Natural Language Programming Using Class Sequential Rules

Cohan Sujay Carlos
Aiaioo Labs
Bangalore, India
cohan@aiaioo.com

Outline

- Introduction
- Motivation
- Class Sequential Rules (CSRs)
- Extensions to CSRs
- Corpus
- Experiments
- Related Work

What is **Natural Language Programming**?

"In order to make machines significantly easier to use, it has been proposed (to try) to design machines that we could **instruct in our native tongues**."

- Edsger W. Dijkstra

1978 paper "On the foolishness of Natural Language Programming"

- Natural Language is inadequate **for math** … "Greek math became stuck because it remained a verbal, pictorial activity …"

- A sharp decline in people's mastery of their own languages in the last decades

But things have changed since 1978

Computers are no longer just used for math.

They are (mostly?) used for:

- communication
- entertainment
- **knowledge management** (query languages)
- **controlling hardware** (command languages)

Overview

Developed a system for **Natural Language Programming** using **Class Sequential Rules**

Proposed a set of **programming primitives**

Evaluated the system on the task of identifying **those primitives** and the **entities** they contain

Class Sequential Rule

Sequence of symbols $i_1 - i_n$ that matches text in which the symbols appear in that order

Example, I = < $i_1\ i_2\ i_3$ > where $i_1\ i_2\ i_3$ are unigrams

Matches:

A i_1 B i_2 C D i_3 E

i_1 i_2 R S i_3

i_1 i_2 i_3 M N

Does not match:

A B i_2 i_1 C D i_3 E

A B i_2 C D i_3 E

Class Sequential Rule

Placeholders $x_i - x_n$ among the symbols $i_1 - i_n$ that match text between the symbols on either side

Example, $I = < i_1 \, i_2 \, x_1 \, i_3 >$ *where* $i_1 \, i_2 \, i_3$ *are unigrams*

When I matches A i_1 *B* i_2 *C D* i_3 *E --- x_1 equals 'C D'*

When I matches $i_1 \, i_2$ *R S* i_3 *--- x_1 equals 'R S'*

When I matches $i_1 \, i_2 \, i_3$ *M N --- x_1 equals ''*

Class Sequential Rules

Capabilities of the formalism:

- Identify **types** of sentences
- Extract **entities** from those sentences

Mapping to **Natural Language Programming**:

- **Types** = programming primitives
- **Entities** = variables, literals and expressions

Example

S = increment the value of x by 2 * 3

I_1 = < increment of VARIABLE by EXPRESSION >

I_1 matches S: increment the value of x by 2 * 3

VARIABLE = ‘x’

EXPRESSION = ‘2 * 3’

Example Continued

E = 2 * 3 → (* 2 3)

I_2 = < EXPRESSION * EXPRESSION >

I_2 matches E: 2 * 3

EXPRESSION = ‘2’

EXPRESSION = ‘3’

Example Continued

Mapping to **Natural Language Programming**:

- **Types** = programming primitives
- **Entities** = variables, literals and expressions

increment the value of x by 2 * 3

Class Sequential Rules

(+= x (* 2 3))

Overview

Developed a system for **Natural Language Programming** using **Class Sequential Rules**

Proposed a set of **programming primitives**

Evaluated the system on the task of identifying **those primitives** and the **entities** they contain

Core Primitives

Type	Arity	Example
If	2 or 3	If x is 2 say “Hi”
Unless	2	Exit unless x is equal to 2
While	2	While x is 2, print “Yo!”
Until	2	Till x is 2, keep adding 1 to x
Continuation	1	Also, print x and increment x
Assignment	2	Let x be equal to 3

Other Primitives

Type	Arity	Example
Imperatives	0 to Infinity	Say “Hi”
Questions	2	What is 3 * 2 ?
Y/N Questions	2	Is x equal to 2?

Some Expressions

Type	Arity	Example
Addition	2	x plus y
Subtraction	2	x minus y
Less than	2	x is less than y
Equality	2	x and y are equal
Disjunction	2	x or y
Conjunction	2	x and y

The only data types supported right now are numbers & strings

Intermediate Representation

Natural Language Form	Intermediate Representation Form
If x equals y print x	(if (= x y) (print x))
Assign y to x	(= x y)
If x > y, let x be equal to y	(if (> x y) (= x y))
Add x to y	(+= x y)
while x is less than y, print x and increment x.	(while (< x y) (& (print x) (+= x 1)))
Print x and then print y.	(& (print x) (print y))

Overloading of & and = in the intermediate representation

But CSRs aren’t powerful enough

Three Difficult Sentences *
Also { x = 2 } .
Also , { x = 2 } .
Also { if x = 3 , ++x } .

