Knowledge-Based Question Answering System Final Report

Xu Bing (3035027710)

Table of Contents

Abstract 4
Chapter 1 5
Introduction 5
1.1 Background 5
1.2 Project Objective 6
1.3 Report Overview 8
Chapter 2 10
System Construction 10
2.1 Semantic representation 10
2.2 Knowledge base 11
2.3 SEMPRE Investigation 13
2.2.1 Main theme 13
Chapter 3.....15
System Architecture 15
3.1 Resource loading and options setting 16
3.2 Preprocessing 17
3.3 Basic learning loop 18
3.4 Parser and feature extractor 18
3.5 Learner 19
3.6 Executor and output 19
Chapter 4 20
Feature Representation 20
4.1 Definition and significance.....20
4.2 Feature model and implementation 23
4.3 SEMPRE feature Specification 27
4.4 Motivation and improvements 32
Chapter 5 35
Model and Learner 35
5.1 SEMPRE modeling 35
5.1.1 Log-linear model 35
5.1.2 Parameter estimation model 36
5.1.3 Optimization/Parameter finding method 38
5.2 SEMPRE Learning Algorithm Implementation 39

5.3 Motivation to attempt with linear SVM 41
5.3.1 Problem Statement 41
5.3.2 Log-linear vs SVM 42
5.3.3 SVM 43
5.3.4 Linear SVM Implementation 44
Chapter 6 45
Experiments and Results 45
6.1 Free917 Performance 45
6.2 Our Attempts / Experiments 46
Chapter 7 48
Difficulties & Further development 48
7.1 System components 48
7.2 Difficulties & Problem solving 48
7.3 Further development 53
7.3.1 Parallel Computing/ Distributed System 53
7.3.2 Querying Logic 53
7.3.3 System Components and Debugging 54
Chapter 8.....55
Acknowledgement 55
Chapter 9.....56
References 56
Chapter 10 58
Appendix 58
10.1 - SVM.java 58
10.2 SEMPRE Class specification 63

Abstract

Knowledge based question answering system emerges as an alternate way to help people get useful information in data explosion era. Users can ask questions in human natural language and get back the answers also in natural language.

In this project we model the core of a KBQA system as a supervised machine learning system which takes in question-answer(annotated) training examples and output a semantic parser, which in fact is the optimised weight vector of the feature representation of the data. Thus two main parts of the system are feature extraction and machine learning. The project is based on Stanford SEMPRE and we focus on improvements of the original system in these two aspects. New features in respect to the original feature categories are added and linear SVM is implemented to provide an alternative algorithm other than the original log linear regression model.

Chapter 1

Introduction

1.1 Background

We live in the era of big data which call for more efficient strategies in processing data and the tremendous development in information technology in return led to more explosion of data. We tend to use search engine when we want to search for the information we need.However there exists the situation that we only want to have a quick precise answer instead of large number of related documents or websites. Question Answering system make up for this scenario. It provides user with an interface where users are able to ask questions and retrieve answers using natural language(NL) queries, instead of relying on the keyword-based information retrieval mechanism. This brings another benefit as providing the more natural human-machine interaction feelings by using natural language.

Nowadays, researches have been done on knowledge based question answering systems where the possible candidate answers are retrieved from an ontology and ranked to select the best for user. Knowledge base, also seen as ontology, is the structured knowledge representations. The rise of various knowledge base(KB) like Freebase, DBpedia and YAGO also provides rich source of structured information in almost all knowledge fields to build a QA system.

The project aims to implement a knowledge based question answering system and do future improvements on some specific aspects of the KBQA system.

1.2 Project Objective

There are three objectives of this project:

- ***Implementation of a basic knowledge-based question answering system;***
- ***Improvements on the feature parts;***
- ***Improvements on the learning algorithms;***

The basic requirement of the project is to implement a demonstrable knowledge based question answering system. After research on the difference of open KB and curated KB we chose the latter one as scope. SEMPRE framework from Stanford is used and based on this we worked on two aspects to improve the system performance. The first one is the feature part. As each of the natural language utterance should be represented as a set of features for computer to understand. The more

effective features you can extract from the process of derivation, the more informant you can dig out from the semantic meaning thus better you can describe the utterance and do the parsing. A few new features with respect to the original feature type are added here, including some for semantic function usage and Bridging process. Details about the feature extraction will be illustrated in Chapter 4. The second improvement direction is the learning algorithm which is important for the performance and effectiveness of candidate answer ranker. The original SEMPRE tool adopts log linear model and use stochastic gradient descent to optimise the objective function. However, log linear model is simple and efficient but may not be the best choice to train the the feature weight vector. We tried another algorithm alternative linear SVM to implement the learner part of the system. Details about the machine learning part of the system will be stated in Chapter 5.

1.3 Report Overview

This report is composed in 10 chapters and they arranged in the following way:

Chapter 1: Project Overview. We briefly introduce the backgrounds, motivations and objectives for this project we plan to achieve.

Chapter 2: System Construction. In this chapter, we will give an overview about different components within the system including the external knowledge base we use and a few key classes, schemes implemented in the system as we will refer to them frequently through the whole report.

Chapter 3: System Architecture. This chapter is charge of explaining how the system works generally and explains specifically a few key steps throughout the whole procedure.

Chapter 4: Feature Specification. This chapter will go deep into feature part introducing specifically about the workflow of features generation and extraction as well as the new feature we add.

Chapter 5: Learner and Model Specification. This chapter is responsible for explaining for previous settings of learning algorithms and modeling. Furthermore, the new model will be introduced in details.

Chapter 6: Experiments and Results. This chapter will show a few test running results of the implement of our system and a few analysis towards the results.

Chapter 7: Difficulties and Further Improvements. This chapter will give a review of the whole project. A conclusion about the hardship and problem within the process will be displayed together with some suggestions about the further improvements of the system.

Chapter 8: Acknowledgement. In this chapter, we will show our gratitude and appreciation for those people and kind help we accept through the whole project.

Chapter 9: References.In this chapter, we will list the references.

Chapter 10: Appendix, other useful information and some key codes.

Chapter 2

System Construction

2.1 Semantic representation

Semantic representations are generally logical forms which are expressions using a specific defined artificial language. Constants symbols and relational or compositional grammars are used to encode the semantic information behind the utterance. By semantic parsing the utterance to logical forms, we are able to reach much deeper and context-aware understanding beyonds merely the words. Besides, logical forms are crucial when building the natural language interface of knowledge base(here Freebase) as to perform database queries.
The system here adopts Lambda Dependency-Based Compositional Semantics as formal language which is notably simpler than lambda calculus. One example is :

- *Utterance: "people who have lived in Seattle"*
- *Logical form (lambda calculus): λx.∃e.PlacesLived(x, e) ∧ Location(e, Seattle)*
- *Logical form (lambda DCS): PlacesLived.Location.Seattle*

*In the system, the logical forms are constructed by recursively using 'call' which takes a function followed by arguments. In general they are in the form of, (call <function-name> <logical-form-argument-1> ... <logical-form-argument-n>). For example, (call .substring (string "what is this?") (number 5) (number 7)). In addition, we could use lambda expressions to abstract the computation and parsing process. For example,(lambda x (call * (var x) (var x))); it takes a variable x and calculate the square.*

2.2 Knowledge base

Google Freebase(2013) is used in the system to work as the knowledge base. It organizes the knowledge facts in a set of assertions in the form of triples (e1, p, e2) $\in E \times P \times E$, where e1 and e2 represent entities and p represents relationship between the entities, e.g.,

(fb:en.barack_obama, people.person.place_of_birth, fb:en.honolulu)

This assertion contains two entities fb:en.barack_obama and fb:en.honolulu and the relation "people.person.place_of_birth "associating them.All the information in the base is connected by some level of properties. A few entities and properties are selected from the base to illustrate as below:

