RDN-Boost

A Guide

http://pages.cs.wisc.edu/~tushar/rdnboost/index.html

Task

- Given:
  - train(t)
  - car(t, c)
- Todo:
  - Learn rules for target(t)

Facts

train(T1)

car(T1, C1)

train(T2)

train(T3)

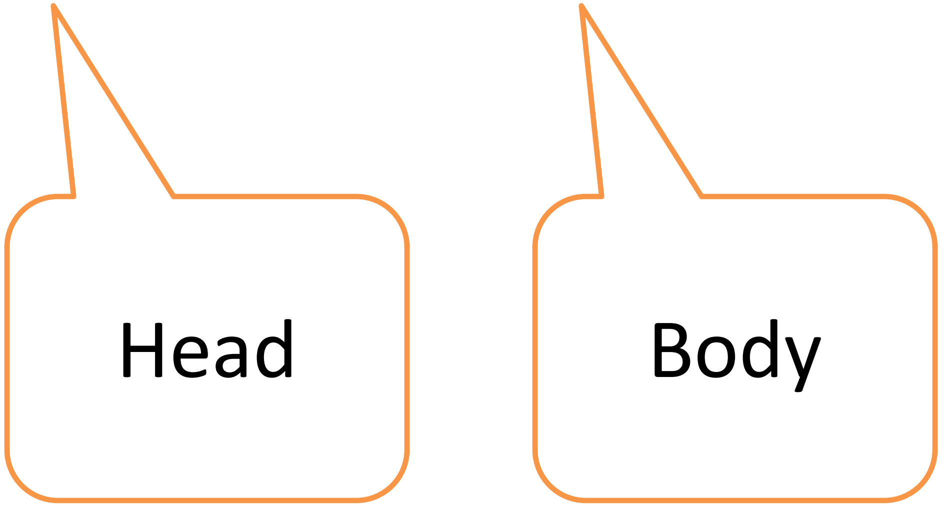
car(T3, C2)

car(T3,C3)

http://pages.cs.wisc.edu/~tushar/rdnboost/index.html

Problem 1

- target(X) is true, if train has a car
  - target(X) <- car(X,Y)



Head

Body

http://pages.cs.wisc.edu/~tushar/rdnboost/index.html

Inductive Logic Programming

- Start with target(X)
  - target(X) <- car(X,Z)
  - target(X) <- car(Y,X)
    - Does not make sense since car has car id as the second argument and target has train id as the first argument
  - target(X) <- car(Y,Z)
    - Does not help since the rule says that a train is of target type if some train has a car

http://pages.cs.wisc.edu/~tushar/rdnboost/index.html

Provide type information

- To avoid target(X) <- car(Y,X) provide type information

- mode: target(t)
- mode: car(t, c)

- But what about target(X) <- car(Y,Z) ?

http://pages.cs.wisc.edu/~tushar/rdnboost/index.html

Modes to the rescue

- car(t,c) must use the current train variable
  - i.e. variable of type t should already be mentioned before
- '+' in a mode exactly does that
- But the variable of type c in car may not be seen before
- '-' in a mode exactly does that

mode: car(+t, -c)

http://pages.cs.wisc.edu/~tushar/rdnboost/index.html

Problem 2

- Additional facts
  - big(c)
  - small(c)

- target(X) is true if there is a big car and a small car in the train
  - target(X) <- car(X,Y) , big(Y), car(X,Z), small(Z)

http://pages.cs.wisc.edu/~tushar/rdnboost/index.html

Modes

- car(+t, -c)
- big(+c)
- small(+c)

- big(-c) would give us rules like

  target(X) <- big(Y)

http://pages.cs.wisc.edu/~tushar/rdnboost/index.html

ILP search

- Target(X)
  - car(X,Y)
    - big(Y)
      - car(X,Z)
        - small(Z)
      - small(Y)
    - small(Y)
    - car(X,Z)

http://pages.cs.wisc.edu/~tushar/rdnboost/index.html

But ILP is greedy search

- Target(X)
  - car(X,Y)
    - big(Y)
      - car(X,Z)
        - small(Z)
      - small(Y)
    - small(Y)
    - car(X,Z)

Not informative

http://pages.cs.wisc.edu/~tushar/rdnboost/index.html

Don’t be so greedy

- Increase lookahead to 2
- Target(X)
  - car(X,Y), big(Y)
    - car(X,Z), small(Z)
    - …
  - …

- setParam: nodeSize=2.

http://pages.cs.wisc.edu/~tushar/rdnboost/index.html

Problem 3

- Add facts
  - animal(c, a)
  - a={Dog, