SFU NatLangLab

Bootstrapping via Graph Propagation

Max Whitney Anoop Sarkar
Simon Fraser University
Natural Language Laboratory
http://natlang.cs.sfu.ca

Bootstrapping

- Semi-supervised (vs supervised)
- Single domain (vs domain adaptation)
- Small amount of seed data/rules (vs domain adaptation)

Bootstrapping

- Semi-supervised (vs supervised)
- Single domain (vs domain adaptation)
- Small amount of seed data/rules (vs domain adaptation)

Assumption:

- No transductive learning

General approaches to semi-supervised learning

- Clustering
  concept must be identifiable
- Maximum likelihood
  problems with local optima

generative

---

discriminative

- Co-training
  learn from agreement between models; *need independent views*
- **Self-training**
  learn from agreement between features

The Yarowsky algorithm

- Yarowsky algorithm: self-training algorithm by David Yarowsky (1995)
- Works well empirically
- Little theoretical analysis

The Yarowsky algorithm

- Yarowsky algorithm: self-training algorithm by David Yarowsky (1995)
- Works well empirically
- Little theoretical analysis

- Co-training by Avrim Blum and Tom Mitchell (1998):
  The paper has been cited over 1000 times, and received the 10 years Best Paper Award at the 25th International Conference on Machine Learning (2008)

The Yarowsky algorithm

- Yarowsky algorithm: self-training algorithm by David Yarowsky (1995)
- Works well empirically
- Little theoretical analysis

- Co-training by Avrim Blum and Tom Mitchell (1998):
  The paper has been cited over 1000 times, and received the 10 years Best Paper Award at the 25th International Conference on Machine Learning (2008)
- Collins and Singer (1999) provide Co-Boost: co-training with a per-iteration objective function and good accuracy

The Yarowsky algorithm

- Yarowsky algorithm: self-training algorithm by David Yarowsky (1995)
- Works well empirically
- Little theoretical analysis

- Co-training by Avrim Blum and Tom Mitchell (1998):
  The paper has been cited over 1000 times, and received the 10 years Best Paper Award at the 25th International Conference on Machine Learning (2008)
- Collins and Singer (1999) provide Co-Boost: co-training with a per-iteration objective function and good accuracy

- Can we do the same for the Yarowsky algorithm?

Example task: word sense disambiguation

Data from Canadian Hansards (Eisner and Karakos, 2005):

- 2 labels (senses)
- features are adjacent and context (nearby) words
- 2 seed rules

Example task: word sense disambiguation

303 unlabelled training examples:

- Full time should be served for each **sentence** .
- The Liberals inserted a **sentence** of 14 words which reads :
- They get a concurrent **sentence** with no additional time added to their sentence .
- The words tax relief appeared in every second **sentence** in the federal government's throne speech .

⋮

Example task: word sense disambiguation

303 unlabelled training examples:

- Full time should be served for each **sentence** .
- The Liberals inserted a **sentence** of 14 words which reads :
- They get a concurrent **sentence** with no additional time added to their sentence .
- The words tax relief appeared in every second **sentence** in the federal government's throne speech .

⋮

2 seed rules:

context: served	sense 1
context: reads	sense 2

Example task: word sense disambiguation

303 unlabelled training examples:

- Full time should be served for each **sentence** .
- The Liberals inserted a **sentence** of 14 words which reads :
- They get a concurrent **sentence** with no additional time added to their sentence .
- The words tax relief appeared in every second **sentence** in the federal government's throne speech .

⋮

2 seed rules:

context: served	sense 1
context: reads	sense 2

Example task: word sense disambiguation

303 unlabelled training examples:

- Full time should be served for each **sentence** .
- The Liberals inserted a **sentence** of 14 words which reads :
- They get a concurrent **sentence** with no additional time added to their sentence .
- The words tax relief appeared in every second **sentence** in the federal government's throne speech .

$\vdots$

2 seed rules:

context: served	sense 1
context: reads	sense 2

Example task: word sense disambiguation

303 unlabelled training examples:

- Full time should be served for each **sentence** .
- The Liberals inserted a **sentence** of 14 words which reads :
- They get a concurrent **sentence** with no additional time added to their sentence .
- The words tax relief appeared in every second **sentence** in the federal government's throne speech .

⋮

2 seed rules: $\rightarrow$ 76.99% accuracy on unseen test set

context: served	sense 1
context: reads	sense 2

Example task: word sense disambiguation

303 unlabelled training examples:

- Full time should be served for each **sentence** .
- The Liberals inserted a **sentence** of 14 words which reads :
- They get a concurrent **sentence** with no additional time added to their sentence .
- The words tax relief appeared in every second **sentence** in the federal government's throne speech .

⋮

2 seed rules:

context: served	sense 1
context: reads	sense 2

→ 76.99% accuracy on unseen test set

non-seeded accuracy (Daume, 2011)

Example task: named entity classification

Data from NYT (Collins and Singer, 1999):

- 3 labels (person, location, organization)
- spelling features from words in phrase,
  context features from parse tree
- 7 seed rules

Example task: named entity classification

89305 unlabelled training examples:

Example task: named entity classification

89305 unlabelled training examples:

- Union Bank would automatically give it a foothold in this market in **California** .
- It is an ironic agreement , given Mr. Jobs' historical disdain for **IBM** .
- It is an ironic agreement , given **Mr. Jobs**' historical disdain for IBM .

$\vdots$

Example task: named entity classification

89305 unlabelled training examples:

- Union Bank would automatically give it a foothold in this market in **California** .
- It is an ironic agreement , given Mr. Jobs' historical disdain for **IBM** .
- It is an ironic agreement , given **Mr. Jobs**' historical disdain for IBM .

⋮

7 seed rules:

spelling: New-York	location
spelling: California	location
spelling: U.S.	location
spelling: Microsoft	organization
spelling: I.B.M.	organization
spelling: *Incorporated*	organization
spelling: *Mr.*	person

Example task: named entity classification

89305 unlabelled training examples:

- Union Bank would automatically give it a foothold in this market in **California** .
- It is an ironic agreement , given Mr. Jobs' historical disdain for **IBM** .
- It is an ironic agreement , given **Mr. Jobs**' historical disdain for IBM .

⋮

7 seed rules:

spelling: New-York	location
spelling: California	location
spelling: U.S.	location
spelling: Microsoft	organization
spelling: I.B.M.	organization
spelling: *Incorporated*	organization
spelling: *Mr.*	person

Example task: named entity classification

89305 unlabelled training examples:

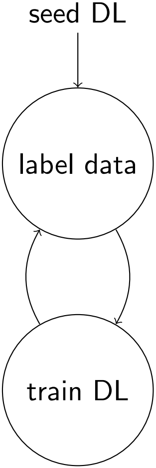
- Union Bank would automatically give it a foothold in this market in **California** .
- It is an ironic agreement , given Mr. Jobs' historical disdain for **IBM** .
- It is an ironic agreement , given **Mr. Jobs**' historical disdain for IBM .

⋮

7 seed rules:

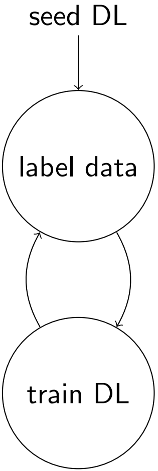
spelling: New-York	location	
spelling: California	location	
spelling: U.S.	location	
spelling: Microsoft	organization	$\rightarrow$ 89.97% test accuracy
spelling: I.B.M.	organization	
spelling: *Incorporated*	organization	
spelling: *Mr.*	person	

Yarowsky algorithm (Yarowsky, 1995; Collins and Singer, 1999)

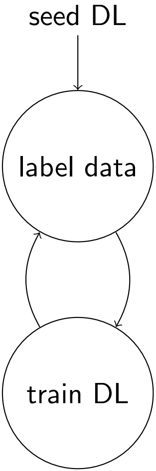


Yarowsky algorithm (Yarowsky, 1995; Collins and Singer, 1999)

1.0	context: served	sense 1
1.0	context: reads	sense 2



Yarowsky algorithm (Yarowsky, 1995; Collins and Singer, 1999)



1.0	context: served	sense 1
1.0	context: reads	sense 2

Full time should be served for each **sentence** .

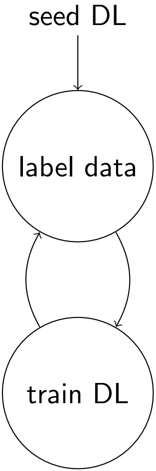
The Liberals inserted a **sentence** of 14 words which reads :

The **sentence** for such an offence would be a term of imprisonment for one year .

Mr. Speaker , I have a question based on the very last **sentence** of the hon. member .

⋮

Yarowsky algorithm (Yarowsky, 1995; Collins and Singer, 1999)



1.0	context: served	sense 1
1.0	context: reads	sense 2

Full time should be served for each **sentence** .

The Liberals inserted a **sentence** of 14 words which reads :

The **sentence** for such an offence would be a term of imprisonment for one year .

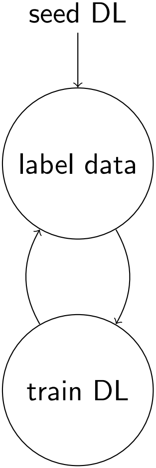
Mr. Speaker , I have a question based on the very last **sentence** of the hon. member .

⋮

1.0	context: served	sense 1
1.0	context: reads	sense 2
.976	context: serv*	sense 1
.976	context: read*	sense 2
.969	next word: reads	sense 2
.969	next word: read*	sense 2
.955	previous word: his	sense 1
.955	previous word: hi*	sense 1
.955	context: inmate	sense 1
.917	previous word: their	sense 1
.917	previous word: relevant	sense 2
.917	previous word: next	sense 2

⋮

Yarowsky algorithm (Yarowsky, 1995; Collins and Singer, 1999)



1.0	context: served	sense 1
1.0	context: reads	sense 2

Full time should be $\boxed{\text{served}}$ for each **sentence** .

The Liberals inserted a **sentence** of 14 words which $\boxed{\text{reads}}$:

The **sentence** for such an offence would be a term of imprisonment for one year .

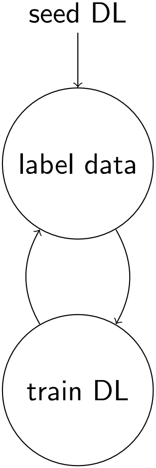
Mr. Speaker , I have a question based on the very last **sentence** of the hon. member .

⋮

1.0	context: served	sense 1
1.0	context: reads	sense 2
.976	context: serv*	sense 1
.976	context: read*	sense 2
.969	next word: reads	sense 2
.969	next word: read*	sense 2
.955	previous word: his	sense 1
.955	previous word: hi*	sense 1
.955	context: inmate	sense 1

threshold

Yarowsky algorithm (Yarowsky, 1995; Collins and Singer, 1999)



1.0	context: served	sense 1
1.0	context: reads	sense 2

Full time should be $\boxed{\text{served}}$ for each **sentence** .

The Liberals inserted a **sentence** of 14 words which $\boxed{\text{reads}}$:

The **sentence** for such an offence would be a term of imprisonment for one year .

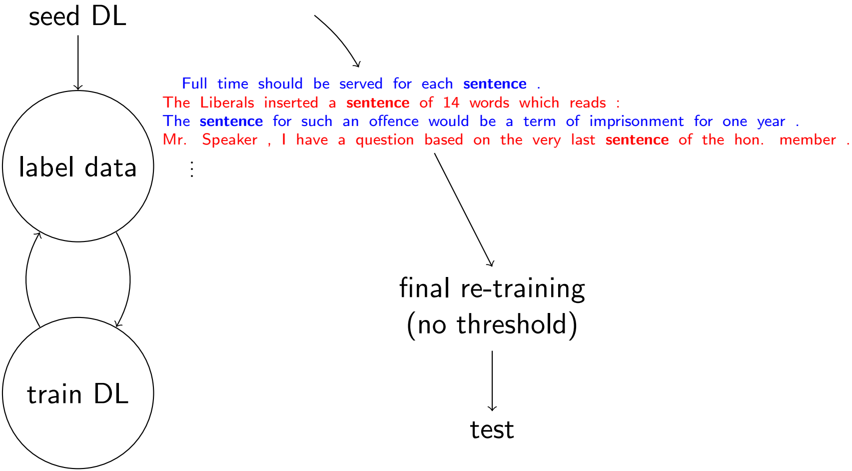
Mr. Speaker , I have a question based on the very last **sentence** of the hon. member .

⋮

1.0	context: served	sense 1
1.0	context: reads	sense 2
.976	context: serv*	sense 1
.976	context: read*	sense 2
.969	next word: reads	sense 2
.969	next word: read*	sense 2
.955	previous word: his	sense 1
.955	previous word: hi*	sense 1
.955	context: inmate	sense 1

threshold

Yarowsky algorithm (Yarowsky, 1995; Collins and Singer, 1999)



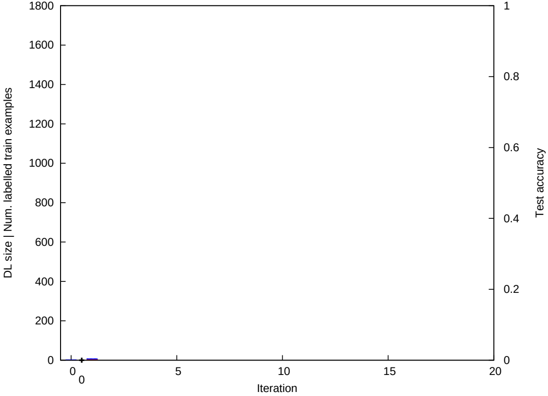
Example decision list for the named entity task

Rank	Score	Feature	Label
1	0.999900	New-York	loc.
2	0.999900	California	loc.
3	0.999900	U.S.	loc.
4	0.999900	Microsoft	org.
5	0.999900	I.B.M.	org.
6	0.999900	Incorporated	org.
7	0.999900	Mr.	per.
8	0.999976	U.S.	loc.
9	0.999957	New-York-Stock-Exchange	loc.
10	0.999952	California	loc.
11	0.999947	New-York	loc.
12	0.999946	*court-in*	loc.
13	0.975154	*Company-of*	loc.

$\vdots$

Context features are indicated by *italics*; all others are spelling features. Seed rules are indicated by **bold** ranks.