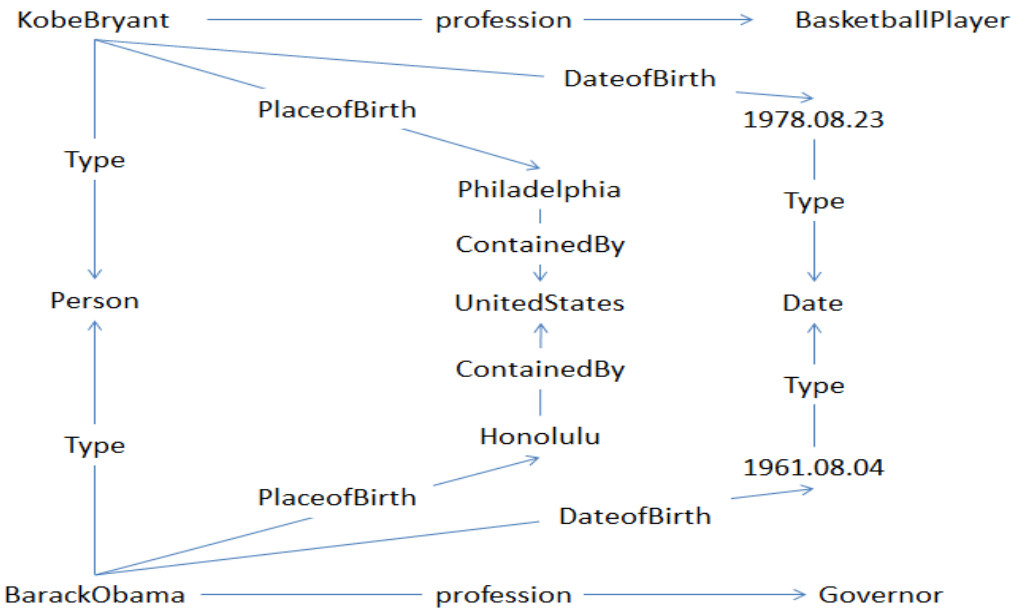


Fig 2.1.a A Demonstration of the Information Representation in Freebase

The whole Freebase has 41M non-numeric entities, 19K properties, and 596M assertions. Besides, Virtuoso is used as the database server which wrap the querying iterations with Freebase

2.3 SEMPRE Investigation

This project is based on Stanford SEMPRE v1.0:

```
@inproceedings{berant2013freebase,
  author = {J. Berant and A. Chou and R. Frostig and P. Liang},
  booktitle = {Empirical Methods in Natural Language Processing (EMNLP)},
  title = {Semantic Parsing on {F}reebase from Question-Answer Pairs},
  year = {2013},
}
```

2.2.1 Main theme

SEMPRE is an outstanding result of combining the semantics compositionality and machine learning. As our research shows, the CCG way of semantic parsing (Combinatory Categorial Grammars), which is led by University of Washington NLP team, put more efforts and constraints on the Grammar formalism and semantics inference procedure. SEMPRE, on the contrary, rely on very few simple grammar rules to do the compositionality and will subsequently over-generate the possible logical predicates. Thus the feature extraction and parameter learning are important as we actually encode the soft rules and categories in the model features to select the candidates of biggest probability.

We can model the whole system as a machine learning system, where It is based on supervised learning where it takes in training examples which are pairs of (question, detonated answer) and outputs the optimised feature weight vector. The feature weight vector is optimised

by a number of training iterations thus to generate the best parsing hypothesises in the form of logical predicates when given a new question utterance.

The existing tradeoffs here is how many details we encode in the representation of semantic meaning, namely, feature representations. If the features are too coarse, we ignore useful information which provide potential guidance when parsing utterances. However, if the features reflect too many details, the machine learning process will finally overfit the training set and result in weaker parsing ability.

Two basic parsing mechanisms of SEMPRE are lexicon alignment and bridging. (Details will be illustrated in Chapter 4). During the parsing process where the system apply various semantic functions onto the lowest level tokens and phrases to get the root derivation and later execute to get the final denotations of the logical forms (from knowledge base), corresponding features are extracted and added to the derivation thus enter the learning iteration to update its bound weight parameter.

Chapter 3

System Architecture

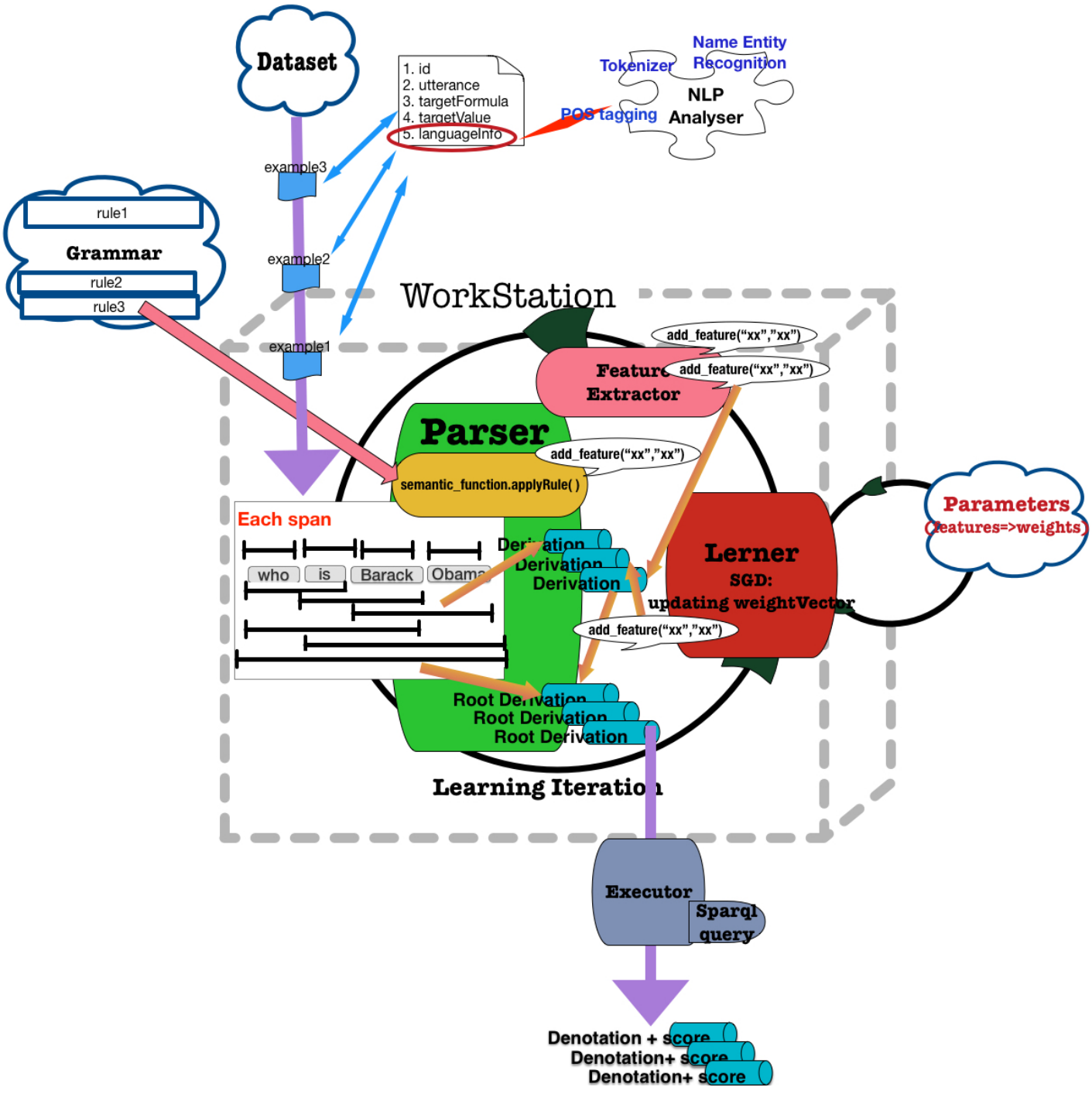


Fig. 3.a System Architecture Illustration

Following the above architecture figure, this section demonstrates in details how the system works by showing the complete work flow.

3.1 Resource loading and options setting

At the start of the execution, system loads in the training and validation dataset in the form of its utterance and target formula.
For instance,

> *{"utterance": "how many tv programs did danny devito produce", "targetFormula": "(count (!fb:tv.tv_producer_term.program ((lambda x (fb:tv.tv_producer_term.producer (var x))) fb:en.danny_devito)))"}.*

We take each of them as one example. Each example is basically a wrap of the utterance. System assigns each of them an id, and records its targetFormula and targetValue.
In addition, system loads in grammar which is a set of rules. SEMPRE defines just a few simple composition rules instead of large amount of manually specified rules or rules from annotated logical forms like in CCG-based systems. The rule is in the form of (rule <target-category> (<source-1> ... <source-k>) <semantic-function>). The left hand side is the target category; the right hand side is a set of source categories. And the semantic function is the work horse used to do the transformation of the derivations. Each rule specifies how a set of derivations is combined to the target category. The rule is used to do the basic lexicon recognition

and compositional work based on the POS tag to decide which spans or which intermediate set should trigger the specific parsing job.
The grammar also includes a few hard-coded rules to recognize the special Wh-word. For example,(rule $Unary (who) (ConstantFn (fb:type.object.type fb:people.person) fb:people.person)) is used to parse "who" to logical form (ConstantFn (fb:type.object.type fb:people.person) fb:people.person) and label it with the $Unary category;
(rule $Unary (who) (ConstantFn (fb:type.object.type fb:organization.organization) fb:organization.organization)) is used to parse "who" to another possible logical formula " (ConstantFn (fb:type.object.type fb:organization.organization) fb:organization.organization)" and label it with the $Unary category.

