.42	131	53.46
Text editing	492	169	34.34	172	34.95

Using only Component 3 (Connections probability)

Table 5.2: Weakness V/s Goodness of Component Scores: Top ranks

However, the authors of [10] report a success rate of 73% for keyword programming in web domain and the authors of [11] report a success rate of 59% for keyword programming in Java.

5.3 Strength of the component scores

In section 4.3, we defined various component scores and also gave arguments as to why each of these is a good score but a weak one. We conducted experiments to verify our arguments. In these experiments, we evaluated our metrics by using only individual component scores. We present the results using both the ranking schemes. *1334 ranking* helps to project the weakness of a component, whereas *1224 ranking* helps to project the goodness of a component. Weakness refers to several translations having same value for a component score, while goodness refers to the correct translation having a high value for a component score. The results are presented in tables 5.2 and 5.3.

First let us look at the results of *1334 ranking*. For the number of top ranked benchmarks, the best performance using only component 1 is 17.95%, that of component 2 is 31.02% and

Domain	Total benchmarks	1334 ranking Top 3	1334 ranking % of top 3	1224 ranking Top 3	1224 ranking % of top 3
ATIS	535	153	28.59	496	92.71
Automata	245	129	52.65	218	88.97
Text editing	492	220	44.71	454	92.27

Using only Component 1 (Fraction of Used Words)

Domain	Total benchmarks	1334 ranking Top 3	1334 ranking % of top 3	1224 ranking Top 3	1224 ranking % of top 3
ATIS	535	149	27.85	321	60.00
Automata	245	129	52.65	164	66.93
Text editing	492	214	43.49	353	71.74

Using only Component 2 (Word to Terminal probability)

Domain	Total benchmarks	1334 ranking Top 3	1334 ranking % of top 3	1224 ranking Top 3	1224 ranking % of top 3
ATIS	535	267	49.90	267	49.90
Automata	245	205	83.67	208	84.89
Text editing	492	301	61.17	311	63.21

Using only Component 3 (Connections probability)

Table 5.3: Weakness V/s Goodness of Component Scores: Top 3 ranks

that of component 3 is 51.42% all for Automata domain, whereas their combination resulted in a performance of 84.89% for the same domain (the best being 88.41% for ATIS). Similar results hold for top 3 benchmarks also. These show that our component scores are indeed very weak and hence cannot be used as final scores.

1224 ranking, on the other hand, projects a different nature of the component scores. For e.g., for component 1, the performance is as good as the one we got by combining the scores (table 5.1) and in fact slightly better. This shows that the correct translation indeed has a high value for this component, indicating that this is a good score for ranking. However, this alone is not powerful enough to distinguish the correct translation from the incorrect ones. Similar result holds for component 2 also. The performance of component 3, on the other hand, is practically same in both the ranking schemes. However, from *1334 ranking*, it is clear that component 3 is the strongest of all the components (though far weaker than the combined score).

We have just seen that our component scores are too weak to be used as final scores, but when combined, produce a very strong final score. The next question is, how significant is the contribution of each component score to the final score? Is it worth, the effort of computing a particular component score? In other words, can we drop a component score and still get comparable results? To this end, we evaluated our metrics after dropping one component at a time and the results are presented in table 5.4.

Dropping component 3 resulted in a huge decrease in the performance, as high as 81.86% in the worst case. Even in the best case, the decrease is still very large (47.75%). Dropping component 1 also resulted in heavy degradation of the performance, although not as high as that in dropping component 3. These show that components 1 and 3 (*fraction of used words* and *connections probability* respectively) are highly significant contributors to the final score.

Component 2, on the other hand, seems not so heavily contributing to the final score as the maximum drop in performance is only 4.67%. However, one must note that component 2 when combined with component 1 (or 3) resulted in much better performance than their individual performances. For example using only component 3 resulted in 108, 126 and 169 top ranked benchmarks in each of the domains (table 5.2), whereas, using component 3 along with component 2 resulted in 309, 153 and 308 top ranked benchmarks respectively (5.4). Thus,

Domain	Total	All comp.	Drop Component 1 Top	Drop Component 1 % change	Drop Component 2 Top	Drop Component 2 % change	Drop Component 3 Top	Drop Component 3 % change
ATIS	535	473	309	-30.65	448	-4.67	35	-81.86
Automata	245	208	153	-22.44	203	-2.04	91	-47.75
Text Editing	492	405	308	-19.71	382	-4.67	81	-65.85

Top ranks

Domain	Total	All comp.	Drop Component 1 Top	Drop Component 1 % change	Drop Component 2 Top	Drop Component 2 % change	Drop Component 3 Top	Drop Component 3 % change
ATIS	535	499	420	-14.76	492	-1.3	223	-51.58
Automata	245	223	202	-8.57	224	0.4	147	-31.02
Text Editing	492	467	396	-14.43	464	-0.6	287	-36.58