Yarowsky algorithm (Yarowsky, 1995; Collins and Singer, 1999)



1 rule

1.0 context: served

1 rule

1.0 context: reads

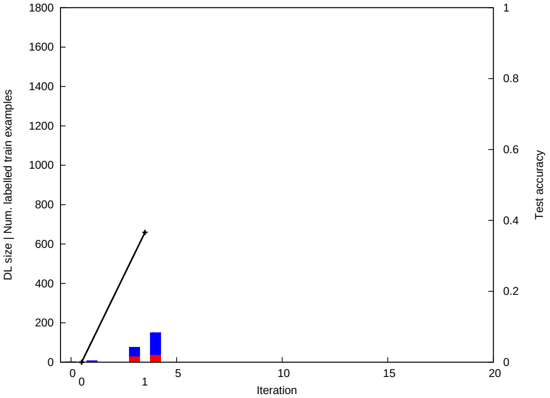
train

4

4

Iteration 0

Yarowsky algorithm (Yarowsky, 1995; Collins and Singer, 1999)



46 rules

1.0 context: served
.976 context: serv*
.976 context: served
.955 context: inmat*
.955 context: releas*

⋮

31 rules

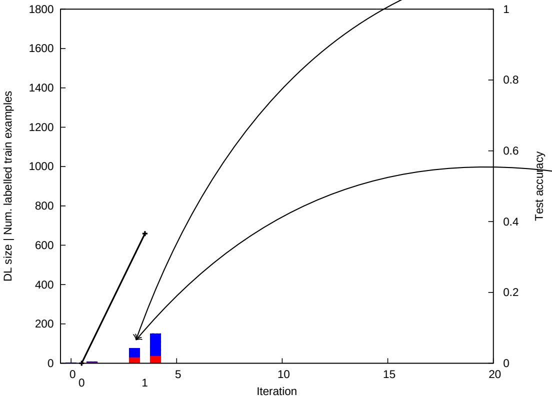
1.0 context: reads
.976 context: read*
.976 context: reads
.969 next: read*
.969 next: reads

⋮

train
114
37

Iteration 1

Yarowsky algorithm (Yarowsky, 1995; Collins and Singer, 1999)



46 rules

1.0	context: served
.976	context: serv*
.976	context: served
.955	context: inmat*
.955	context: releas*

⋮

31 rules

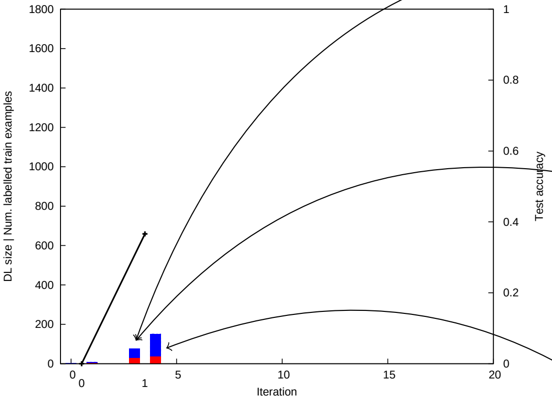
1.0	context: reads
.976	context: read*
.976	context: reads
.969	next: read*
.969	next: reads

⋮

train
114
37

Iteration 1

Yarowsky algorithm (Yarowsky, 1995; Collins and Singer, 1999)



46 rules

1.0	context: served
.976	context: serv*
.976	context: served
.955	context: inmat*
.955	context: releas*

⋮

31 rules

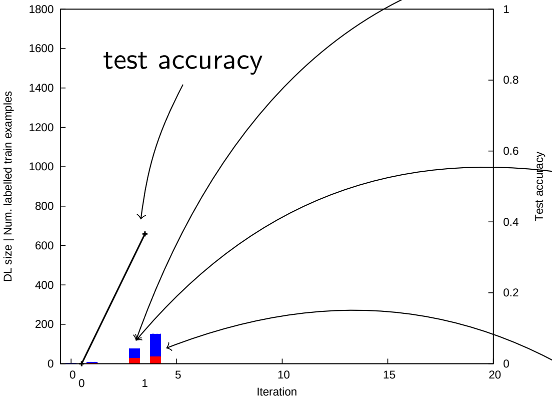
1.0	context: reads
.976	context: read*
.976	context: reads
.969	next: read*
.969	next: reads

⋮

train
114
37

Iteration 1

Yarowsky algorithm (Yarowsky, 1995; Collins and Singer, 1999)



<u>46 rules</u>

1.0	context: served
.976	context: serv*
.976	context: served
.955	context: inmat*
.955	context: releas*

⋮

<u>31 rules</u>

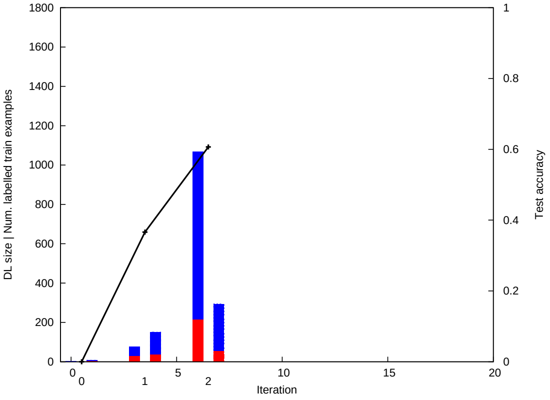
1.0	context: reads
.976	context: read*
.976	context: reads
.969	next: read*
.969	next: reads

⋮

<u>train</u>
114
37

Iteration 1

Yarowsky algorithm (Yarowsky, 1995; Collins and Singer, 1999)



854 rules

1.0 context: served
.998 next: .*
.998 next: .
.995 context: serv*
.995 context: prison*

⋮

214 rules

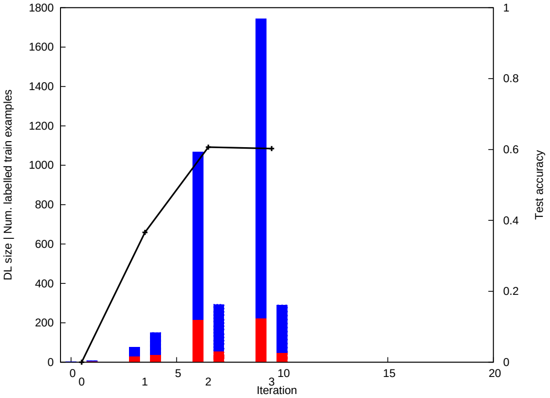
1.0 context: reads
.991 context: read*
.984 context: read
.976 context: reads
.969 context: 11*

⋮

train
238
56

Iteration 2

Yarowsky algorithm (Yarowsky, 1995; Collins and Singer, 1999)



1520 rules

1.0 context: served
.998 next: .*
.998 next: .
.960 context: life*
.960 context: life

⋮

223 rules

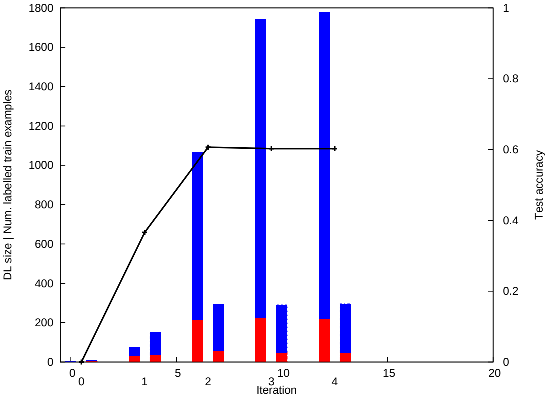
1.0 context: reads
.991 context: read*
.984 context: read
.984 next: :*
.984 next: :

⋮

train
242
49

Iteration 3

Yarowsky algorithm (Yarowsky, 1995; Collins and Singer, 1999)



1557 rules

1.0 context: served
.998 next: .*
.998 next: .
.996 context: life*
.996 context: life

⋮

221 rules

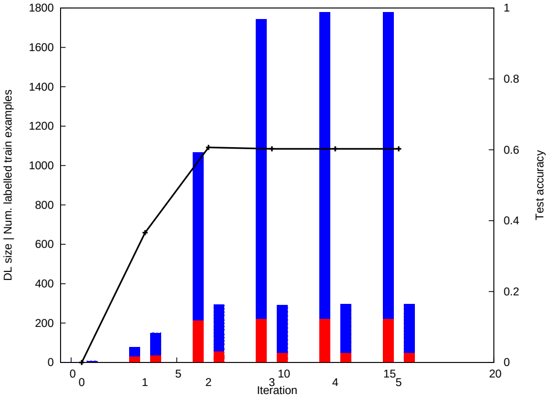
1.0 context: reads
.991 context: read*
.984 context: read
.984 next: :*
.984 next: :

⋮

train
247
49

Iteration 4

Yarowsky algorithm (Yarowsky, 1995; Collins and Singer, 1999)



1557 rules

1.0	context: served
.998	next: .*
.998	next: .
.996	context: life*
.996	context: life

⋮

221 rules

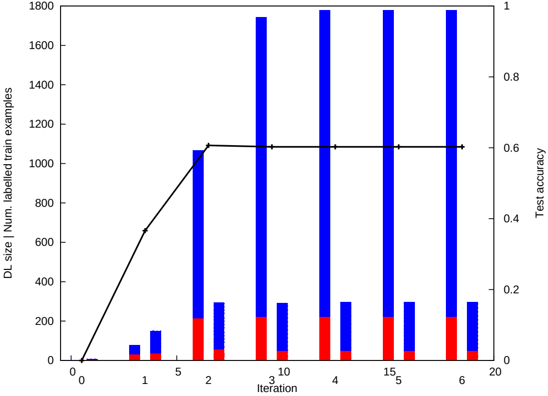
1.0	context: reads
.991	context: read*
.984	context: read
.984	next: :*
.984	next: :

⋮

train
247
49

Iteration 5

Yarowsky algorithm (Yarowsky, 1995; Collins and Singer, 1999)



1557 rules

1.0	context: served
.998	next: .*
.998	next: .
.996	context: life*
.996	context: life

⋮

221 rules

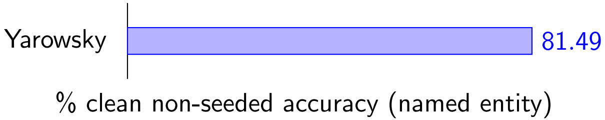
1.0	context: reads
.991	context: read*
.984	context: read
.984	next: :*
.984	next: :

⋮

train
247
49

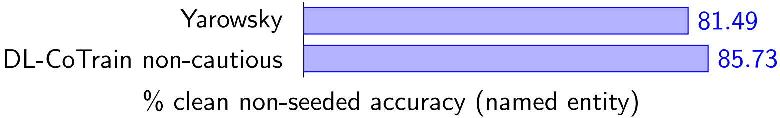
Iteration 6

Performance



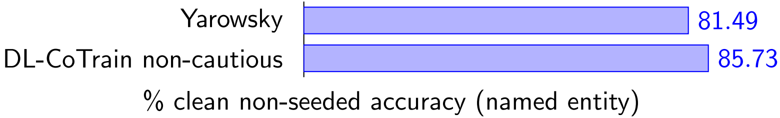
Vs. co-training

DL-CoTrain from (Collins and Singer, 1999):



Vs. co-training

DL-CoTrain from (Collins and Singer, 1999):



Co-training needs two views, eg:

- adjacent words { next word: a, next word: about, next word: according, . . . }
- context words { context: abolition, context: abundantly, context: accepting, . . . }

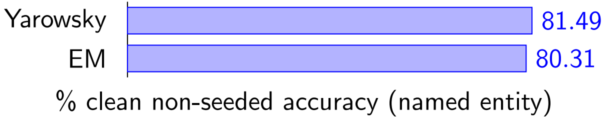
Vs. EM

EM algorithm from (Collins and Singer, 1999):

% clean non-seeded accuracy (named entity)

Vs. EM

EM algorithm from (Collins and Singer, 1999):



With Yarowsky we can exploit type-level information in the DL

Vs. EM

EM

Expected counts on data:

x_1

x_2

x_3

x_4

x_5

$\vdots$

Probabilities on features:

f_1

f_2

f_3

f_4

f_5

$\vdots$

Vs. EM

EM

Expected counts on data:

x_1
x_2
x_3
x_4
x_5
$\vdots$

Probabilities on features:

f_1
f_2
f_3
f_4
f_5
$\vdots$

Yarowsky

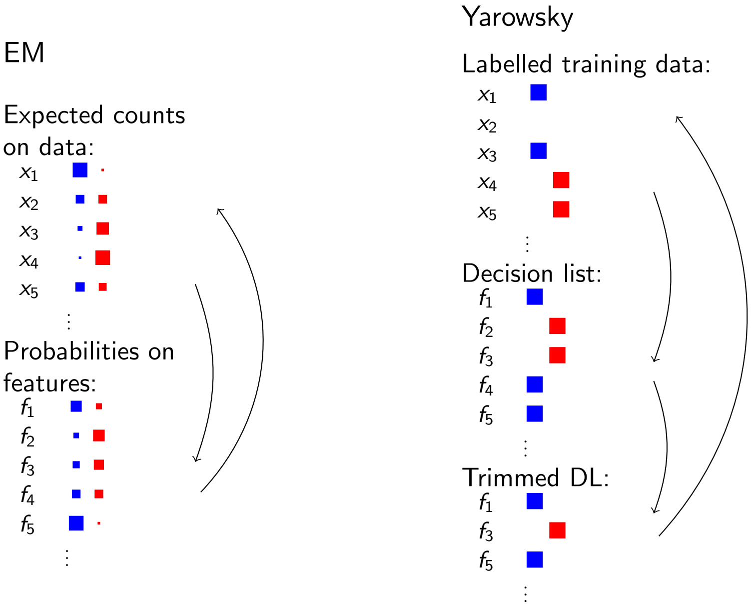
Labelled training data:

x_1
x_2
x_3
x_4
x_5
$\vdots$

Decision list:

f_1
f_2
f_3
f_4
f_5
$\vdots$

Vs. EM



Yarowsky
EM
Expected counts on data:
x_1
x_2
x_3
x_4
x_5
⋮
Probabilities on features:
f_1
f_2
f_3
f_4
f_5
⋮
Labelled training data:
x_1
x_2
x_3
x_4
x_5
⋮
Decision list:
f_1
f_2
f_3
f_4
f_5
⋮
Trimmed DL:
f_1
f_3
f_5
⋮

Cautiousness

Can we improve decision list trimming?

Cautiousness

Can we improve decision list trimming?