3.2 Preprocessing

After loading in each examples (basically the utterance and the target formula), System preprocesses the examples with the help of Stanford CoreNLP tool to get the basic syntactic and semantic information of each utterance.
The process includes word splitting, tokenising, lemmatisation of tokens, POS tagging, name entity recognition and tagging. This preprocessing is basic but necessary because the parsing is based on the obtained information and the system relies on the POS tagging and name entity recognition to restrict what words and phrases in the utterance should be mapped to its corresponding lexicon. (Using

LexiconFn). Besides, the tokens are the building blocks of our span-based analyzing construction.

3.3 Basic learning loop

Here we enter the core workstation where the system starts the basic learning iterations. Each utterance of the example, one by one, is processed by parser, during which the corresponding features are added by applying the semantic function and the feature extractor. After each example is parsed and scored by the ranker, the learner updates the weight vector using stochastic gradient descent method.

3.4 Parser and feature extractor

The utterances are firstly preprocessed by the Stanford coreNLP Analyzer as stated above, where they are cut into tokens and tagged with corresponding syntactic information. The system analyses the example by each span including each token, lemmatised token and each phrase, and lemmatized phrase (Clearly reflected in the figure above). Assuming we wrap the parsing process as a derivation which apply rules onto the children derivations (if any). And here is the bottom level of derivation. Until we reach the root derivation where we arrive at the final formula and become able to execute the query onto freebase to get specific value (also called denotation).

During parsing and applying specific rules, the system extracts corresponding features. Details will be stated in the feature specification section later in the report.

3.5 Learner

At the end of one examples' derivation processing (namely parsing), we sort the predicted root derivations within beam size by their score and output the corresponding denotations.

The probability distribution of the derivations of one example, by applying the max likelihood optimization, is used to update the weight vector of the model. Scoring of the derivation is based on the computing the compatibility of the predicted value and the target value of the example. Detailed learning algorithm and implementation will be stated in the learning section later in the report.

3.6 Executor and output

The logical form derived by parser is transformed into SPARQL query and executed onto the copy of freebase using the Virtuoso engine so as to obtain the corresponding denotation. Denotations within beam size and the updated model parameter (mapping from features to weights) are the output of the system.

Each part of the system we specified above cannot be totally separated because they rely on each other to complete the while learning iteration.

Chapter 4

Feature Representation

4.1 Definition and significance

As stated in previous section, we model the core of the system as a machine learning problem with the target to train optimised weight parameters corresponding to features. The crucial piece of the problem is to determine the appropriate representation of the data. Here comes our features. The key idea here is to map each input-output pair (training example, derivation) to a d-dimensional feature vector $\varphi(x, y) \in R^d$ Each dimension here in our KBQA problem can represent some property of our logical forms we obtain or the resulted denotation after execution on Freebase, for example, we number of possible denotations from one specific logical form; it can also represent the relationship between the

input utterance example and the local forms(such as all the possible alignment by lexicon).

We need to strike a balance of the details reflected in the features. This can be illustrated by an example. The problem is to train a model to classify even and odd numbers.We have the training examples of (number, expected category). such as (thirty on3, Odd), (fifty two, Even). (twenty three, Odd), (eighty six, Even). If we extract the feature representation by using no string, we got no information from the training set. If we got all the tokens of the numbers, we take risk of overfitting. Only when we extract the "last word" of the number words can we get the expected training results. The figure is as below:

		Feature representations $\phi(x, y)$		
	(x, y)	'empty string'	'last word'	'all words'
Train	(twenty five, O)	ϵ	five	[twenty, five]
	(thirty one, O)	ϵ	one ~~eight~~	[thirty, one]
	(forty nine, O)	ϵ	nine	[forty, nine]
	(fifty two, E)	ϵ	two	[fifty, two]
	(eighty two, E)	ϵ	two	[eighty, two]
	(eighty four, E)	ϵ	four	[eighty, four]
	(eighty six, E)	ϵ	six	[eighty, six]
Test	(eighty five, O)	$\epsilon \rightarrow$ E	five $\rightarrow$ O	[eighty, five] $\rightarrow$ E

So we could see the significance of extracting appropriate feature representations from the data.

Feature performance takes effects while the parser starts to parse the example question sentence. The general procedure contains two steps: 1. coarse parse; 2. bridging.

In coarse parse state, the basic principle is text matching and alignment searching. Each question is parsed into tokens (single word) first. By alignment, some tokens are assigned to some entities/unaries in knowledge base, e.g., California will be assigned to fb:en.california; and then parser generates all the possible spans over those tokens (forming phrases) to find better text alignments. For phrase alignment, not only can entities or unaries be assigned, binaries can also be assigned, e.g., born in can be assigned to people.person.plaace_of_birth. After these two steps, a few tokens and phrases should be able to map to some entities and relationships between entities may be discovered as well. During this procedure, a few features like *alignment* will be added to the derivations.

However, in some cases, the relationship between logical forms cannot be declared clearly due to many reasons, for example, the verb in the sentence is too light (is, are, am, go, come, etc.,) to generate the binaries. Then bridging schema will be adopted. Bridging will do type check to

those aligned logical forms and search for all the possible binaries which fulfill the type requirements. By applying various semantic functions, a number of derivations are composed. During this procedure, specific features are added to derivations as well.
In conclusion, various features are added to derivations during the whole process of derivation generation. As different rules are applied, features are added accordingly. Furthermore, due to ambiguity and polysemy of the questions, the parser intends to over-generate the derivations. With different weights assigned to different features, the system picks up the one which ranks on the top.

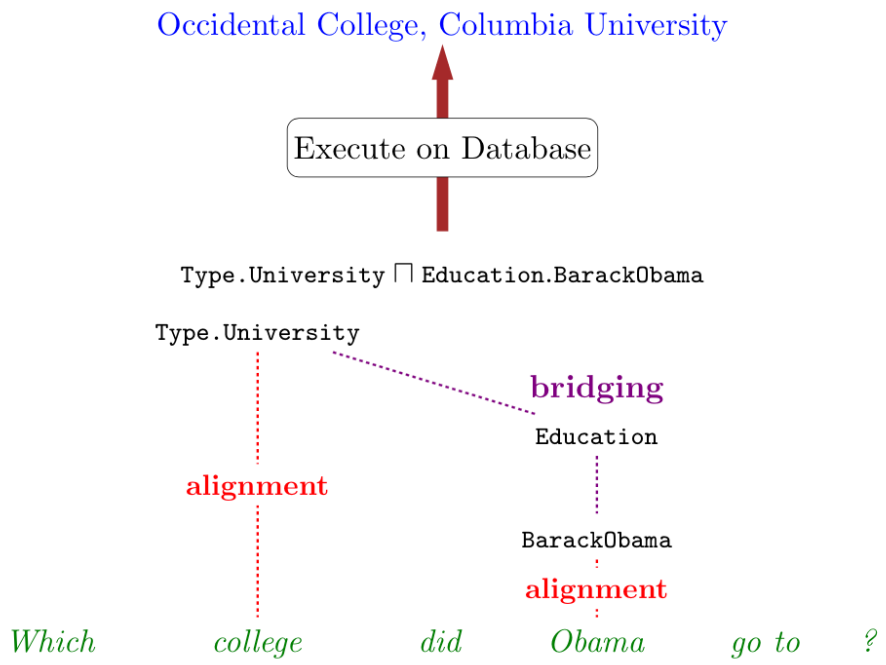
4.2 Feature model and implementation

Limited by our knowledge in Natural language processing and semantics, we stick to the feature model of original SEMRE system. There are two strategies adopted by SEMPRE to get a relatively manageable size of predicate size for each utterance. One is by lexicon alignment and the other is by Bridging.

A lexicon L is a mapping from the natural langue phrases/tokens to its corresponding logical predicates with the generated features during the parsing process.