Class Sequential Rules for Continuation
Also , EXPRESSION .
Also EXPRESSION .

* The flower braces indicate entity spans

Extending CSRs

CSR

- $I = < i_1\, i_2\, i_3 >$ *where* $i_1\, i_2\, i_3$ *are unigrams*

CSR-EX

- $I = < i_1\, i_2\, i_3 >$ *where* $i_1\, i_2\, i_3$ *are n-grams*

CSR-EXs are powerful enough

Class Sequential Rules for Continuation
Also NONE , EXPRESSION .
Also EXPRESSION .

Three Difficult Sentences *	EXPRESSION
Also { x = 2 } .	x = 2
Also , { x = 2 } .	x = 2
Also { if x = 3 , ++x } .	if x = 3 , ++x

* The flower braces indicate entity spans

Learning Algorithms for CSRs

- Bing Liu[*] described an algorithm for learning Class Sequential Rules.

  [*] *Opinion Feature Extraction Using Class Sequential Rules* - Hu and Liu (2006)

- Sequential Pattern Mining algorithms[**] can be used.

  [**] Research Report – Mining Sequential Patterns – Agrawal and Srikanth

Overview

Developed a system for **Natural Language Programming** using **Class Sequential Rules**

Proposed a set of **programming primitives**

Evaluated the system on the task of identifying **those primitives** and the **entities** they contain

Evaluation Corpus

3,000 sentences (3 sets of 1000 each)

Online questionnaire:
1. How would you say "x = 2" in English?
2. How would you say "x != 2" in English?
3. How would you say "x < 2" ?

Download:
http://www.aiaioo.com/corpora/vaklipi2011/

Systems Evaluated

- CSR-BL – CSRs using unigrams
- CSR-EX – CSRs using n-grams
- CSR-Man – Manually created n-gram CSR rules

Results

Identifying the **type** of the programming primitive

Setting	Precision	Recall	F1
CSR-Man	89.2 +- 3.7	64.8 +- 6.2	73.0 +- 4
CSR-BL	85.7 +- 4.5	65.3 +- 5.9	73.1 +- 4
CSR-EX	88.4 +- 3.4	66.5 +- 5.6	74.8 +- 3

Identifying entity spans

Setting	PSCS*
CSR-Man	52.4 +- 9.1
CSR-BL	50.2 +- 8.4
CSR-EX	49.7 +- 8.6

*Percentage of Sentences with Correct Spans

Performance

Counts of sentences in the corpus and performance

Setting	Count	Precision	Recall	F1
equality	298	79.0 +- 6	66.5 +- 19	71.9 +- 10
inequality	165	90.6 +- 14	78.6 +- 6	84.3 +- 9
less than	151	66.8 +- 10	88.4 +- 7	76.8 +- 8
if	118	84.2 +- 5	96.0 +- 8	89.8 +- 4
unless	15	100 +- 0	60.7 +- 15	77.6 +- 9
while	61	92.1 +- 2	88.0 +- 12	89.8 +- 11
until	86	98.8 +- 2	85.8 +- 15	91.9 +- 8
continue	48	78.3 +- 23	22.1 +- 11	40.0 +- 5

Prior Work

- “NLC” – Ballard and Biermann (1979)
- “Metafor” – Lieberman and Liu (2005)
- “Pegasus” – Knoell and Mezini (2006)
- Skeletons – Mihalcea et al (2006)
- Pacman – F. Pane and Brad A. Myers (2000)

Future Work

- Other algorithms
- Other languages
- Other data types
- Other domains of application
- Other corpora
- Translation models

End

Natural Language Programming Using Class Sequential Rules

Cohan Sujay Carlos
Aiaioo Labs
Bangalore, India
cohan@aiaioo.com

Weaknesses

- Not an evaluation of end-to-end performance
- The language of the responses elicited for the corpus is possibly biased or unduly restricted by the questions
- The error margins are high
- Performance measure is not independent of number of types of programming primitives recognized