- (Collins and Singer, 1999) cautiousness:
  take top n rules for each label
  $n = 5, 10, 15, \ldots$ by iteration

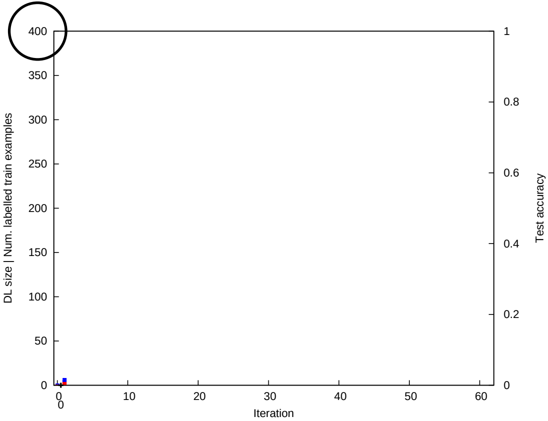
Cautiousness

Can we improve decision list trimming?

- (Collins and Singer, 1999) cautiousness:
  take top n rules for each label
  $n = 5, 10, 15, \ldots$ by iteration

- Yarowsky-cautious
- DL-CoTrain cautious

Yarowsky-cautious algorithm (Collins and Singer, 1999)



1 rule

1.0 context: served

1 rule

1.0 context: reads

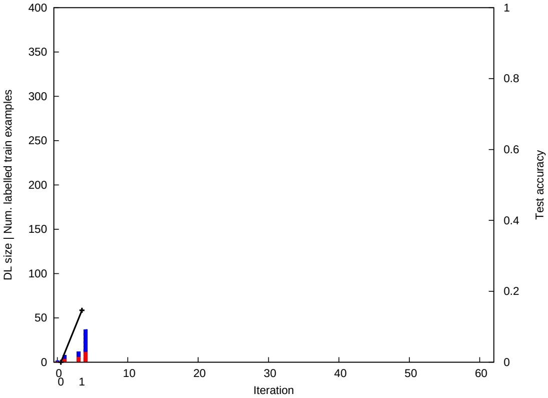
train

4

4

Iteration 0

Yarowsky-cautious algorithm (Collins and Singer, 1999)



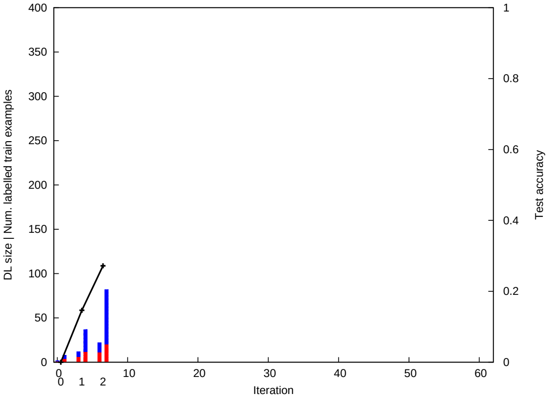
6 rules
1.0 context: served
.976 context: serv*
.976 context: served
.955 context: inmat*
.955 context: releas*
⋮

6 rules
1.0 context: reads
.976 context: read*
.976 context: reads
.969 next: read*
.969 next: reads
⋮

train
25
12

Iteration 1

Yarowsky-cautious algorithm (Collins and Singer, 1999)



11 rules

1.0 context: served
.995 context: serv*
.989 context: serve
.986 context: serving
.984 context: life*

⋮

11 rules

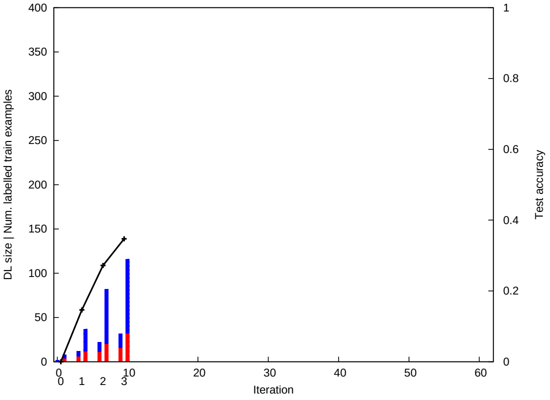
1.0 context: reads
.991 context: read*
.984 context: read
.976 context: reads
.969 next: from*

⋮

train
62
20

Iteration 2

Yarowsky-cautious algorithm (Collins and Singer, 1999)



16 rules

1.0 context: served
.996 context: life*
.996 context: life
.995 context: serv*
.995 context: prison*

⋮

16 rules

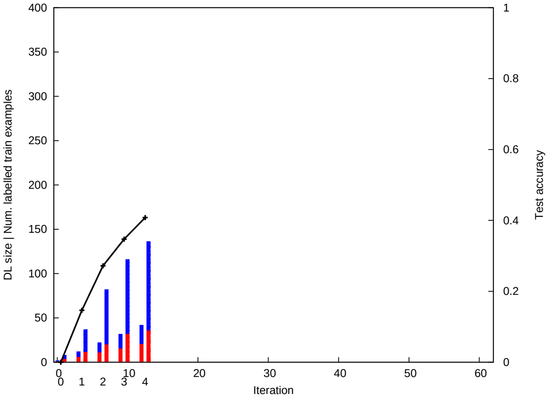
1.0 context: reads
.991 context: read*
.991 next: from*
.991 next: from
.984 context: read

⋮

train
84
32

Iteration 3

Yarowsky-cautious algorithm (Collins and Singer, 1999)



21 rules

1.0	context: served
.996	context: commut*
.996	context: life*
.996	context: life
.995	context: serv*

⋮

21 rules

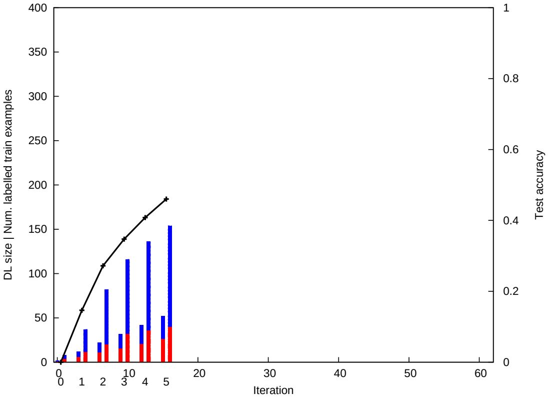
1.0	context: reads
.991	context: read*
.991	next: from*
.991	next: from
.989	context: quot*

⋮

train
100
36

Iteration 4

Yarowsky-cautious algorithm (Collins and Singer, 1999)



26 rules

1.0 context: served
.996 context: commut*
.996 context: life*
.996 context: life
.995 context: serv*

⋮

26 rules

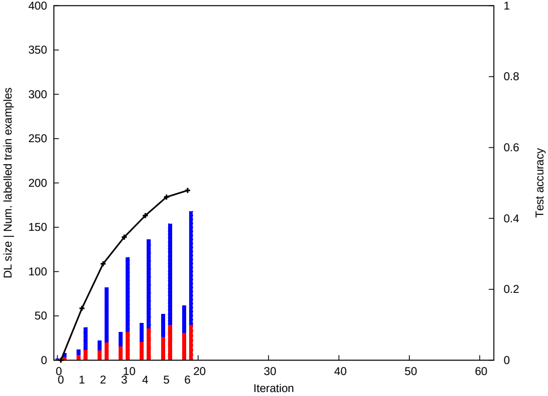
1.0 context: reads
.991 context: read*
.991 next: from*
.991 next: from
.989 context: quot*

⋮

train
114
40

Iteration 5

Yarowsky-cautious algorithm (Collins and Singer, 1999)



31 rules

1.0 context: served
.996 context: commut*
.996 context: life*
.996 context: life
.995 context: serv*
⋮

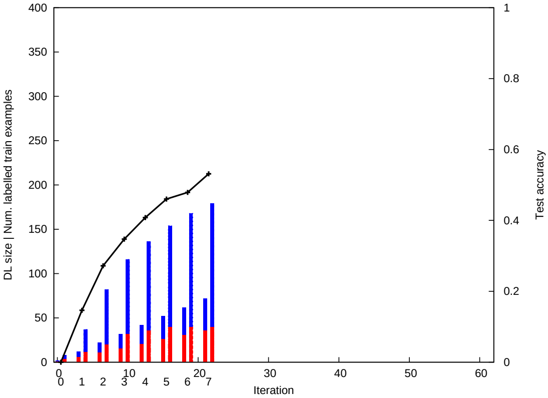
31 rules

1.0 context: reads
.991 context: read*
.991 next: from*
.991 next: from
.989 context: quot*
⋮

train
128
40

Iteration 6

Yarowsky-cautious algorithm (Collins and Singer, 1999)



36 rules

1.0 context: served
.965 context: year*
.996 context: commut*
.996 context: life*
.996 context: life
⋮

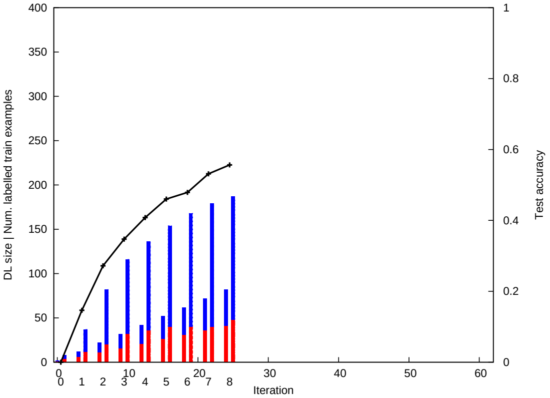
36 rules

1.0 context: reads
.991 context: read*
.991 next: from*
.991 next: from
.989 context: quot*
⋮

train
139
40

Iteration 7

Yarowsky-cautious algorithm (Collins and Singer, 1999)



<u>41 rules</u>

1.0 context: served
.969 context: year*
.996 context: commut*
.996 context: life*
.996 context: life
⋮

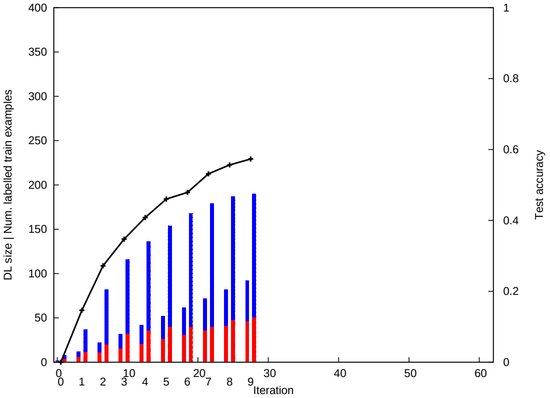
<u>41 rules</u>

1.0 context: reads
.991 context: read*
.991 next: from*
.991 next: from
.989 context: quot*
⋮

<u>train</u>
139
48

Iteration 8

Yarowsky-cautious algorithm (Collins and Singer, 1999)



<u>46 rules</u>

1.0	context: served
.969	context: year*
.996	context: commut*
.996	context: life*
.996	context: life

⋮

<u>46 rules</u>

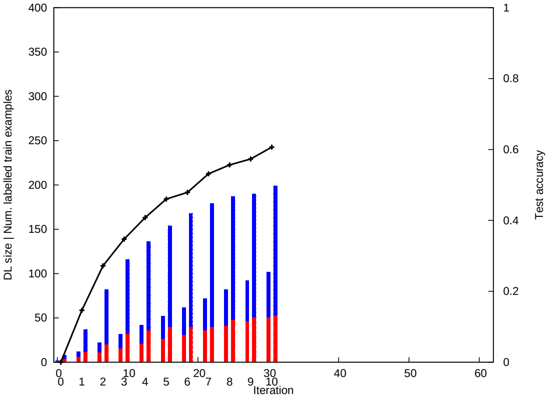
1.0	context: reads
.991	context: read*
.991	next: from*
.991	next: from
.989	context: quot*

⋮

<u>train</u>
139
51

Iteration 9

Yarowsky-cautious algorithm (Collins and Singer, 1999)



51 rules

1.0 context: served
.969 context: year*
.996 context: commut*
.996 context: life*
.996 context: life

⋮

51 rules

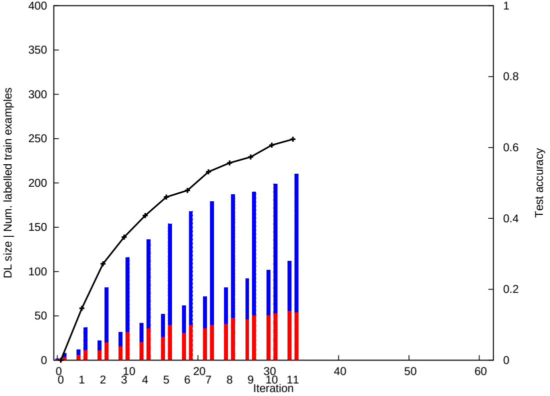
1.0 context: reads
.991 context: read*
.991 next: from*
.991 next: from
.989 context: quot*

⋮

train
146
53

Iteration 10

Yarowsky-cautious algorithm (Collins and Singer, 1999)



56 rules

1.0 context: served
.969 context: year*
.996 context: commut*
.996 context: life*
.996 context: life
⋮

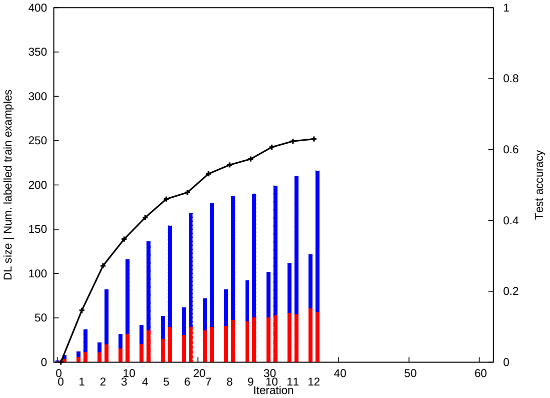
56 rules

1.0 context: reads
.991 context: read*
.991 next: from*
.991 next: from
.989 context: quot*
⋮

train
156
54

Iteration 11

Yarowsky-cautious algorithm (Collins and Singer, 1999)



Iteration 12

<u>61 rules</u>

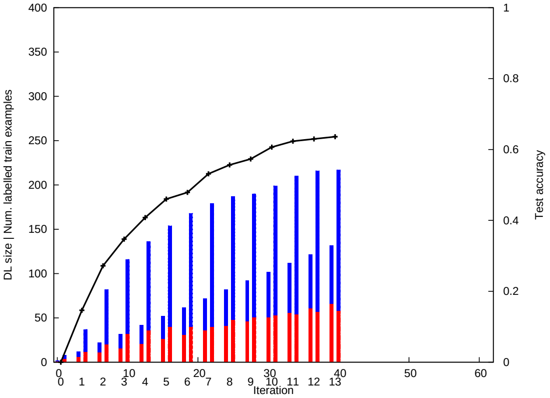
1.0 context: served
.969 context: year*
.996 context: commut*
.996 context: life*
.996 context: life
⋮

<u>61 rules</u>

1.0 context: reads
.991 context: read*
.991 next: from*
.991 next: from
.989 context: quot*
⋮

train
159
57

Yarowsky-cautious algorithm (Collins and Singer, 1999)



66 rules

1.0	context: served
.969	context: year*
.996	context: commut*
.996	context: life*
.996	context: life

⋮

66 rules

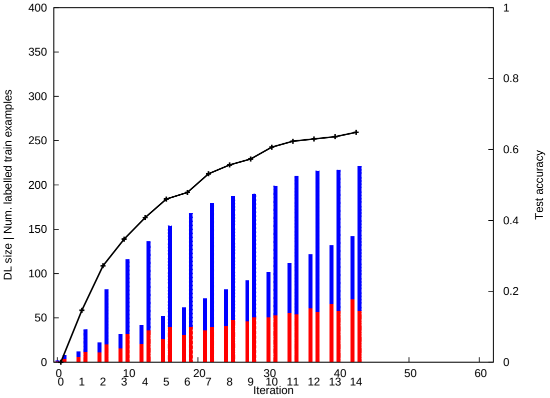
1.0	context: reads
.991	context: read*
.991	next: from*
.991	next: from
.989	context: quot*

⋮

train
159
58

Iteration 13

Yarowsky-cautious algorithm (Collins and Singer, 1999)



71 rules

1.0	context: served
.969	context: year*
.996	context: commut*
.996	context: life*
.996	context: life

⋮

71 rules

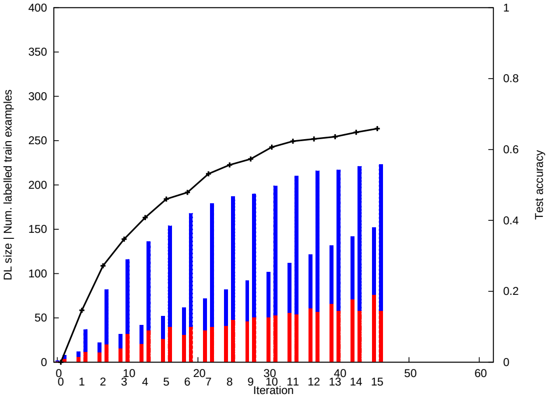
1.0	context: reads
.991	context: read*
.991	next: from*
.991	next: from
.989	context: quot*

⋮

train
163
58

Iteration 14

Yarowsky-cautious algorithm (Collins and Singer, 1999)



76 rules

1.0 context: served
.969 context: year*
.996 context: commut*
.996 context: life*
.996 context: life

⋮

76 rules

1.0 context: reads
.991 context: read*
.991 next: from*
.991 next: from
.989 context: quot*

⋮

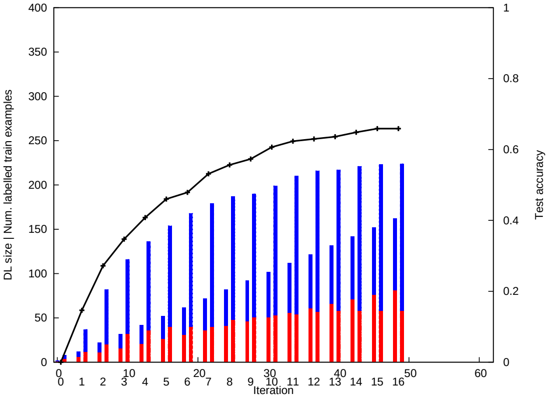
train

165

58

Iteration 15

Yarowsky-cautious algorithm (Collins and Singer, 1999)



81 rules

1.0	context: served
.969	context: year*
.996	context: commut*
.996	context: life*
.996	context: life

⋮

81 rules

1.0	context: reads
.991	context: read*
.991	next: from*
.991	next: from
.989	context: quot*

⋮

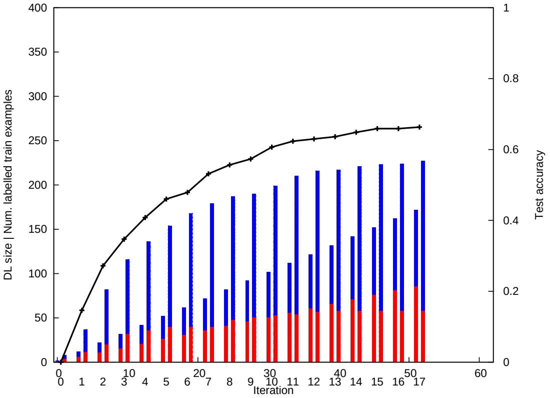
Iteration 16

train

166

58

Yarowsky-cautious algorithm (Collins and Singer, 1999)



86 rules

1.0	context: served
.969	context: year*
.996	context: commut*
.996	context: life*
.996	context: life

⋮

86 rules

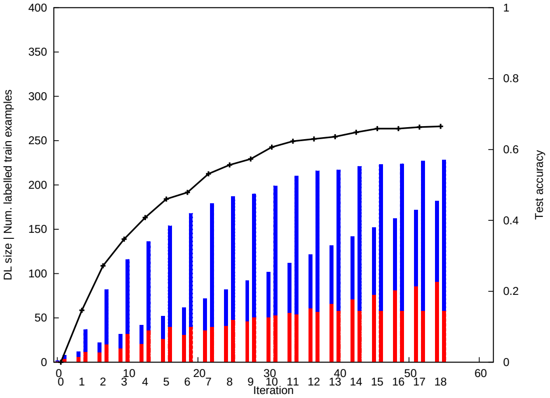
1.0	context: reads
.991	context: read*
.991	next: from*
.991	next: from
.989	context: quot*

⋮

train
169
58

Iteration 17

Yarowsky-cautious algorithm (Collins and Singer, 1999)



91 rules

1.0 context: served
.969 context: year*
.996 context: commut*
.996 context: life*
.996 context: life
⋮

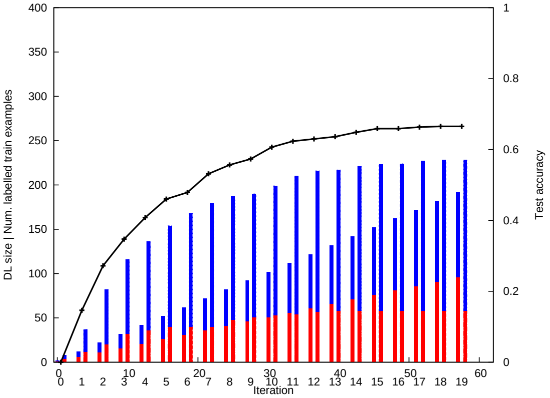
91 rules

1.0 context: reads
.991 context: read*
.991 next: from*
.991 next: from
.989 context: quot*
⋮

train
170
58

Iteration 18

Yarowsky-cautious algorithm (Collins and Singer, 1999)



96 rules

1.0 context: served
.969 context: year*
.996 context: commut*
.996 context: life*
.996 context: life
⋮

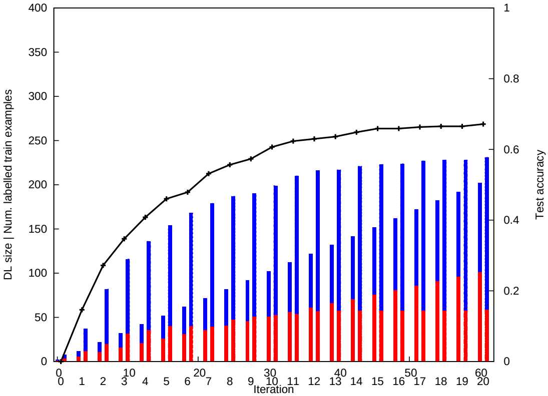
96 rules

1.0 context: reads
.991 context: read*
.991 next: from*
.991 next: from
.989 context: quot*
⋮

train
170
58

Iteration 19

Yarowsky-cautious algorithm (Collins and Singer, 1999)



Iteration 20

<u>101 rules</u>

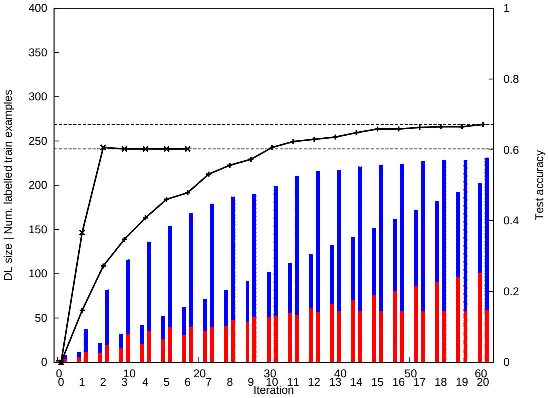
1.0 context: served
.969 context: year*
.996 context: commut*
.996 context: life*
.996 context: life
⋮

<u>101 rules</u>

1.0 context: reads
.991 context: read*
.991 next: from*
.991 next: from
.989 context: quot*
⋮

<u>train</u>
172
59

Yarowsky-cautious algorithm (Collins and Singer, 1999)



101 rules

1.0 context: served
.969 context: year*
.996 context: commut*
.996 context: life*
.996 context: life
⋮

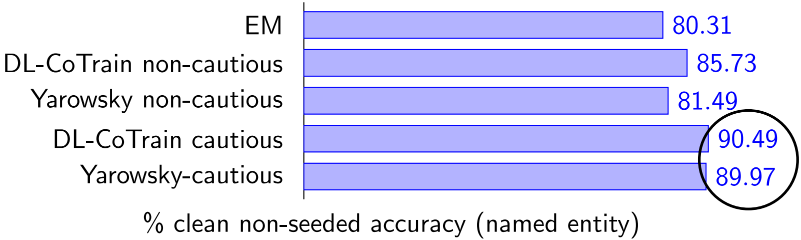
101 rules

1.0 context: reads
.991 context: read*
.991 next: from*
.991 next: from
.989 context: quot*
⋮

train
172
59

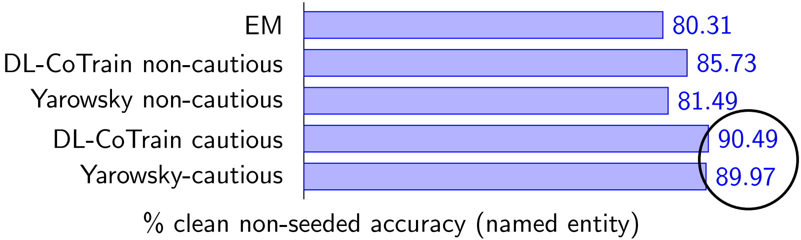
Iteration 20

Yarowsky-cautious vs. co-training and EM



statistically equivalent

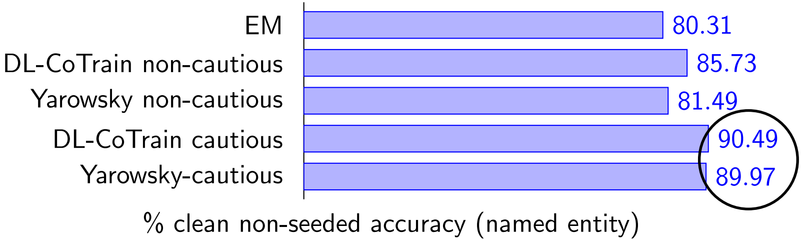
Yarowsky-cautious vs. co-training and EM



statistically equivalent

- Yarowsky performs well

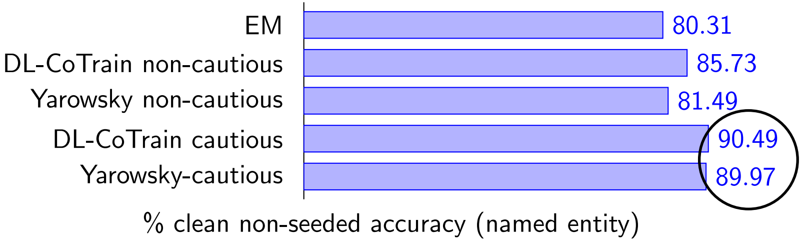
Yarowsky-cautious vs. co-training and EM



statistically equivalent

- Yarowsky performs well
- Cautiousness is important

Yarowsky-cautious vs. co-training and EM

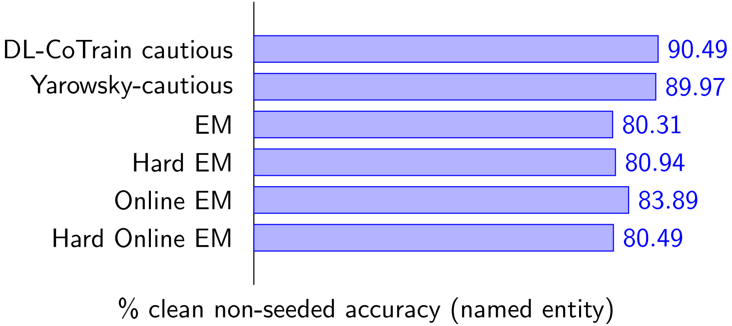


statistically equivalent

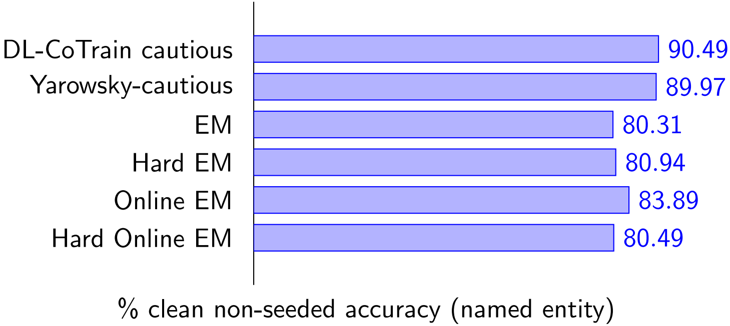
- Yarowsky performs well
- Cautiousness is important
- Yarowsky does not need views

Did we really do EM right?

Did we really do EM right?



Did we really do EM right?