The system first do a coarse parse of each training example, which maps the lowest level unit - (tokens , phrases, lemmatised tokens, lemmatised phrases) to their corresponding unary entities or even binary relations by text matching and lexicon searching. These tokens , phrases, lemmatised tokens, lemmatised phrases of the input utterances are produced by Stanford NLP analyzer during the preprocessing(can be seen in Chapter 3 system architecture.) and the scope is confined to tokens and phrased with ,say, targeted valid POS tagging labels for example. Within this phrase, the features in “Alignment” category is extracted and added to the derivation. For instance, log of the number of entity pairs that occur with the phrase r. Barack can be mapped to fb:en.barack_obama or possibly fb:en.honolulu (by incorrect info in the lexicon). So here the number is 2. Another feature instance is text similarity, namely, the phrase r1 is equal/prefix/suffix of s2 which indicate whether there is overlap of r1 and r2. (More features will be illustrated in section 4.3)

However, there are situations that some predicates are expressed by very ambiguous or light word like " have" or "go". For example, when we execute the query " Whose capital is Beijing?" we can get China as the correct answer. But when we execute the query as " Which country is Beijing in ?",we get a set of incorrect answers like " Peking universities" or even "Renmin university affiliated middle school". Because we cannot obtain the rich information from natural language here, we need to do the bridging from its neighbour logical forms. For example in the below figure, from the neighbouring logical forms, Type.University and BarackObama, we can infer the actual logical form Education which we miss from the original natural language query "Which college did Obama go to?"



Following the coarse parsing, there is the composition process where the system apply the various semantic functions recursively onto the semantic building blocks (unaries/binaries/entities) and their current combining results and added the generated features to the derivation. The practical implementation is to execute this whole process on each span of the input utterance adopting dynamic programming to improve performance. However, this process is still the bottleneck of the system performance. For example, whose capital is Beijing? (unlemmatised version below. The actual implementation contains not only the tokens/ phrases, but also the current derivations, namely the logical forms and accompanied features)

End idx\Start idx	0	1	2
0	whose		
1	whose capital	capital	
2	whose capital Beijing	capital Beijing	Beijing

When we eventually reach the root derivation [eg. the comlemte one—whose capital Beijing] , the result logical predicates then enter the learning iteration to update its corresponding weight parameter. Thus after a few learning iterations, the system output a optimised weight

vector which can best describe this training set with respect to the specified feature representations.

4.3 SEMPRE feature Specification

Following are the detailed specifications of all the features adopted in the SEMPRE system.

[where IF denotes indicator feature (with value 0 or 1) and GF denotes general feature(real numerical values)]

Features that apply to all derivations:

- denotation feature (IF): this feature is in the domain "denotation". It measures and describes how many denotations are returned from the knowledge base after executing the formula (for the initialization, token and phrase are using themselves as formula) of this derivation. This feature is categorized into four kinds: 0 denotations found, 1 denotation found, 2 or more denotations found.

- Whtype features (IF): this feature is in the domain "Whtype". It tests whether the first token of the question sentence starts with

"Wh", which means a few words indicating questionings in English. If so, then the category of the word follows the Wh word is being kept as part of the features as well in order to generate some similarities between different examples.

- Option count features (IF): this feature is in domain "opCount" and it measures how many semantic functions are used in generating the current derivation if the current derivation is constructed by semantic functions with logical forms. However, it is not included Lexicon function if the derivation is constructed from lexicon file. (New features are added regarding to this.)

- Conjoin lemma and binary features (IF): this feature is in domain "lemmaAndBinaries". By the default setting, this feature is disabled. This feature is checking the number of non-trivial POS tags, e.g., noun, verb, preposition, excluding the BE word (e.g., am, is, are…) and the binaries in the example via executing the formula. By using the coreNLP analyzer, every word in the derivation is tagged with a POS (part of speech label). It is taking the POS of the words contained in the derivation into

consideration and counting. The binaries within the derivation are kept in this feature as well.

Features that apply to certain rules (the following features only apply to certain situations while some semantic functions are called to combine logical forms):

- Constant (IF): this feature is in domain "constant". This feature is added while *Constant Function* is applied to construct new derivations of the example. It keeps the information of the aligned phrase and the constant formula.

- Bridge function feature (GF/IF): there are three features in domain "BridgeFn". Two of them are indicator features that record the bridging category (with it is bridging entities, unaries or binaries) and pattern information (header first or modifier first, the tags of the logical forms bridged). The last feature is added while doing bridging (details of bridging will be illustrated later in this section). The last feature is a general feature that keeps the count of the entities and unaries that only fill the modifier's type check but not including the header type. (New features are added regarding to

this)

- POS skipping (IF): this feature is in domain “skipPos”. This feature is to keep the POS tags of the each token in the child derivations that we skip while applying *Select Function* (Select Function is used only to select certain logical form to be effective and ignore the others).

- POS joining (IF): this feature is in domain “joinPos”. It keeps the joint unaries and binaries information if the *Join Function* is successfully operated to generate new deriavtions. If failed, a feature in domain “typeCheck” recording the join mismatch is added to the derivation.

- Type Check (IF): this feature is in domain “typeCheck” as the one that may be generated in *Join Function*. It is added to the derivation if the types are mismatched while performing *Merge Function*.

- Basic Stats (GF/IF): There are three features included in this in domain “basicStats”. The first one keeping the popularity of the

token/phrase to be searched in the knowledge base via *Lexicon Function*. This feature is an indicator feature. The other two are keeping the information of the source of the logical form found and whether the form found is closed enough with the original token/phrase or not. However, for entity and binary lexicon alignment, whether the entity or binary assigned are equal or not is not clear; for unary, the lexicon alignment type is not recorded. The source of the logical form assigned for entity is not specified. (Features are added regarding to this.)

- Tokens Distance (GF): this feature is in domain "tokensDistance". It keeps the records of the denotation of the entities that are aligned.

- Lexicon Alignment (IF): this feature is in domain "lexAlign". It keeps the record of the token/phrase aligned with the lexicon formula of the corresponding lexical entry.

- Alignment Score (GF): this feature is in domain "alignmentScores" domain and it records the alignment score of a certain feature to

the feature vector of the token/phrase

Three types of features implemented in SEMPRE:

- indicator feature: with value 0 or 1 denoting whether a specific requirement is satisfied.
- general feature: with concrete numeric values
- dense value: system specified values during the execution.

When certain conditions are satisfied (such as a grammar rule is used in a semantic function) , the features are added into the derivation and take effect when we compute the score of the logical predicates (in practice, the denotation in the system).

4.4 Motivation and improvements

We can see the significance of feature representation to the learning problem. So we can make experiments to add more new features to get potential improvements in the representation of training data.

Here in below are the new features added to the system with respect to the original category as to improve the describing ability of the feature representation.

These features are added by my teammate Marcus(Ma Zhiyu):

Operation Count (IF): this feature is in domain “opCounter”. It has accomplished a full recording for all possible semantic functions (ConcatFn, LexiconFn, SelectFn) that may be used in the process of derivation generation instead of the partial record before.

- Bridge Function (overlap popularity, equal, IF): this feature is in domain “bridgeFn”. It is recording the number of all the binary relationships that fulfill with the type check requirements for both header and modifier while bridging is performed instead of only checking the modifier’s type before.

- Basic Stats (equal, entity.source, lexicon type alignment, IF): There are four features in domain “basicStats”; .equal measures whether the entity or binary are equivalent to the logical forms assigned based on the form distances. Entity.source is also added for indicating the logical form source for the entity. Unary lexical alignment type feature is also added to keep the information of the lexical type of the unary.

- Lexical Alignment (lexAlign, IF): this feature is in domain "lexAlign", which keeps the records of the formula source of the assignment.

Chapter 5

Model and Learner

5.1 SEMPRE modeling

5.1.1 Log-linear model

As to use the whole contextual information for the language modelling problem, the number of features could be very large, for example, as for unigram, we possibly extract feature for each word token. The usual linear interpolation becomes unwieldy when we are faced with the modelling and estimation of the distribution over all the variables of interest. So SEMPRE uses log-linear model for the language feature combination modelling.

The definition is

$$p(y|x;v) = \frac{\exp(v \cdot f(x,y))}{\sum_{y' \in \mathcal{Y}} \exp(v \cdot f(x,y'))}$$

where

• A set X of possible inputs. (Examples in our system)

• A set Y of possible labels. The set Y is assumed to be finite. (Denotations in our system)

• A positive integer d specifying the number of features and parameters in the model.

• A function f: X × Y → Rd that maps any (x, y) pair to a feature-vector f(x, y). (the Feature Extractor in our system)

• A parameter vector v ∈ Rd (the model parameter which maps features to weights, and should be trained by our learner.)