Top 3 ranks

Table 5.4: Significance of Component Scores

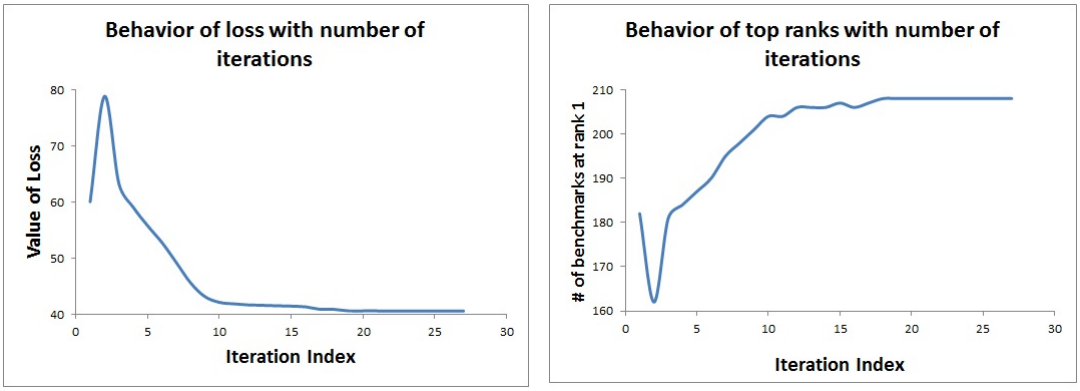
any combination of the components results in much better performance than their individual performances.

On the whole, these experiments show that our two level approach is very effective and works as intended.

5.4 Behavior of loss function

In order to optimise our success metric, we analysed the nature of loss function required in our setting, considered different loss functions and finally chose a shifted variant of logistic function as our loss function. Section 4.4.1 describes this analysis in detail. In this section we demonstrate the practical behavior of our loss function.

Figure 5.1 shows how the value of loss changes with iteration index and the corresponding number of top ranked benchmarks. It can be seen that whenever loss decreases, the number of top ranked benchmarks increases and vice-a-versa. Clearly, the number of top ranked benchmarks and the loss are negatively correlated which is what is required in an ideal setting for optimization.



(a) Loss V/s Iterations (b) # of top ranked benchmarks V/s Iterations

Figure 5.1: Loss Function in action

Domain	Total benchmarks	Equal weightage	Using gradient descent	% change
ATIS	535	392	473	15.14
Automata	245	182	208	10.61
Text Editing	492	364	405	8.33

Top ranks

Domain	Total benchmarks	Equal weightage	Using gradient descent	% change
ATIS	535	486	499	2.42
Automata	245	224	223	-0.40
Text Editing	492	448	467	3.86

Top 3 ranks

Table 5.5: Effect of gradient descent

5.4.1 Effect of Gradient Descent

We used gradient descent to learn a weight vector which is then used to combine the component scores. In this section we present the results of experiments conducted to see whether the use of gradient descent resulted in significant enhancement of results over using equal weightage to all the components. Table 5.5 shows the results.

From the results it is clear that using gradient descent has improved the number of top ranked benchmarks significantly (as large as 15%). However the number of top 3 ranked benchmarks did not improve much. Clearly, learning the weight vector has improved the quality of ranking because several benchmarks which were previously (i.e. using equal weightage) ranked at 2 or

3 (or perhaps even lower) have now (i.e. after gradient descent) moved to rank 1.

As mentioned at the end of 4.4.1, our loss function is non-differentiable. Still, practically it worked very well. From figure 5.1, it can also be noticed that as the gradient descent proceeds, the value of loss decreases and the number of top ranked benchmarks increases, i.e. showing that gradient descent is leading the function in appropriate direction.

5.5 Generality of our learning

By the term *generality of learning* we mean how well do the models learnt for one domain apply for other domains. This is important because if the models are applicable across domains, then the time and effort required for the learning in new domains can be avoided. Moreover, the models learnt for the existing domains can serve as starting points and also as baselines for comparing the performance in new domains.

We have seen that there are two levels of learning (section 4.1) in our approach. Level 1 involves computing component scores whereas level 2 involves combining them. Let us consider these levels one after the other.

Generality of Level 1

Level 1 computes 3 component scores. Component 1 is fraction of used words and its computation does not involve any learning. Computing components 2 (*word to terminal probability*) and 3 (*connections probability*) involves learning. However, this learning is heavily domain specific because the features themselves depend on the grammar G and dictionary D. Consequently, the question of applicability of learning across domains does not make sense as different domains have completely different and unrelated grammars and dictionaries.

Generality of Level 2

Level 2 involves learning a weight vector for combining the component scores and is domain independent. So, here the question of generality is applicable and translates to *can the weight vector learnt for one domain be used for other domains while getting comparable performances in all the domains*. We performed experiments to answer this question and the results are presented in table 5.6.