Multiple runs of EM. Variance of results:

- EM: $\pm.34$
- Hard EM: ±2.53
- Online EM: $\pm.45$
- Hard Online EM: $\pm.68$

Yarowsky algorithm: (Abney, 2004)'s analysis

Yarowsky algorithm lacks theoretical analysis

Yarowsky algorithm: (Abney, 2004)'s analysis

Yarowsky algorithm lacks theoretical analysis

- (Abney, 2004) gives bounds for some variants
  (no cautiousness, no algorithm)

Yarowsky algorithm: (Abney, 2004)'s analysis

Yarowsky algorithm lacks theoretical analysis

- (Abney, 2004) gives bounds for some variants
  (no cautiousness, no algorithm)
- Basis for our work

Yarowsky algorithm: (Abney, 2004)'s analysis

Yarowsky algorithm lacks theoretical analysis

- (Abney, 2004) gives bounds for some variants
  (no cautiousness, no algorithm)
- Basis for our work

  Training examples x, labels j:

  - Full time should be served for each **sentence** .
  - The Liberals inserted a **sentence** of 14 words which reads :
  - They get a concurrent **sentence** with no additional time added to their sentence .
  - The words tax relief appeared in every second **sentence** in the federal government's throne speech .

  $\vdots$

labelling distributions $\phi_x(j)$
peaked for labelled example x
uniform for unlabelled example x

Yarowsky algorithm: (Abney, 2004)'s analysis

Yarowsky algorithm lacks theoretical analysis

- (Abney, 2004) gives bounds for some variants
  (no cautiousness, no algorithm)
- Basis for our work

  Training examples x, labels j:

  - Full time should be served for each **sentence** .
  - The Liberals inserted a **sentence** of 14 words which reads :
  - They get a concurrent **sentence** with no additional time added to their sentence .
  - The words tax relief appeared in every second **sentence** in the federal government's throne speech .

  ⋮

  labelling distributions $\phi_x(j)$
  peaked for labelled example x
  uniform for unlabelled example x

  Features f, labels j:

  - context: reads
  - context: served
  - context: inmate
  - next: the
  - context: article
  - previous: introductory
  - previous: passing
  - next: said

  ⋮

  $\longleftarrow$ parameter distributions $\theta_f(j)$
  normalized DL scores for feature f
  DL chooses $\arg\max_j \max_{f \in F_x} \theta_f(j)$

Yarowsky algorithm: (Abney, 2004)'s analysis

Yarowsky algorithm lacks theoretical analysis

- (Abney, 2004) gives bounds for some variants
  (no cautiousness, no algorithm)
- Basis for our work

  Training examples x, labels j:

  - Full time should be served for each **sentence** .
  - The Liberals inserted a **sentence** of 14 words which reads :
  - They get a concurrent **sentence** with no additional time added to their sentence .
  - The words tax relief appeared in every second **sentence** in the federal government's throne speech .

  ⋮

  Features f, labels j:

  - context: reads
  - context: served
  - context: inmate
  - next: the
  - context: article
  - previous: introductory
  - previous: passing
  - next: said

  ⋮

labelling distributions $\phi_x(j)$
peaked for labelled example x
uniform for unlabelled example x

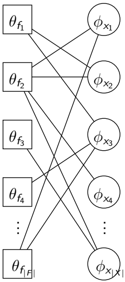
$\longleftarrow$ parameter distributions $\theta_f(j)$
normalized DL scores for feature f
DL chooses $\arg\max_j \max_{f \in F_x} \theta_f(j)$
alternative: $\arg\max_j \sum_{f \in F_x} \theta_f(j)$

Yarowsky algorithm: (Haffari and Sarkar, 2007)'s analysis

- (Haffari and Sarkar, 2007) extend (Abney, 2004) to bipartite graph representation
  (polytime algorithm; no cautiousness)

Yarowsky algorithm: (Haffari and Sarkar, 2007)'s analysis

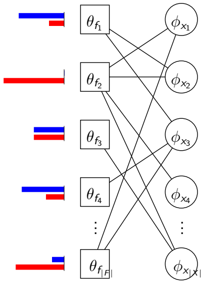
- (Haffari and Sarkar, 2007) extend (Abney, 2004) to bipartite graph representation
  (polytime algorithm; no cautiousness)



Yarowsky algorithm: (Haffari and Sarkar, 2007)'s analysis

- (Haffari and Sarkar, 2007) extend (Abney, 2004) to bipartite graph representation
  (polytime algorithm; no cautiousness)

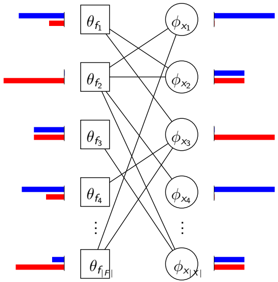
features f
parameter distributions $\theta_f(j)$



Yarowsky algorithm: (Haffari and Sarkar, 2007)'s analysis

- (Haffari and Sarkar, 2007) extend (Abney, 2004) to bipartite graph representation
  (polytime algorithm; no cautiousness)

features f
parameter distributions $\theta_f(j)$

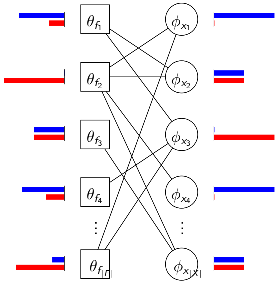


examples x
labelling distributions $\phi_x(j)$

Yarowsky algorithm: (Haffari and Sarkar, 2007)'s analysis

- (Haffari and Sarkar, 2007) extend (Abney, 2004) to bipartite graph representation
  (polytime algorithm; no cautiousness)

features f
parameter distributions $\theta_f(j)$



examples x
labelling distributions $\phi_x(j)$

algorithm: $\overrightarrow{}$ $\overleftarrow{}$ fix one side, update other

Objective Function

- KL divergence between two probability distributions:

$$KL(p||q) = \sum_i p(i) \log \frac{p(i)}{q(i)}$$

Objective Function

- KL divergence between two probability distributions:

$$KL(p||q) = \sum_i p(i) \log \frac{p(i)}{q(i)}$$

- Entropy of a distribution:

$$H(p) = -\sum_i p(i) \log p(i)$$

Objective Function

- KL divergence between two probability distributions:

$$KL(p||q) = \sum_i p(i) \log \frac{p(i)}{q(i)}$$

- Entropy of a distribution:

$$H(p) = -\sum_i p(i) \log p(i)$$

- The Objective Function:

$$\mathcal{K}(\phi, \theta) = \sum_{(f_i, x_j) \in \text{Edges}} KL(\theta_{f_i} || \phi_{x_j}) + H(\theta_{f_i}) + H(\phi_{x_j}) + \text{Regularizer}$$

Objective Function

- KL divergence between two probability distributions:

$$KL(p||q) = \sum_i p(i) \log \frac{p(i)}{q(i)}$$

- Entropy of a distribution:

$$H(p) = -\sum_i p(i) \log p(i)$$

- The Objective Function:

$$\mathcal{K}(\phi, \theta) = \sum_{(f_i, x_j) \in \text{Edges}} KL(\theta_{f_i} || \phi_{x_j}) + H(\theta_{f_i}) + H(\phi_{x_j}) + \text{Regularizer}$$

- Reduce uncertainty in the labelling distribution while respecting the labeled data

Generalized Objective Function

- Bregman divergence between two probability distributions:

$$
\begin{aligned}
B_\psi(p||q) &= \sum_i \psi(p(i)) - \psi(q(i)) - \psi'(q(i))(p(i) - q(i)) \\
B_{t \log t}(p||q) &= KL(p||q)
\end{aligned}
$$

Generalized Objective Function

- Bregman divergence between two probability distributions:

$$
\begin{aligned}
B_\psi(p||q) &= \sum_i \psi(p(i)) - \psi(q(i)) - \psi'(q(i))(p(i) - q(i)) \\
B_{t\log t}(p||q) &= KL(p||q)
\end{aligned}
$$

- ψ-Entropy of a distribution:

$$
\begin{aligned}
H_\psi(p) &= -\sum_i \psi(p(i)) \\
H_{t\log t}(p) &= H(p)
\end{aligned}
$$

Generalized Objective Function

- Bregman divergence between two probability distributions:

$$\begin{aligned} B_\psi(p||q) &= \sum_i \psi(p(i)) - \psi(q(i)) - \psi'(q(i))(p(i) - q(i)) \\ B_{t\log t}(p||q) &= KL(p||q) \end{aligned}$$

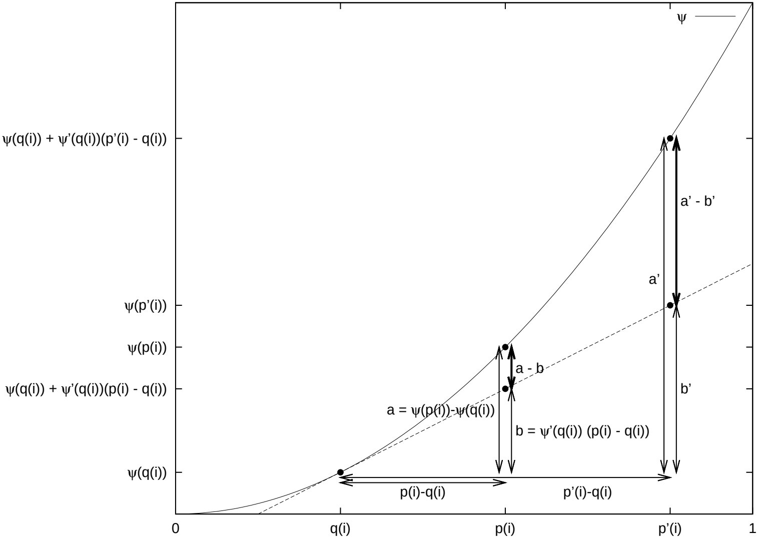
- ψ-Entropy of a distribution:

$$\begin{aligned} H_\psi(p) &= -\sum_i \psi(p(i)) \\ H_{t\log t}(p) &= H(p) \end{aligned}$$

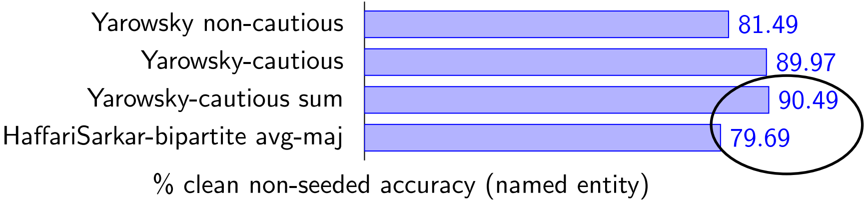
- The Generalized Objective Function:

$$\mathcal{K}_\psi(\phi, \theta) = \sum_{(f_i, x_j) \in \mathrm{Edges}} B_\psi(\theta_{f_i}||\phi_{x_j}) + H_\psi(\theta_{f_i}) + H_\psi(\phi_{x_j}) + \mathrm{Regularizer}$$

Generalized Objective Function

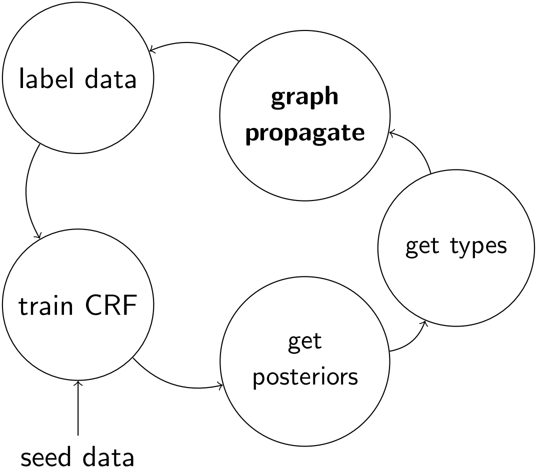


Variants from (Abney, 2004; Haffari and Sarkar, 2007)



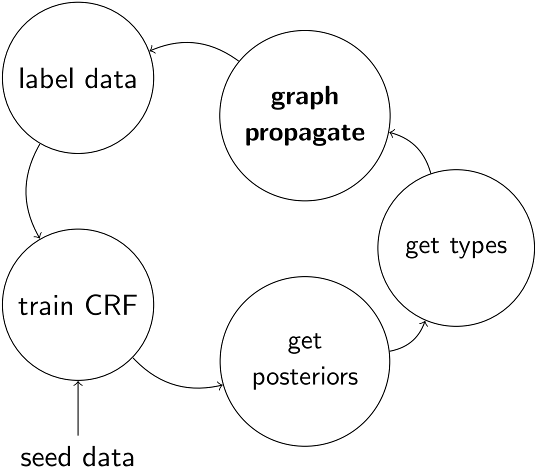
Graph-based Propagation (Subramanya et al., 2010)

Self-training with CRFs:

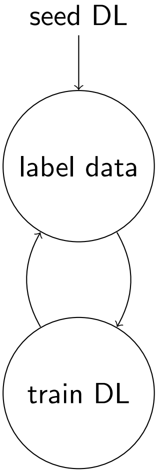


Graph-based Propagation (Subramanya et al., 2010)

Self-training with CRFs:



Compare with Yarowsky:



Our contributions

1. A **cautious**, **well-performing** Yarowsky variant with a **per-iteration objective**

Our contributions

1. A **cautious**, **well-performing** Yarowsky variant with a **per-iteration objective**
2. **Unification of various bootstrapping algorithms**: (Collins and Singer, 1999), (Abney, 2004), (Haffari and Sarkar, 2007), (Subramanya et al., 2010)

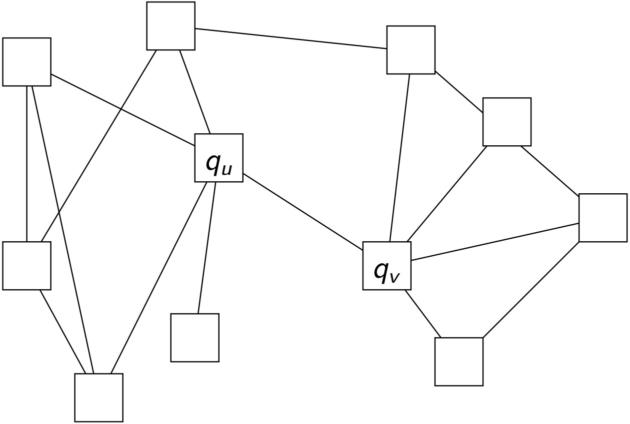
Our contributions

1. A **cautious**, **well-performing** Yarowsky variant with a **per-iteration objective**
2. **Unification of various bootstrapping algorithms**: (Collins and Singer, 1999), (Abney, 2004), (Haffari and Sarkar, 2007), (Subramanya et al., 2010)
3. More evidence that **cautiousness** is important