The model can also be transformed to the form of

$$\begin{aligned} \log p(y|x;v) &= v \cdot f(x,y) - \log \sum_{y' \in \mathcal{Y}} \exp\left(v \cdot f(x,y')\right) \\ &= v \cdot f(x,y) - g(x) \end{aligned}$$

where

$$g(x) = \log \sum_{y' \in \mathcal{Y}} \exp\left(v \cdot f(x,y')\right)$$

5.1.2 Parameter estimation model

For each our training example, the question-answer pairs (xi,yi). The log conditional probability as indicated above log(p|x;v), given the weight mapping vector, is higher means that the model fits this specific example better.

So when we summing over all our training examples as a function of the weight vector L(v), we get:

This could be seen as a measure of how the weight mapping parameter

$$L(v) = \sum_{i=1}^{n} \log p(y^{(i)}|x^{(i)}; v)$$

fits our total training set. Intuitively, we want the parameter to be the one where we get the largest L(v), which means the maximum likelihood of the training example.

Besides, due to that our feature should be very sparse and the number of

$$v_{ML} = \arg\max_{v \in \mathbb{R}^d} L(v)$$

features can be pretty large, the trained parameters is easily to be over-fitting and there is possibility that some feature weights become very large, even diverging to infinity, we apply the regularization term to add penalty for large parameters. SEMPRE chooses 2- norm regularization here. So the final objective function becomes,

$$L'(v) = \sum_{i=1}^{n} \log p(y^{(i)}|x^{(i)}; v) - \frac{\lambda}{2} \sum_{k} v_k^2$$

5.1.3 Optimization/Parameter finding method

Because this is convex function, we can use gradient-ascent like method to diverge to the global maximum.
The basic gradient ascent algorithm is

Initialization: $v = 0$

Iterate until convergence:

- Calculate $\delta_k = \frac{dL(v)}{dv_k}$ for $k = 1 \dots d$
- Calculate $\beta^* = \arg\max_{\beta \in \mathbb{R}} L(v + \beta\delta)$ where δ is the vector with components δ_k for $k = 1 \dots d$ (this step is performed using some type of line search)
- Set $v \leftarrow v + \beta^* \delta$

Figure 2: A gradient ascent algorithm for optimization of $L(v)$.

However, computing the cost and gradient for the entire training example set can be very slow and intractable on a single machine because the dataset is too big to fit in main memory. So the system cannot use these kinds of batch methods. Instead, here the system chooses online learning algorithm as stochastic gradient ascent. This algorithm updates the parameter after justing |batchSize| trading

examples instead of all. So the new update is given by

$$\theta = \theta - \alpha \nabla_\theta J(\theta; x^{(i)}, y^{(i)})$$

with a pair of (xi,yi) of the training example.
In the original SEMPRE system, the batch size is set to one. And it uses a variance of stochastic gradient ascent, AdaGrad, which uses per-feature adaptive step size control for each update.

5.2 SEMPRE Learning Algorithm Implementation

We explain the log-linear model and the practical iterative way (SGD) SEMPRE uses to model the learning process in the previous subsection. This subsection will state the detailed implementation of the learning algorithm in the original SEMPRE.
As the system uses stochastic gradient descent algorithm, in each iteration, it will loop through the randomly shuffled examples and update the weight vector with each example.
For each example, system computes the predicted scores for all the root derivations by "doing dot product of the feature vector and the weight vector". After exponential normalization, we record the probability distribution of the derivations. Then system compares the predicted derivation's value to the target value to compute the compatibility, thus

we record the true probability. (Where the incorrect derivations' probability is nearly 0 and the correct ones get its share of 1.)

For each derivation, we move forward (real_probability - predicted_probability) *step_size on the weight of its corresponding features.

So it is obvious that we increase the expected features' weight and decrease the less expected features' weight value. Hopefully, if the step size is valid, according to the attribute of stochastic gradient descent, the weight vector will finally converge to the one which fit our training example best.

The system didn't include the regularization term in the objective function which should be seen as a deficiency. We computed the 2-norm value of the weight vector each time and log to see that they are obviously increasing, even to 7 or 8, which shows that it is bad choice to model the process without including the regularization term which could not avoid the possibility of very large parameters.

5.3 Motivation to attempt with linear SVM

5.3.1 Problem Statement

Here we propose another perspective to see our KBQA system — taking it as a whole machine learning task. Each derivation, i.e. the parsing process is attaching features to the specific example. The system cannot understand the natural language sentences or the logical forms. What the system sees is a set of features belonging to the object we analyze; in other words, the features are the presence of the observed example.
So the training process can be seen as to use observation/sample pairs to find a good classifier to do the classification between all the derivations from the observed example.

Namely, find the weights of the features. Both exponential way- log linear and the marginal way - SVM could do the classification job[1].
[Log-Linear Models Noah A. Smith∗ Department of Computer Science / Center for Language and Speech Processing Johns Hopkins University nasmith@cs.jhu.edu December 2004]

1 Stanford NLP slides can be found: web.stanford.edu/class/cs124/lec/Maximum_Entropy_Classifiers

5.3.2 Log-linear vs SVM

The original SEMPRE uses discriminate log-linear model, which is a widely-used tool in NLP classification tasks (Berger et al.,1996; Ratnaparkhi, 1998). It has flexibility as its advantage, for example, it can take features of any form and the features can overlap or just predicting parts of the data instead of the whole. This is necessary for the task because SEMPRE uses relative simple and less rules for the composition, so it needs large amount of overlapping features to encode the soft rules. Except for its advantage, there are also some distinctions between the maximum likelihood parameter estimation of log-linear model and support vector machine algorithm which gives us motivation for attempt of SVM for SEMPRE.

As we have loose composition constraints on the derivation, we end up over-generating the predicted logical forms. We also take advantage of large number of features. Therefore, we have high dimensional spaces. SVMs are still efficient in high dimensional.

Additionally, like with any linear regression, every training point has a certain influence on the estimated log linear function. SVM's tend to only consider points near the margin and are formulated so that only points near the decision boundary really make a difference. Points that are "obvious" have no effect on the decision boundary. This can be very different from logistic regression, as "obvious" points can sometimes be very influential.

And professor Andrew Ng in the ML course also advice to use linear SVM when the dimension of feature vector is much larger than the number of data set.

All the above reasons give us a motivation to try using linear SVM to possibly improve SEMPRE's performance.

5.3.3 SVM

Support vector machines belongs to a family of generalised linear classifier where the classification (or boundary) function is a hyperplane in the multidimensional feature space that separates cases of different class labels.

Details of SVM should be seen in corresponding tutorials and are not stated here.

[Good SVM Tutorial recommendation — Classification,Regression and Ranking Hwanjo Yu . Sungchul Kim,from Data Mining Lab, Department of Computer Science and Engineering, Pohang University of Science and Technology, Pohang, South Korea]

The Definition can be interpreted as: [graph here]

$$\min_{w,b} \frac{1}{2} \| w \|^2$$

$$\text{subject to } y_i(w^T x_i + b) \geq 1,$$

5.3.4 Linear SVM Implementation

We are still using stochastic gradient descent algorithm to do the weights estimation task of the SVM.
The stochastic gradient algorithm for SVM can be stated as:

$$Q_{\text{svm}} = \lambda w^2 + \max\{0, 1 - y\, w^\top \Phi(x)\} \qquad w \leftarrow w - \gamma_t \begin{cases} \lambda w & \text{if } y_t\, w^\top \Phi(x_t) > 1, \\ \lambda w - y_t\, \Phi(x_t) & \text{otherwise.} \end{cases}$$

Features $\Phi(x) \in \mathbb{R}^d$, Classes $y = \pm 1$
Hyperparameter $\lambda > 0$

We implemented learn SVM model as a separate SVM class which adopts the same method name updateWeights() and updateCounts() to provide the same service to learner as to maintain the work flow of original learner class to the largest extent. Some details of the implementation may not be smart due to limited experience and new exposure to machine learning, but the modification is bug-fixed (The code could be seen in the appendix). However, we haven't got positive experiment results till now.

Chapter 6

Experiments and Results

6.1 Free917 Performance

Due to the limitation of server hardware, we cannot fully reproduce the free917 training results. What we are going to present below is based on a relatively small sample size. We have tested various combinations of training parameters setting (whether more iterations with smaller training example size --- 30 example / iteration * 5 iterations, or less iteration with bigger training example size --- 150 example / iteration * 1 iteration). The average time cost for the whole training period is about 40 hours.

According to the EMNLP 2013, the system accuracy for free917 is 62%.