			Using weights learnt for					
			ATIS		Automata		Text Editing	
Domain	**Total**	**Best result**	**Top**	**% change**	**Top**	**% change**	**Top**	**% change**
ATIS	535	473	473	0.00	465	-1.49	446	-5.04
Automata	245	208	203	-2.04	208	0.00	205	-1.22
Text Editing	492	405	402	-0.60	401	-0.81	405	0.00

Top Ranks

			Using weights learnt for					
			ATIS		Automata		Text Editing	
Domain	**Total**	**Best result**	**Top**	**% change**	**Top**	**% change**	**Top**	**% change**
ATIS	535	499	499	0.00	498	-0.18	497	-0.37
Automata	245	223	222	-0.40	223	0.00	224	0.40
Text Editing	492	467	465	-0.40	468	0.20	467	0.00

Top 3 Ranks

Table 5.6: Generality of learning

The results show that the weight vector learnt for one domain performs remarkably well with other domains as well. The average decrease in the number of top ranked benchmarks is only 1.87% (maximum decrease being 5.04%). As for the number of top 3 ranked benchmarks, the change is practically insignificant (maximum decrease is less than 0.5%). From this, we can say that the learning performed at level 2 for one domain can be successfully employed for new domains.

Chapter 6

Conclusion and Future Work

We addressed the problem of efficiently choosing the correct translation of an English sentence from among a set of candidate translations by proposing a two level approach. We defined 3 component scores of a translation, which help to quantify the possibility of its correctness. We proposed a novel loss function for learning the combination function for the component scores. We experimented on 3 different domains and the results showed that our approach works really well across the domains. We also tried to validate our features and the results showed that 2 of the 3 component scores are extremely crucial for extracting a correct translation. We showed through experiments that the loss function we defined indeed captures the trend in ranking. Experiments also showed that the gradient descent technique we used, helped in enhancing the results very significantly and more importantly, the learning from one domain can be carried over to other domains.

Experiments show that component 2 (word to terminal probability) is not a very significant contributor to the ranking. We believe that the component in itself is a crucial one, but the way we are learning it is not effective. More specifically, the features are not powerful enough to learn the actual essence of component 2. Investigation into what features would help better learning of component 2 would be an interesting avenue for further work.

Also, we have seen that the loss function we defined is non-differentiable, but still works remarkably well with gradient descent. Future work can focus on designing differentiable functions which approximate our loss function. This may cause further improvement in performance because of the better interplay of the underlying theories.

Another interesting avenue for further work is to validate the features of component 3. We proposed a total of 7 features to learn component 3. Validating experiments can be done on the lines of our experiments on the strength of the component scores. By dropping one feature at a time and also using only one feature at a time, one can experimentally find the contribution of each of the features. This can provide more and new insights into the characteristics of the candidate translations in particular and natural language translation in general.

Bibliography

[1] Khan Academy. `https://www.khanacademy.org/`.

[2] Bruce W. Ballard and Alan W. Biermann. Programming in natural language: Nlc as a prototype. In *Proceedings of the 1979 annual conference*, ACM '79, pages 228–237, New York, NY, USA, 1979. ACM.

[3] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning*, ICML '05, pages 89–96, New York, NY, USA, 2005. ACM.

[4] ClearTrip. `http://www.cleartrip.com/`.

[5] Aditya Desai. Translating Natural Language to Formulas using Machine Learning. B.Tech Project Report. He is a co-member of this project, 2013.

[6] Yoav Freund, Raj Iyer, Robert E. Schapire, and Yoram Singer. An efficient boosting algorithm for combining preferences. *J. Mach. Learn. Res.*, 4:933–969, December 2003.

[7] Nidhi Jain. A Generic Framework for Translating Natural Language Descriptions into Domain-Specific Logical Expressions. M.Tech Project Report. She is a co-member of this project, May 2013.

[8] Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '02, pages 133–142, New York, NY, USA, 2002. ACM.

[9] Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. Lexical generalization in ccg grammar induction for semantic parsing. In *Proceedings of the Confer-*

ence on Empirical Methods in Natural Language Processing, EMNLP '11, pages 1512–1523, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[10] Greg Little and Robert C. Miller. Translating keyword commands into executable code. In *Proceedings of the 19th annual ACM symposium on User interface software and technology*, UIST '06, pages 135–144, New York, NY, USA, 2006. ACM.

[11] Greg Little and Robert C. Miller. Keyword programming in java. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, ASE '07, pages 84–93, New York, NY, USA, 2007. ACM.

[12] MakeMyTrip. `http://www.makemytrip.com/`.

[13] Stanford NLP Parser. `http://nlp.stanford.edu:8080/corenlp/`.

[14] Hoifung Poon. Grounded unsupervised semantic parsing. In *ACL*. 2013.

[15] Ana-Maria Popescu, Oren Etzioni, and Henry Kautz. Towards a theory of natural language interfaces to databases. In *Proceedings of the 8th international conference on Intelligent user interfaces*, IUI '03, pages 149–157, New York, NY, USA, 2003. ACM.

[16] David Price, Ellen Rilofff, Joseph Zachary, and Brandon Harvey. Naturaljava: a natural language interface for programming in java. In *Proceedings of the 5th international conference on Intelligent user interfaces*, IUI '00, pages 207–211, New York, NY, USA, 2000. ACM.

[17] Jean E. Sammet. The use of english as a programming language. *Commun. ACM*, 9(3):228–230, March 1966.

[18] Yatra. `http://www.yatra.com/`.

[19] Luke S. Zettlemoyer and Michael Collins. Online learning of relaxed ccg grammars for parsing to logical form. In *In Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL-2007*, pages 678–687, 2007.