Graph propagation

(Subramanya et al., 2010)'s propagation:

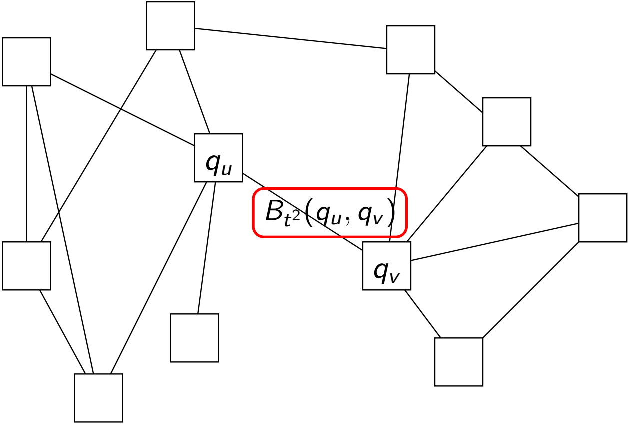
$$\mu \sum_{\substack{u \in \mathcal{V} \\ v \in \mathcal{N}(u)}} w_{uv} \; B_{t^2}(q_u, q_v) \; + \nu \sum_{u \in V} \; B_{t^2}(q_u, U)$$



Graph propagation

(Subramanya et al., 2010)'s propagation:

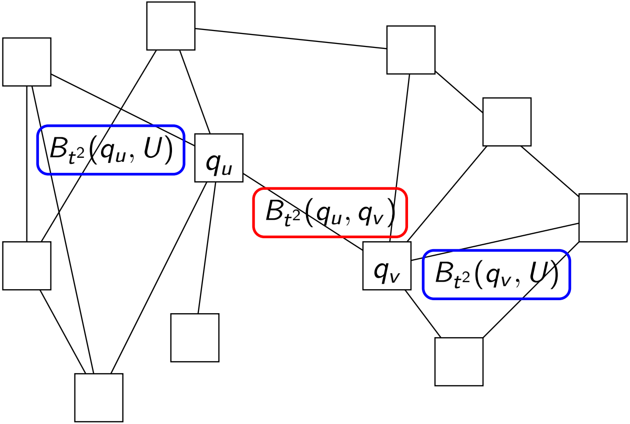
$$\mu \sum_{\substack{u \in \mathcal{V} \\ v \in \mathcal{N}(u)}} w_{uv} \boxed{B_{t^2}(q_u, q_v)} + \nu \sum_{u \in V} B_{t^2}(q_u, U)$$



Graph propagation

(Subramanya et al., 2010)'s propagation:

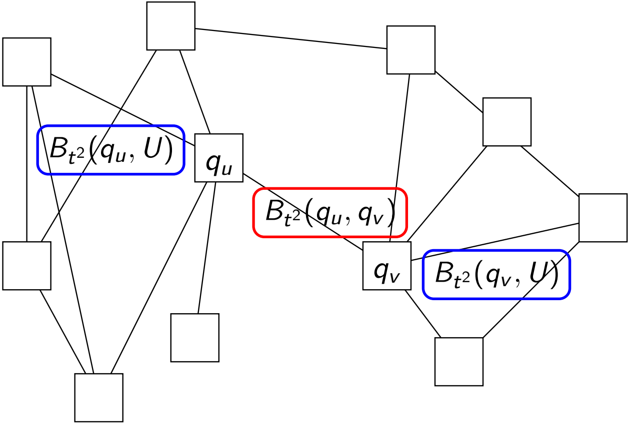
$$\mu \sum_{\substack{u \in \mathcal{V} \\ v \in \mathcal{N}(u)}} w_{uv} \boxed{B_{t^2}(q_u, q_v)} + \nu \sum_{u \in V} \boxed{B_{t^2}(q_u, U)}$$



Graph propagation

(Subramanya et al., 2010)'s propagation:

$$\mu \sum_{\substack{u \in \mathcal{V} \\ v \in \mathcal{N}(u)}} w_{uv} \boxed{B_{t^2}(q_u, q_v)} + \nu \sum_{u \in V} \boxed{B_{t^2}(q_u, U)}$$

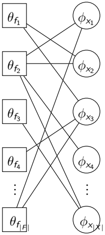


efficient iterative updates

Using graph propagation

$$\mu \left(\sum_{\substack{u \in \mathcal{V} \\ v \in \mathcal{N}(u)}} w_{uv} B_{t^2}(q_u, q_v) + H_{t^2}(q_u) \right) + \nu \sum_{u \in V} B_{t^2}(q_u, U)$$

ϕ-θ:



use bipartite graph from

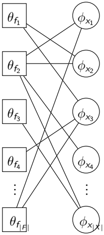
(Haffari and Sarkar, 2007)

(motivated by similar objective)

Using graph propagation

$$\mu \left(\sum_{\substack{u \in \mathcal{V} \\ v \in \mathcal{N}(u)}} w_{uv} B_{t^2}(q_u, q_v) + H_{t^2}(q_u) \right) + \nu \sum_{u \in V} B_{t^2}(q_u, U)$$

ϕ-θ:

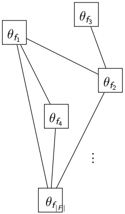


use bipartite graph from

(Haffari and Sarkar, 2007)

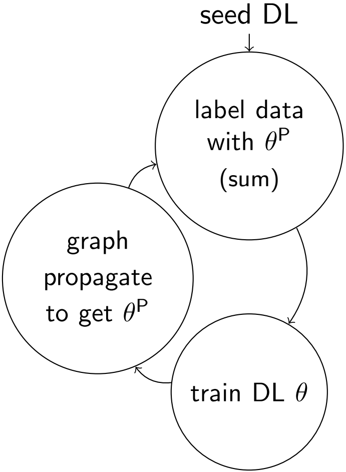
(motivated by similar objective)

θ-only:

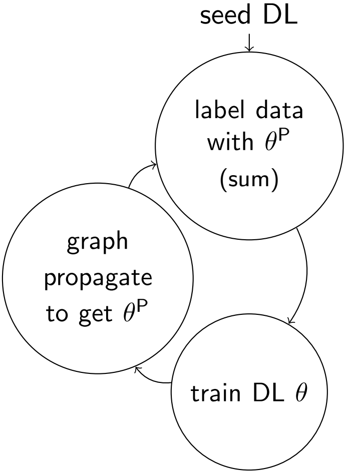


use only θ in unipartite graph

Yarowsky-prop (our algorithm)

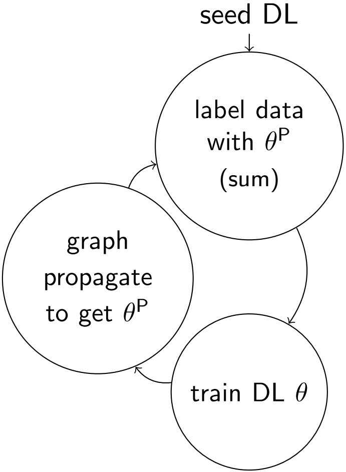


Yarowsky-prop (our algorithm)



Can use ϕ-θ (bipartite)
or θ-only (unipartite)
(or two more, in ACL 2012 paper)

Yarowsky-prop (our algorithm)

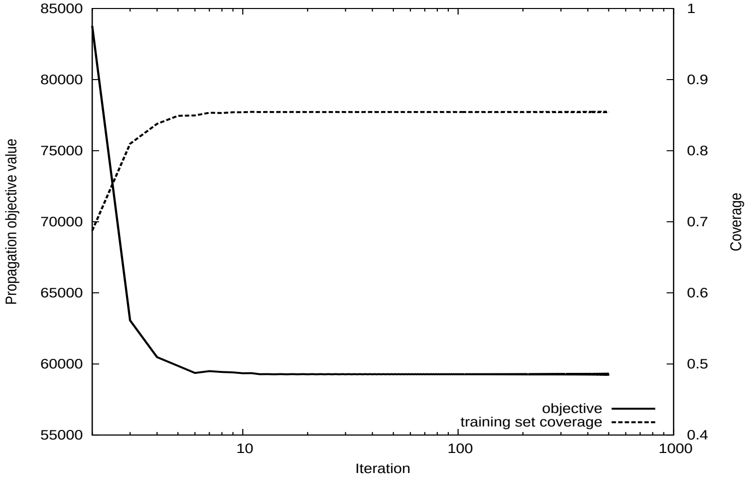


Can use ϕ-θ (bipartite) or θ-only (unipartite)
(or two more, in ACL 2012 paper)

- Optimizes (Subramanya et al., 2010)'s objective per iteration
- Use cautiousness decisions of θ, label with θ^{P}

Yarowsky-prop: objective behaviour

$$\mu \left(\sum_{\substack{u \in \mathcal{V} \\ v \in \mathcal{N}(u)}} w_{uv} B_{t^2}(q_u, q_v) + H_{t^2}(q_u) \right) + \nu \sum_{u \in V} B_{t^2}(q_u, U)$$



The basic Yarowsky algorithm.

Require: training data X and a seed DL $\theta^{(0)}$

1: **for** iteration $t = 1, 2, \ldots$ to maximum or convergence **do**

2: apply $\theta^{(t-1)}$ to X to produce $Y^{(t)}$

3: train a new DL $\theta^{(t)}$ on $Y^{(t)}$, keeping only rules with score above ζ

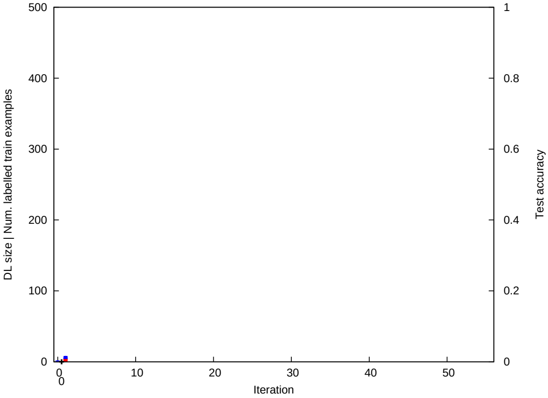
4: **end for**

5: train a final DL θ on the last $Y^{(t)}$ // retraining step

Yarowsky-prop algorithm (θ-only form)

let θ_{fj} be the scores of the seed rules // crf_train
for iteration t to maximum or convergence **do**
    let $\pi_x(j) = \frac{1}{|F_x|} \sum_{f \in F_x} \theta_{fj}$ // post._decode
    let $\theta^{\mathsf{T}}_{fj} = \frac{\sum_{x \in X_f} \pi_x(j)}{|X_f|}$ // token_to_type
    propagate θ^{T} to get θ^{P} // graph_propagate
    label the data with θ^{P} // viterbi_decode; *cautiousness*
    train a new DL θ_{fj} // crf_train
end for

Yarowsky-prop-cautious: behaviour



1 rule

1.0 context: served

1 rule

1.0 context: reads

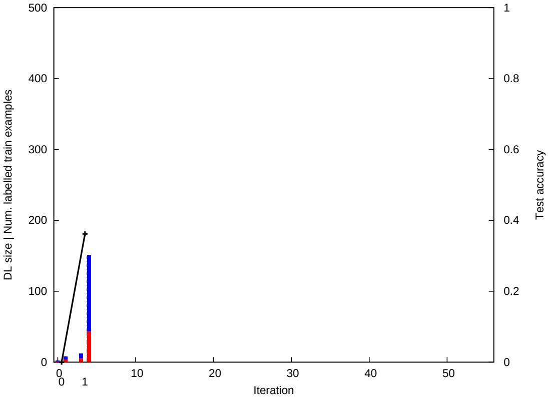
train

4

4

Iteration 0

Yarowsky-prop-cautious: behaviour



6 rules

1.0 context: served
.976 context: serv*
.976 context: served
.955 context: inmat*
.955 context: releas*

⋮

6 rules

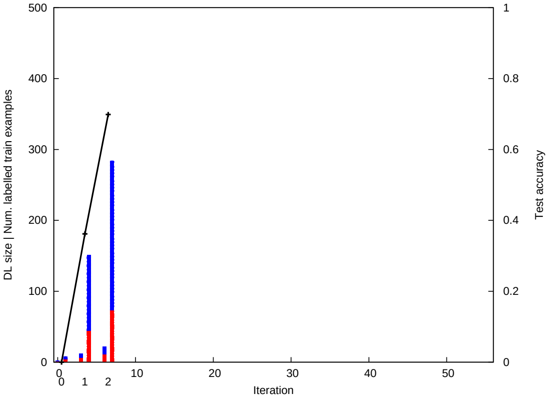
1.0 context: reads
.976 context: read*
.976 context: reads
.969 next: read*
.969 next: reads

⋮

train
107
44

Iteration 1

Yarowsky-prop-cautious: behaviour



11 rules

1.0 context: served
.995 context: serv*
.989 context: serve
.986 context: serving
.984 context: life*

⋮

11 rules

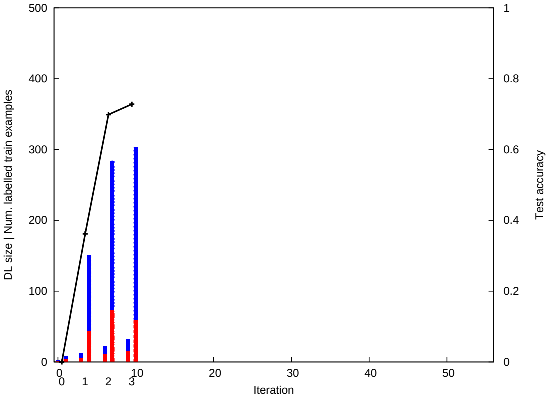
1.0 context: reads
.991 context: read*
.984 context: read
.976 context: reads
.969 next: from*

⋮

train
211
73

Iteration 2

Yarowsky-prop-cautious: behaviour



16 rules

1.0 context: served
.996 context: life*
.996 context: life
.995 context: serv*
.995 context: prison*
⋮