Here below is a little slice of all the 42493 features it generates:

```
lexAlign :: live --- fb:music.artist.origin -4.625999740302934
lexAlign :: name --- !fb:people.family_name.people_with_this_family_name
-4.626392183448551
```

```
lexAlign :: puerto rico --- fb:en.puerto_rico  -4.630450822151956
lexAlign :: do --- !fb:book.book_edition.author_editor -4.633164541488024
lexAlign :: sport --- (fb:type.object.type fb:sports.sport)    -4.636516283542151
BridgeFn :: binary=fb:base.politicalconventions.convention_speech.date
-4.665402027669657
BridgeFn :: inject_order=modifier-head,pos=N-IN    -4.685682362272001
lexAlign :: of government --- fb:en.head_of_government -4.727178619949043
whType :: token0=when,type=fb:people.person    -4.774340619945352
lexAlign :: thailand --- fb:m.05ml1tx  -4.795771086682366
```

6.2 Our Attempts / Experiments

Due to the small sample size we used, our experiments with the modifications in feature and modelling parts did not result in positive feedbacks. We got the highest accuracy rate of 35% on a smaller set of test cases as well. Here is a conclusion of that training:

```
Evaluation stats for iter=2.dev {
  parsed = 1/ << 1 ~ 0 >> /1 (7)
  numTokens = 6/ << 8.714 ~ 1.666 >> /11 (7)
  maxCellSize = 557530@[$BaseSet 0:7[how many teams participate in the uefa]]/ << 1383379.286 ~ 564238.336 >>
  fallOffBeam = 1/ << 1 ~ 0 >> /1 (7)
  coarseParseTime = 4/ << 16 ~ 14.152 >> /47 (7)
  parseTime = 38934/ << 435164.286 ~ 312760.511 >> /1082170 (7)
  totalDerivs = 1544571/ << 9285121.714 ~ 6368452.264 >> /21163392 (7)
  valueCorrect = 0/ << 0.143 ~ 0.350 >> /1 (7)
  formulaCorrect = 0/ << 0.143 ~ 0.350 >> /1.000 (7)
  correct = 0/ << 0.143 ~ 0.350 >> /1.000 (7)
  oracle = 0/ << 0.143 ~ 0.350 >> /1 (7)
  partCorrect = 0/ << 0.143 ~ 0.350 >> /1.000 (7)
  partOracle = 0/ << 0.143 ~ 0.350 >> /1 (7)
  meanBeamJump = 3867.813/ << 32104.146 ~ 42571.192 >> /132543.887 (7)
  meanRootBeamJump = 0/ << 651.905 ~ 1128.684 >> /3142 (7)
  numCandidates = 500/ << 500 ~ 0 >> /500 (7)
  parsedNumCandidates = 500/ << 500 ~ 0 >> /500 (7)
  exec-cached = 0/ << 0.217 ~ 0.412 >> /1 (3500)
  exec-time = 1/ << 12.671 ~ 14.332 >> /303 (2742)
  exec-error = 0/ << 0 ~ 0 >> /0 (2742)
  correctIndexAfterParse = 0/ << 0 ~ 0 >> /0 (1)
  correctMaxBeamPosition = 484/ << 484 ~ 0 >> /484 (1)
  correctMaxUnsortedBeamPosition = 433641/ << 433641 ~ 0 >> /433641 (1)
  entity = 0/ << 0.502 ~ 0.500 >> /1 (707)
  unary = 0/ << 0.323 ~ 0.468 >> /1 (65)
  binary = 0/ << 0.568 ~ 0.495 >> /1 (95)
}
```

Fig 6.2.a Evaluation of the Training Process

Furthermore, this is the screenshot of part of all the 4568 features:

```
lexAlign :: verizon ? --- fb:en.verizon_communications  1.0540046172892115
lexAlign :: is okemos michigan --- fb:en.okemos 1.0538955875513127
lexAlign :: jessica mcclure --- fb:en.jessica_mcclure   1.052971446183791
lexAlign :: flow --- fb:geography.river.basin_countries 1.0529654072369223
lexAlign :: cheat --- (lambda x (!fb:fictional_universe.romantic_involvement.partner (!fb:fictional_
lexAlign :: danny ainge --- fb:en.danny_ainge   1.052252501130089
basicStats :: unary.ALIGNMENT   1.0520221115076402
lexAlign :: does cuba --- fb:en.cuba    1.0509895547351105
lexAlign :: carmen electra --- fb:en.carmen_electra 1.0503980589640283
lexAlign :: dodgers --- fb:en.los_angeles_dodgers   1.0503123028849115
lexAlign :: kermit --- fb:en.kermit_the_frog    1.050303959587919
lexAlign :: of israel people --- fb:en.israel   1.0502192538908777
lexAlign :: kris humphries --- fb:en.kris_humphries 1.0494542887454563
lexAlign :: julian fellowes --- fb:en.julian_fellowes   1.0493849005015388
lexAlign :: mount st helens --- fb:en.mount_st_helens   1.049088269539869
lexAlign :: who was michael jackson 's --- fb:en.michael_jackson    1.0486910481589078
lexAlign :: the pittsburgh pirates --- fb:en.pittsburgh_pirates 1.048616209328013
```

Fig 6.2.b Features Generation

Chapter 7

Difficulties & Further development

7.1 System components

Three components of the system are Stanford SEMPRE, Google's KB freebase and the Virtuoso-opensource graph database engine. We downloaded the source code of SEMPRE and compile it.

7.2 Difficulties & Problem solving

- Deprecated Freebase API.

Google's Freebase API has became deprecated on June 30, 2015, so we

have to downloaded the whole database and decompressed it. The database takes up about 80G so we have to apply for more disk space.

- Virtuoso installation problem.

Ubuntu 14.04 uses bison 3.0.2. We failed to make every time when trying to install virtuoso. After searching we locate the problem to be “bison 3.0.2 which generates slightly different code for a reentrant parser. “

We then apply a patch to update the “getdate.y” file to resolve this problem with bison v2.3 and above.

- Bugs in scripts in SEMPRE2.0

“ ./run @mode=query @sparqlserver=localhost:3001 -formula '(fb:location.location.containedby fb:en.california)'

returns an empty list instead of three items.” which turns out to be bugs in the virtuoso script.

- Switched to SEMPRE1.0 to reproduce the results of paper Jonathan Berant, Andrew Chou, Roy Frostig, Percy Liang. Semantic Parsing on Freebase from Question-Answer Pairs. Empirical Methods in Natural Language Processing (EMNLP), 2013.

- Out of memory when trying to run the training process each time.

The program got exception and failed many times when we tried to train the model. After going through the log, we locate it to be OOM problem.

Originally we only have 2G memory. We applied for 4G but it still didn't work. Then we tried to get more memory but IT service run out their resources and could only extend our memory to 8G as maximum. Unfortunately it still doesn't work.

We asked for Pengcheng (Professor Ben Kao's Mphil student) if he has any virtuoso installed server and Pengcheng kindly let us use his galaxy007 server to do the sparql query.

- Some system tricks we learned due to obstacles.

  - screen.

We need to run the program to train the model parameter by machine learning, which takes a long time. We have to leave the server and let the program keep running. So we have to solve the problem that this program process is the child process of our ssh bash window. As we logged out from the server, the process is also killed.

We tried to use "nohup" and "disown" at first but due to

that the SEMPRE is started by shell script, the relationship between processes is multi-layer. So this method doesn't work.

Finally we successfully used screen to compete the task.

- Port forwarding

We borrowed the galaxy server to do the database query. However we need to do port forwarding to redirect our query requests from 3093 port of our FYP server to the corresponding port of the galaxy server. This task is not easy and takes a lot of time to search. The reason is that the galaxy server is within a firewall, thus our FYP server could only access it through gatekeeper.cs.hku.hk. Then we need to ssh to a cluster ip and then to the server ip.

Finally we learned how to do multi-layer ssh tunnelling.

- Connection Exception

After we successfully started running the system again and sending request to the virtuoso service installed on the galaxy server. But weirdly the process failed a few times due to connectionException. We checked that the virtuoso server is well running on the galaxy server and the ssh tunnel's process is well running.

In the end the problem is located to be " possibly long time no message sent to ssh server then the ssh maybe frozen. We set a ssh flag variable to let the client send of message to server after a specific duration. Then the problem is solved.

- Hard to debug for the code

When we changed some code, it is hard to debug due to everything in SEMPRE is dynamic, which means that minor typos result in empty results rather than errors. So the only method is to print out more log information to help locate the problems.

- SEMPRE less resources

SEMPRE2.0 is relatively young so that there are some bugs have not been fixed and less resources could show related problems and solutions.

7.3 Further development

After we go through and investigate the whole SEMPRE system, some potential enhancements can be done to the system in order to improve the systems performance. Below is the list of the potential improvements of the system.

7.3.1 Parallel Computing/ Distributed System

One of the biggest problems we encounter in the project is the execution time and resources the system takes up. Usually, it takes more than 10 minutes to finish parsing one example sentence and it operates slower as the memory resources are gradually taken up. So the system could be reconstructed into a parallel system that can be run on multi-core servers or even make it running on distributed systems. We know the difficulty of system reconstruction is huge, so we only list the possible plan here for reference.

7.3.2 Querying Logic

As far as we understand towards to the system, for every possible logical form, the system evaluates its correctness by executing both the target form and the candidate form in the knowledge base and compares the denotation from them. If the denotations accord, the system will treat that candidate logical form is true. However, this approach is risky

when a totally wrong logical form can return the same denotation as the correct one. In that case, the form is treated as correct, e.g., (lambda x (fb:capital.country (var x))) (fb:en.china) is the correct logical form, but a logical form like below is generated:
(fb:people.person.place_of_birth fb:en.mazhiyu) (I assume that Freebase has the information about me and I was born in Beijing.)
According to the evaluation schema, the system will consider the second logical form to be correct as it returns the correct denotation. However, it is obvious that logical form is incorrect. Although this kind of error can be reduced by more example input and more iterations of training, it is better to have some more solid evaluation approach. Comparing logical form is not a good either; logical forms can be different to express the same meaning.

7.3.3 System Components and Debugging

The various components of the system are coupled to a great extent so it is hard to analyse and evaluate the components separately when we encounter bugs. For example, the current learning algorithm is restricted to change as there's big coupling issue in the learner class and the learning algorithm.
Besides, in order to make it easier for developer to debug, it should be better if a few necessary unit-test files can be provided.

Chapter 8

Acknowledgement

We are grateful to acknowledge Prof. Ben Kao as our supervisor for his guidance and lead during the course of this project work and the proposal of this FYP topic with little limitation.

We extend our sincere thanks to Mr. Pengcheng Yin who continuously helped us throughout the project.

We are also grateful to K.P Chan as FYP administrator providing the guidance during the whole year.

Also thanks for members from the IT support team providing us with continuous help.

We would also like to thank Stanford NLP team for writing the very useful SEMPRE (Semantic Parsing with Execution http://nlp.stanford.edu/software/sempre/)

Chapter 9

References

Kwiatkowski, L. Zettlemoyer, S. Goldwater, and M. Steedman. 2010. Inducing probabilistic CCG grammars from logical form with higher-order unifi-cation. In Empirical Methods in Natural Language Processing (EMNLP)

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on Freebase from question-answer pairs. In Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, pages 1533–1544, Seattle, Washington,USA, October. Association for Computational Linguistics.

Mihai Surdeanu,Massimiliano Ciaramita,Hugo Zaragoza† earning to Rank Answers to Non-Factoid Questions from Web Collections

Qingqing Cai and Alexander Yates. 2013. Largescale semantic parsing via schema matching and lexicon extension. In Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ages 423–433,Sofia, Bulgaria, August. Association for Computational Linguistics.

Michael Collins. Log linear models.

Semantic Parsing with Execution

http://nlp.stanford.edu/software/sempre/

SEMPRE documentation.

https://github.com/percyliang/sempre/blob/master/DOCUMENTATION.md

SEMPRE tutorial.
https://github.com/percyliang/sempre/blob/master/TUTORIAL.md

Yoav Artzi, Nicholas FitzGerald and Luke Zettlemoyer ACL 2013 Tutorial.Semantic Parsing with Combinatory Categorial Grammars.

Noah A. Smith, December 2004] ,Log-Linear Models

Abdullah M. Moussa and Rehab F. Abdel-Kader,QASYO: A Question Answering System for YAGO Ontology

Hwanjo Yu . Sungchul Kim from Data Mining Lab, Department of Computer Science and Engineering, Pohang University of Science and Technology, Pohang, South Korea. Classification, Regression and Ranking,

L´eon Bottou, Microsoft Research, Redmond, WA, Stochastic Gradient Descent Tricks.

Chapter 10

Appendix

10.1 - SVM.java

```
package edu.stanford.nlp.sempre;

import fig.basic.ListUtils;
import fig.basic.LogInfo;
import fig.basic.Option;
import fig.basic.StopWatchSet;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * Created by bingxu on 19/4/16.
 */
public class SVM {
  public static class Options {
    @Option(gloss = "SVMnormalize")boolean normalize = true;
  }
  public static final double BIAS = 0;
  public static final double REGULARIZED_BIAS = 0;
  public static boolean REGULARIZED_BIAS_set = true;
  public static boolean BIAS_set = true;
  double  lambda;
```

```
    double  eta0;
    double  wDivisor;
    double  wBias;
    double  t;
    public double learnRate = 0;

    public SVM(double lambda,double eta0){
        this.eta0 = eta0;
        this.wDivisor=1;
        this.wBias = 0;
        this.t = 0;
        this.lambda = lambda;
    }

    // Here are utility methods.
    //including loss functions.  ---TODO whether to choose HingeLoss or Logloss. Could switch it in computedExpectedWeights function


    // logloss(a,y) = log(1+exp(-a*y))
    static double Logloss_loss(double a, double y)
    {
        double z = a * y;
        if (z > 18)
            return Math.exp(-z);
        if (z < -18)
            return -z;
        return Math.log(1 + Math.exp(-z));
    }
    // -dloss(a,y)/da
    static double Logloss_dloss(double a, double y)
    {
        double z = a * y;
        if (z > 18)
            return y * Math.exp(-z);
        if (z < -18)
            return y;
        return y / (1 + Math.exp(z));
    }



    // hingeloss(a,y) = max(0, 1-a*y)
    static double HingeLoss_loss(double a, double y)
    {
        double z = a * y;
        if (z > 1)
            return 0;
        return 1 - z;
```

```
  }
  // -dloss(a,y)/da
  static double HingeLoss_dloss(double a, double y)
  {
    double z = a * y;
    if (z > 1)
      return 0;
    return y;
  }


  /// Renormalize the weights
  void renorm(Params params){
    if (wDivisor != 1.0)
    {
      params.scaleWeights(1.0 / wDivisor); // I added this method to Params.java
      wDivisor = 1.0;
    }
  }

  /// Compute the norm of the weights
  double wnorm(Params params)
  {
        double norm = ListUtils.dot(params.getWeight(),params.getWeights()) /
wDivisor / wDivisor;
    #if REGULARIZED_BIAS
    if(REGULARIZED_BIAS_set){
      norm += wBias * wBias
    }
    return norm;
  }



  // Train on just one example and update the weight vector

  /// Perform one iteration of the SGD algorithm with specified gains
        public void computeExpectedCounts(Params params,Example ex,
List<Derivation> derivations,Map<String,Double> counts,FeatureMatcher
updateFeatureMatcher)
  {
    int n = derivations.size();
    if(n==0) return;
    wDivisor = wDivisor / (1-learnRate * lambda);

    if (wDivisor > 1e5) renorm(params);
    double d_overDeriv = 0;
        for(int i=0;i<n;i++){ //All derivations they are computed by the previous
params, so they contribute together for one update
```

```
        Derivation deriv = derivations.get(i);
        double s = deriv.score/wDivisor + wBias;
        //update for regularization term
        double y = ex.targetValue.getCompatibility(deriv.value) == 1 ? 1 : -1;
        // update for loss term
        double d = Logloss_dloss(s, y);
        if (d != 0)
                        deriv.incrementAllFeatureVector(d*learnRate*wDivisor, counts,
updateFeatureMatcher);
        //w.add(x, eta * d * wDivisor);// if d!= w*eta*d*wDivisor is the counts

    }

    if(BIAS_set){
        double etab = learnRate * 0.01;
        if(REGULARIZED_BIAS_set) {
            wBias *= (1 - etab * lambda);
        }
                wBias += etab * d_overDeriv;// TODO : what should I do with this
d_overDeriv( how to combine d?)??? Or I just swtich off the bias now!
    }

  }

   ///Here I maintain the original methods call like in learner. They hopefully do the same
work with linear-SVM

    public void updateCounts(Params params,Example ex, Map<String, Double>
counts,FeatureMatcher updateFeatureMatcher) {
                        computeExpectedCounts(params,ex,  ex.predDerivations,
counts,updateFeatureMatcher);
  }

  public void updateWeights(Params params,Map<String, Double> counts) {
    StopWatchSet.begin("Learner.updateWeights using SVM here!!!!");
    LogInfo.begin_track("Updating weights");
    double sum = 0;
    for (double v : counts.values()) sum += v * v;
    LogInfo.logs("L2 norm: %s", Math.sqrt(sum));
    params.update(counts);
    counts.clear();
    LogInfo.end_track();
    StopWatchSet.end();
  }

///next is about how to compute the learning rate-learnRate?( need a subset sample?
Or just set?
  ///also include a test function here
```

```
  // double testOne(const SVector &x, double y, double *ploss, double *pnerr);


/// Perform one epoch with fixed eta and return cost

      double evaluateLearnRate(Params params,int imin, int imax, List<Example>
examples, FeatureMatcher featureMatcher,double eta)
   {
      HashMap<String, Double> counts = new HashMap<String, Double>();

          // It seems Just cloning the weights map is more convenient
     // assert(imin <= imax);
      for (int i=imin; i<=imax; i++)
         this.updateCounts(params, examples.get(i), counts, featureMatcher);

      //updateWeights(params,counts);

      double loss = 0;
      double cost = 0;
      double error = 0;
      List<Double> resBundle = new ArrayList<>();
      resBundle.add(loss);
      resBundle.add(error);
      for (int i=imin; i<=imax; i++)
         this.testOne(examples.get(i), resBundle);
      loss = loss / (imax - imin + 1);
      cost = loss + 0.5 * lambda * wnorm(params);
      // cout << "Trying eta=" << eta << " yields cost " << cost << endl;
      return cost;
   }


   /// Compute the output for one example.
   double testOne(Example ex, List<Double> resBundle)
   {
      int n = ex.predDerivations.size();
      double ss=0;
      for(int i=0;i<n;i++){
         Derivation deriv = ex.predDerivations.get(i);
         double s = deriv.score/wDivisor+wBias;
                double y = ex.targetValue.getCompatibility(deriv.value) == 1 ? 1 : -1;
resBundle.add(0,Logloss_loss(s,y)); += Logloss_loss(s, y);
         resBundle.add(1,((s * y <= 0) ? 1.0 : 0.0));
         ss+=s;
      }
      resBundle.set(0,resBundle.get(0)/n);
      resBundle.set(1,resBundle.get(1)/n);

      return ss/n;
```

```
  }

    void determineLearnRate(Params params,int imin, int imax, List<Example>
examples,FeatureMatcher featureMatcher)
  {
    LogInfo.begin_track("Computing the learning rate for SVM algorithm here!!! ");


    const double factor = 2.0;
    double loEta = 1;
                double loCost = evaluateLearnRate(params,imin, imax, examples,
featureMatcher,loEta);
    double hiEta = loEta * factor;
                        double hiCost = evaluateLearnRate(params,imin, imax,
examples,featureMatcher, hiEta);
    if (loCost < hiCost)
      while (loCost < hiCost)
      {
        hiEta = loEta;
        hiCost = loCost;
        loEta = hiEta / factor;
          loCost = evaluateLearnRate(params,imin, imax, examples,featureMatcher,
loEta);
      }
    else if (hiCost < loCost)
      while (hiCost < loCost)
      {
        loEta = hiEta;
        loCost = hiCost;
        hiEta = loEta * factor;
                            hiCost = evaluateLearnRate(params,imin, imax, examples,
featureMatcher,hiEta);
      }
      eta0 = loEta;
    LogInfo.logs("# Using eta0= ", eta0);

    LogInfo.end_track();

  }


}
```