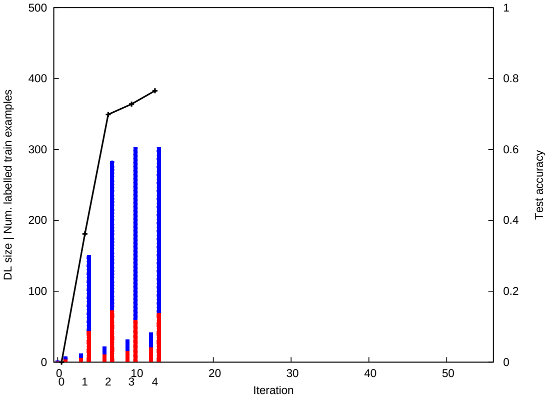
16 rules

1.0 context: reads
.991 context: read*
.991 next: from*
.991 next: from
.984 context: read
⋮

train
243
60

Iteration 3

Yarowsky-prop-cautious: behaviour



Iteration 4

21 rules

1.0	context: served
.996	context: commut*
.996	context: life*
.996	context: life
.995	context: serv*

⋮

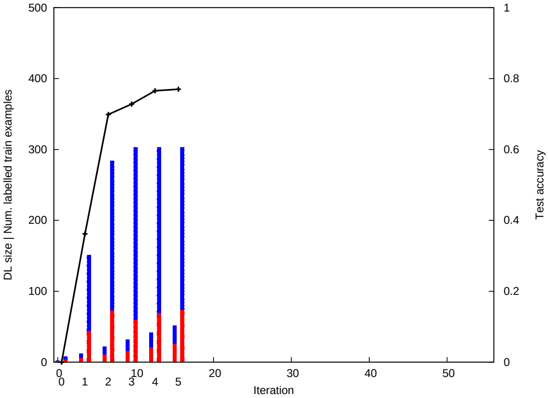
21 rules

1.0	context: reads
.991	context: read*
.991	next: from*
.991	next: from
.989	context: quot*

⋮

train
233
70

Yarowsky-prop-cautious: behaviour



26 rules

1.0 context: served
.996 context: commut*
.996 context: life*
.996 context: life
.995 context: serv*

⋮

26 rules

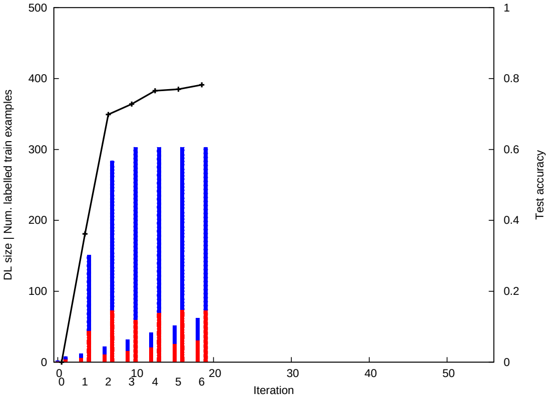
1.0 context: reads
.991 context: read*
.991 next: from*
.991 next: from
.989 context: quot*

⋮

train
229
74

Iteration 5

Yarowsky-prop-cautious: behaviour



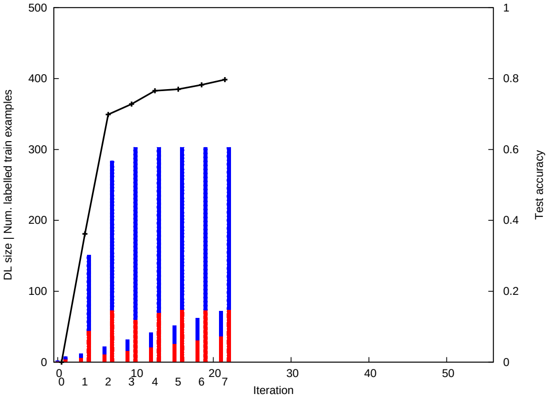
31 rules
1.0 context: served
.996 context: commut*
.996 context: life*
.996 context: life
.995 context: serv*
⋮

31 rules
1.0 context: reads
.991 context: read*
.991 next: from*
.991 next: from
.989 context: quot*
⋮

train
230
73

Iteration 6

Yarowsky-prop-cautious: behaviour



<u>36 rules</u>

1.0 context: served
.996 context: commut*
.996 context: life*
.996 context: life
.995 context: serv*

⋮

<u>36 rules</u>

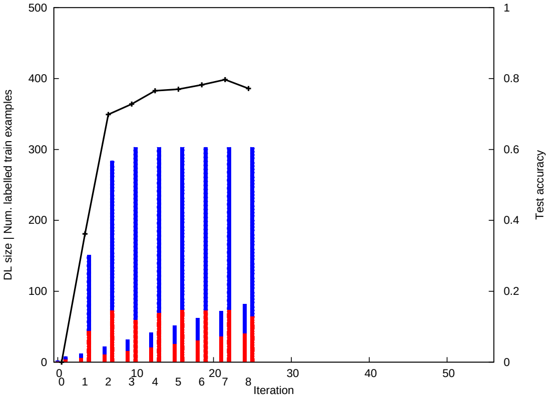
1.0 context: reads
.991 context: read*
.989 context: quot*
.988 context: quote
.984 context: put*

⋮

<u>train</u>
229
74

Iteration 7

Yarowsky-prop-cautious: behaviour



41 rules

1.0 context: served
.997 context: year*
.996 context: commut*
.996 context: life*
.996 context: life
⋮

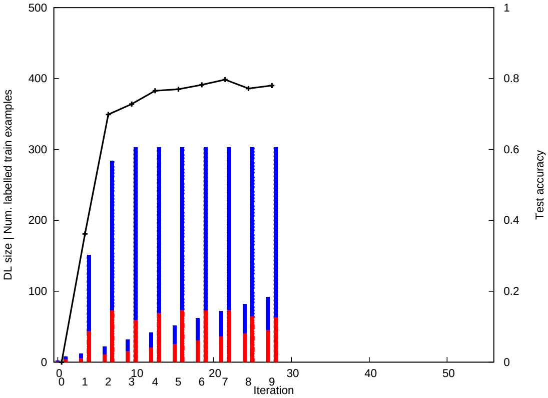
41 rules

1.0 context: reads
.991 context: read*
.989 context: quot*
.988 context: quote
.984 context: put*
⋮

train
238
65

Iteration 8

Yarowsky-prop-cautious: behaviour



<u>46 rules</u>

1.0 context: served
.997 context: year*
.996 context: commut*
.996 context: life*
.996 context: life

⋮

<u>46 rules</u>

1.0 context: reads
.989 context: quot*
.988 context: quote
.984 context: read
.984 previous: thi*

⋮

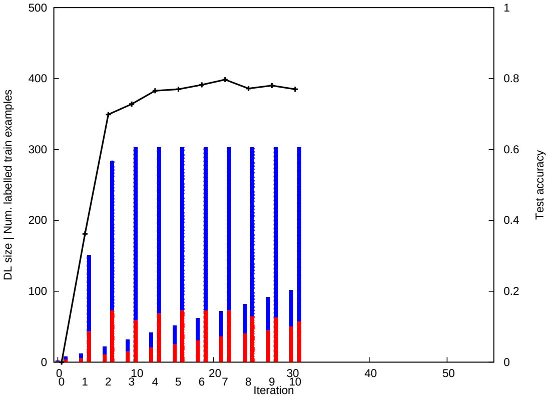
<u>train</u>

240

63

Iteration 9

Yarowsky-prop-cautious: behaviour



51 rules

1.0 context: served
.997 context: year*
.996 context: commut*
.996 context: life*
.996 context: life
⋮

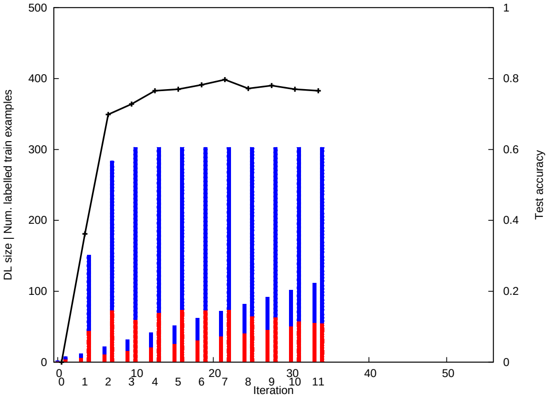
51 rules

1.0 context: reads
.990 context: read*
.989 context: quot*
.988 context: quote
.984 context: read
⋮

train
245
58

Iteration 10

Yarowsky-prop-cautious: behaviour



56 rules

1.0	context: served
.997	context: year*
.996	context: commut*
.996	context: death*
.996	context: life*

⋮

56 rules

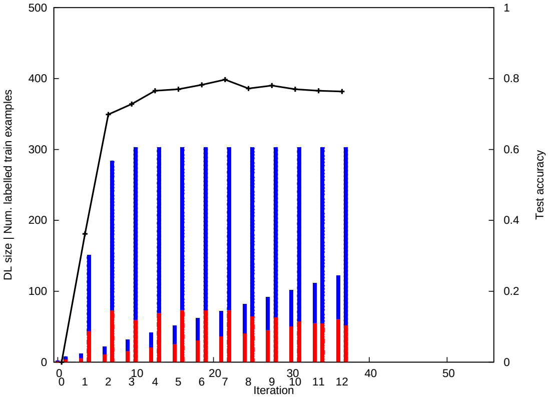
1.0	context: reads
.989	context: quot*
.988	context: quote
.984	context: read
.984	next: :*

⋮

train
248
55

Iteration 11

Yarowsky-prop-cautious: behaviour



61 rules

1.0	context: served
.997	context: year*
.996	context: commut*
.996	context: death*
.996	context: life*

⋮

61 rules

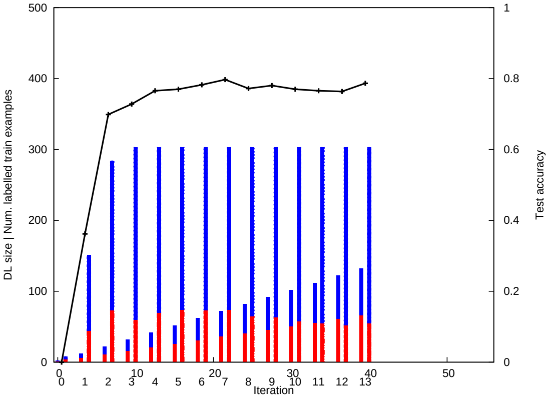
1.0	context: reads
.990	context: read*
.989	context: quot*
.988	context: quote
.984	context: read

⋮

train
251
52

Iteration 12

Yarowsky-prop-cautious: behaviour



66 rules

1.0	context: served
.971	previous: the*
.971	previous: the
.997	context: year*
.996	context: commut*

⋮

66 rules

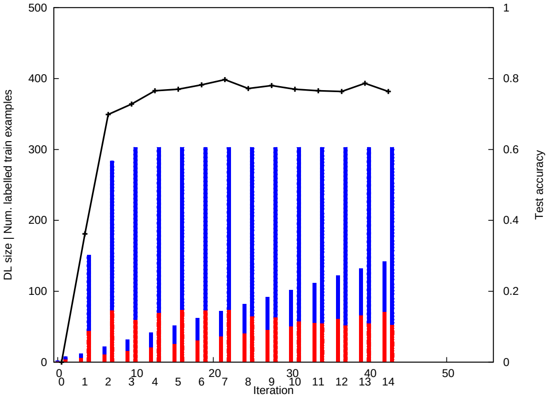
1.0	context: reads
.989	context: quot*
.988	context: quote
.984	context: read
.984	next: :*

⋮

train
248
55

Iteration 13

Yarowsky-prop-cautious: behaviour



<u>71 rules</u>

1.0 context: served
.971 previous: the*
.971 previous: the
.997 context: year*
.996 context: commut*

⋮

<u>71 rules</u>

1.0 context: reads
.990 context: read*
.989 context: quot*
.988 context: quote
.984 context: read

⋮

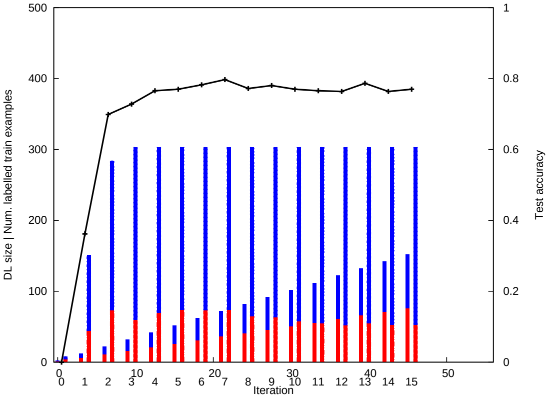
<u>train</u>

250

53

Iteration 14

Yarowsky-prop-cautious: behaviour



76 rules

1.0 context: served
.971 previous: the*
.971 previous: the
.997 context: year*
.996 context: commut*

⋮

76 rules

1.0 context: reads
.989 context: quot*
.988 context: quote
.984 context: read
.984 next: :*

⋮

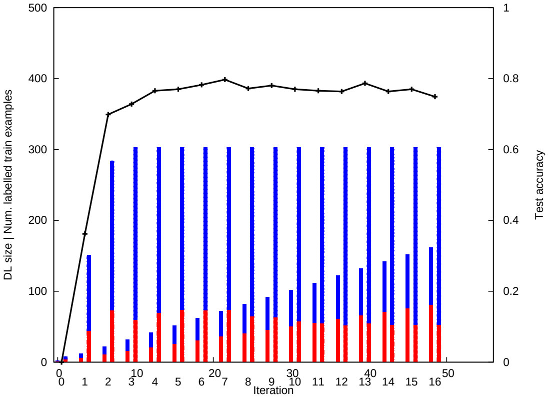
Iteration 15

train

250

53

Yarowsky-prop-cautious: behaviour



81 rules

1.0 context: served
.971 previous: the*
.971 previous: the
.997 context: year*
.996 context: commut*

⋮

81 rules

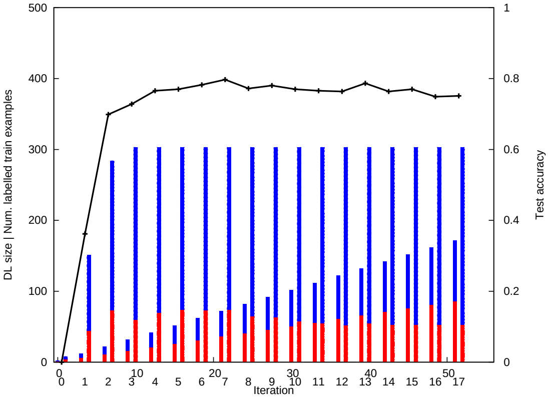
1.0 context: reads
.989 context: quot*
.988 context: quote
.984 context: read
.984 next: :*