10.2 SEMPRE Class specification

A documentation of the core classes are stated here as to provide references for later understanding of the system.

- Formula class (referred as *logical form* in this report for clearer illustration): it is a hierarchical expression for representing logical forms of primitive concrete values, or certain manipulations on logical forms. Formula can also accept Lambda DCS. There are a few basic types for Formula class, e.g., *String, Boolean, List…* Also, Formula supports some functions operated on basic primitive Formulas with keyword "call" specified and Lambda DCS operations, which contains basically unary and binary types. Unary denotes a set of entities; binary denotes a relationship between unaries. In short, Formula is representing the logical forms together with some function allowed, e.g.,

  (boolean False) # primitive logical form

  (fb:en.california) # entity representation

  (fb:people.person.place_of_birth, fb:en.california) # unary representation

  (fb:people.person.place_of_birth) # binary representation

  (call + (number 4) (number 5)) # function operation on Formula

- Derivation class (referred as *derivation* in this report for clearer illustration): one derivation contains one logical form together with a few other relative class member associated with the logical form including the rules it depends on to generate the logical form; feature vector while generating; category and start/end index, etc.,. Very importantly, each derivation contains an evaluation score, which is calculated by the dot product of the feature vector with the weights vector.

- Value class (referred as *denotation* in this report for clearer illustration): it is the returned value of Formula class, in other words, the execution result of the logical form, e.g.,

  (number 4) # number value of 4

  (date 1992-10-26) # date value of 26th Oct, 1992

  (fb:en.hongkong) # Hong Kong

- Semantic Function class (referred as *semantic function* in this report for clearer illustration): the operations that are allowed on logical forms. By applying semantic functions, a set of logical forms can result to one output logical form, e.g.,

(and (fb:people.person.profession fb:en.scientist) (fb:people.person.place_of_birth fb:en.seattle))

this is an example of merge function of intersection, which takes the intersection entities of two logical forms, which in the case, returns a list of people who is a scientist and live in Seattle.

(not (fb:location.location.containedby fb:en.california))

this is an example if reverse function which take the complement of the entities returned from the logical form, which in this case, returns a list of location that are not contained by California.

- Rule class (referred as *rule* in this report for clearer illustration): one specific rule is matching one certain sentence pattern. In each rule definition statement, it starts with a keyword "rule" followed by an output label with input pattern. At the end of the statement, a series of semantic functions together with their arguments should be defined.

(rule $output_label ($token/phrase) (SemanticFn <arg>))

$token/phrase is the type requirement for the incoming pattern. After applying the

semantic function, the returned value is labelled as the output_label.

(rule $expression ($token) (NumberFn))

this is a rule definition stating that for an input token, returns the corresponding number formula and labels that logical form as expression.

To be more specific, with the above rule, every time we type "three", the parser can interpret it as (number 3) with label "expression".

- Grammar class (referred as *grammar* in this report for clearer illustration): a set of rules defined in the system. Grammar file is specified while running the system and the system first reads in all the grammar in order to further parse the input question sentences.

- Parser class (referred as *parser* in this report for clearer illustration): the critical part for parsing the question sentence. In our project, it is initialized as BeamParser, which has the options for the beam size for each sentence. For default, we set it into 500 per example. That means 500 logical forms will be generated as candidates to

execute for the answer. Parser works in the way that parses the example into derivations with feature vector based on different rules specified and knowledge-based information. In code realization, after reading grammar, parser groups the rules into two categories: categorized unary and non-categorized unary (details will be illustrated in section 4.4).

- Feature class (referred as *feature* in this report for clearer illustration): one of the most important components for derivations. There are only two kinds of features adopted in the system: 1. indicator feature; 2. general features (dense feature is never used in the system, so it is not counted; details will be illustrated in section 4.3). While generating derivations, according to various scenarios happened, corresponding features will be attached to derivations. These features serve as standards to describe derivations. Similar features will be adopted during similar derivation generating procedure. Learner will learn to give various weights to different features. Together with the weights, a score can be computed for each derivation, based on which the best derivation will be selected, e.g.,

lexAlign :: colorado springs --- fb:en.colorado_springs_airport

this features tells that according to "lexAlign" feature is added and the content of the feature is to assign phrase "colorado spring" to be the entity of fb:en.colorado_spring_airport.

- Params class (referred as *weight* in this report for clearer illustration): the weight vector computed for the feature vector including the step-size for the Stochastic Gradient Descent algorithm. In the train mode, these weights will be updated after each example by learner, e.g.,

  lexAlign :: colorado springs --- fb:en.colorado_springs_airport -1.4547687140818955

  # the weight of the "lexAlign" feature of colorado spring assigned to fb:en.colorado_spring_airport is -1.4547687140818955

- Learner class (referred as *learner* in this report for clearer illustration): given a set of derivation with different denotations attached different feature space, the learner in the system will adopt Stochastic Gradient Descent algorithm to optimize log-linear model, then update the weight vector based on the examples in dataset.