⋮

Iteration 16

train
250
53

Yarowsky-prop-cautious: behaviour



86 rules

1.0 context: served
.971 previous: the*
.971 previous: the
.997 context: year*
.996 context: commut*

⋮

86 rules

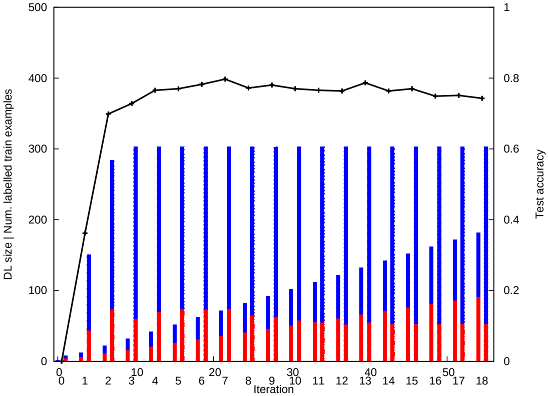
1.0 context: reads
.989 context: quot*
.988 context: quote
.984 context: read
.984 next: :*

⋮

train
250
53

Iteration 17

Yarowsky-prop-cautious: behaviour



91 rules

1.0 context: served
.971 previous: the*
.971 previous: the
.997 context: year*
.996 context: commut*
⋮

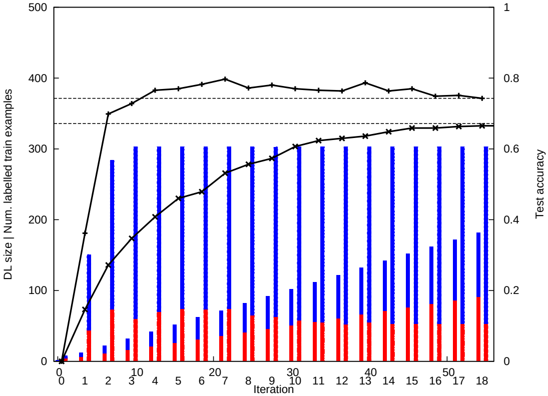
91 rules

1.0 context: reads
.989 context: quot*
.988 context: quote
.984 context: read
.984 next: :*
⋮

train
250
53

Iteration 18

Yarowsky-prop-cautious: behaviour



91 rules

1.0 context: served
.971 previous: the*
.971 previous: the
.997 context: year*
.996 context: commut*
⋮

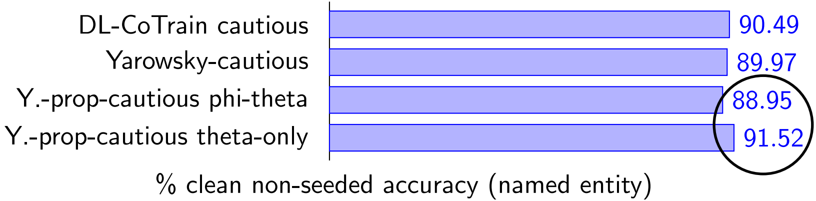
91 rules

1.0 context: reads
.989 context: quot*
.988 context: quote
.984 context: read
.984 next: :*
⋮

train
250
53

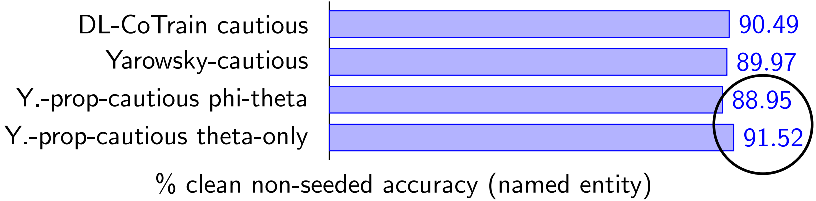
Iteration 18

Results



Statistically equivalent to DL-CoTrain

Results



Statistically equivalent to DL-CoTrain

But:

- No need for views
- Per-iteration objective

Correct Yarowsky-prop Examples

Gold label	Features
location	X0_Waukegan *X01_maker*, *X3_LEFT*
location	X0_Mexico, X42_president, X42_of *X11_president-of*, *X3_RIGHT*
location	X0_La-Jolla, X2_La, X2_Jolla *X01_company*, *X3_LEFT*

Figure: Named entity test set examples where Yarowsky-prop θ-only is correct and no other tested algorithms are correct.

Software available at
`https://github.com/sfu-natlang/yarowsky`

Thank you!

More results

Algorithm	Task			
	named entity	drug	land	sentence
Num. train	89305	134	1604	303
Num. test examples	962	386	1488	515
DL-CoTrain (non-cautious)	*85.73*	58.73	*77.72*	*51.05*
DL-CoTrain (cautious)	90.49	58.17	*77.72*	*65.69*
Yarowsky	*81.49*	*57.62*	78.41	*54.81*
Yarowsky-cautious	89.97	52.63	**78.48**	**76.99**
Yarowsky-cautious-sum	90.49	52.63	*77.72*	**76.99**
HS-bipartite avg-maj	*79.69*	50.14	*77.72*	*51.67*
EM	*80.31*	52.49	*31.12*	*65.23*
$\pm$	0.34	0.28	0.03	3.55
Yarowsky-prop ϕ-θ	*77.89*	51.80	*77.72*	*51.88*
Yarowsky-prop θ-only	*75.84*	52.91	*77.72*	*51.05*
Yarowsky-prop-cautious ϕ-θ	88.95	55.40	*77.72*	*72.18*
Yarowsky-prop-cautious θ-only	**91.52**	*57.06*	*77.72*	*73.22*

clean non-seeded accuracy

EM results

Algorithm	Task			
	named entity	drug	land	sentence
Num. train	89305	134	1604	303
Num. test examples	962	386	1488	515
Yarowsky	*81.49*	*57.62*	78.41	*54.81*
Yarowsky-cautious	89.97	52.63	**78.48**	**76.99**
Yarowsky-prop-cautious θ-only	**91.52**	*57.06*	*77.72*	*73.22*
EM	*80.31*	52.49	*31.12*	*65.23*
$\pm$	0.34	0.28	0.03	3.55
Hard EM	*80.95*	52.91	*40.12*	*63.47*
$\pm$	2.53	0.74	13.39	6.37
Online EM	*83.89*	54.29	*45.00*	*56.25*
$\pm$	0.45	0.94	21.29	3.28
Hard online EM	*80.41*	54.54	*50.51*	*56.28*
$\pm$	0.68	1.03	23.02	3.56

clean non-seeded accuracy

Decision lists

(Collins and Singer, 1999)'s DL scores:

$$\theta_{fj} \propto \frac{|\Lambda_{fj}| + \epsilon}{|\Lambda_f| + L\epsilon}$$

max definition of π (strict DL):

$$\pi_x(j) \propto \max_{f \in F_x} \theta_{fj}$$

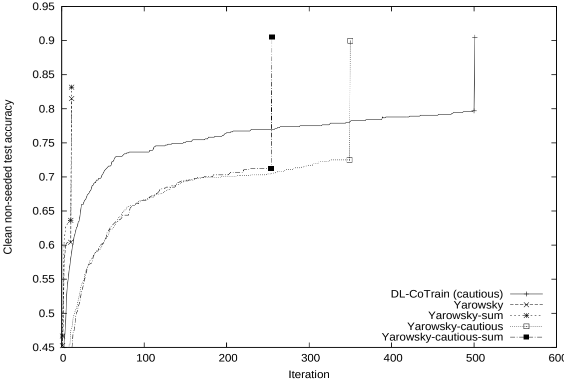
sum definition of π:

$$\pi_x(j) = \frac{1}{|F_x|} \sum_{f \in F_x} \theta_{fj}$$

HS-bipartite.

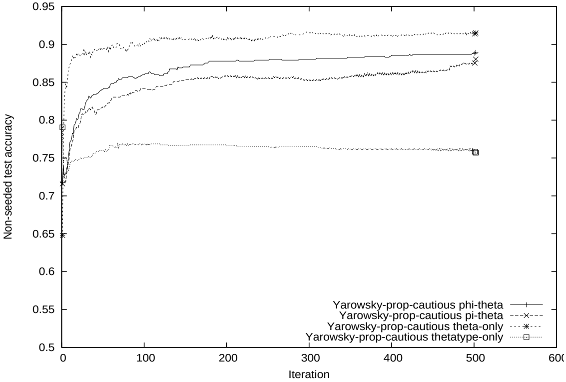
```
apply θ^(0) to X produce a labelling Y^(0)
for iteration t to maximum or convergence do
    for f ∈ F do
        let p = examples-to-feature({φ_x : x ∈ X_f})
        if p ≠ U then let θ_f = p
    end for
    for x ∈ X do
        let p = features-to-example({θ_f : f ∈ F_x})
        if p ≠ U then let φ_x = p
    end for
end for
```

Acuracy plot: (Collins and Singer, 1999) algorithms



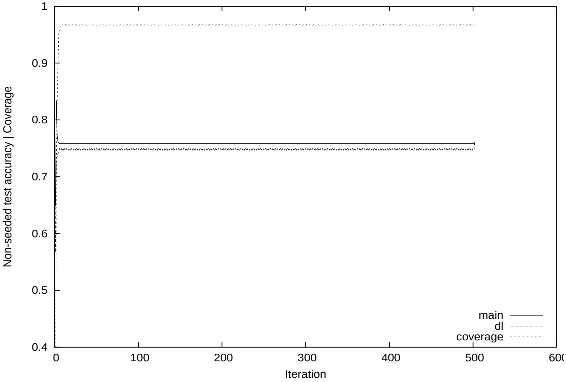
Non-seeded test accuracy versus iteration for various algorithms on named entity. The results for the Yarowsky-prop algorithms are for the propagated classifier θ^P, except for the final DL retraining iteration.

Acuracy plot: Yarowsky-prop cautious



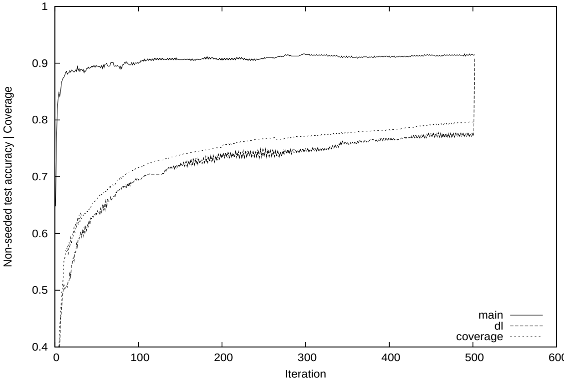
Non-seeded test accuracy versus iteration for various algorithms on named entity. The results for the Yarowsky-prop algorithms are for the propagated classifier θ^P, except for the final DL retraining iteration.

Accuracy and coverage plot: non-cautious



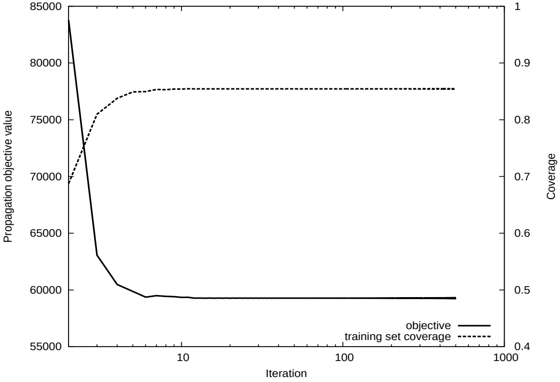
Internal train set coverage and non-seeded test accuracy (same scale) for Yarowsky-prop θ-only on named entity.

Accuracy and coverage plot: cautious



Internal train set coverage and non-seeded test accuracy (same scale) for Yarowsky-prop θ-only on named entity.

Objective plot

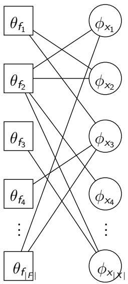


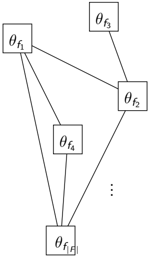
Non-seeded test accuracy (left axis), coverage (left axis, same scale), and objective value (right axis) for Yarowsky-prop ϕ-θ. Iterations are shown on a log scale. We omit the first iteration (where the DL contains only the seed rules) and start the plot at iteration 2 where there is a complete DL.

Graph structures for propagation.

Method	$\mathcal{V}$	$\mathcal{N}(u)$	q_u
ϕ-θ	$X \cup F$	$\mathcal{N}_x = F_x,\ \mathcal{N}_f = X_f$	$q_x = \phi_x,\ q_f = \theta_f$
π-θ	$X \cup F$	$\mathcal{N}_x = F_x,\ \mathcal{N}_f = X_f$	$q_x = \pi_x,\ q_f = \theta_f$
θ-only	F	$\mathcal{N}_f = \bigcup_{x \in X_f} F_x \setminus f$	$q_f = \theta_f$
θ^{T}-only	F	$\mathcal{N}_f = \bigcup_{x \in X_f} F_x \setminus f$	$q_f = \theta_f^{\mathsf{T}}$

$$\mu \sum_{\substack{u \in \mathcal{V} \\ v \in \mathcal{N}(u)}} w_{uv} B_{t^2}(q_u, q_v) + \nu \sum_{u \in V} B_{t^2}(q_u, U)$$





Abney, S. (2004).
Understanding the Yarowsky algorithm.
Computational Linguistics, 30(3).

Collins, M. and Singer, Y. (1999).
Unsupervised models for named entity classification.
In *In EMNLP 1999: Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 100–110.

Daume, H. (2011).
Seeding, transduction, out-of-sample error and the Microsoft approach...
Blog post at *http://nlpers.blogspot.com/2011/04/seeding-transduction-out-of-sample.html*.

Eisner, J. and Karakos, D. (2005).
Bootstrapping without the boot.
In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language*

Processing, pages 395–402, Vancouver, British Columbia, Canada. Association for Computational Linguistics.

Haffari, G. and Sarkar, A. (2007).
Analysis of semi-supervised learning with the Yarowsky algorithm.
In *UAI 2007, Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence, Vancouver, BC, Canada*, pages 159–166.

Subramanya, A., Petrov, S., and Pereira, F. (2010).
Efficient graph-based semi-supervised learning of structured tagging models.
In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 167–176, Cambridge, MA. Association for Computational Linguistics.

Yarowsky, D. (1995).
Unsupervised word sense disambiguation rivaling supervised methods.
In *Proceedings of the 33rd Annual Meeting of the Association*

for Computational Linguistics, pages 189–196, Cambridge, Massachusetts, USA. Association for Computational Linguistics.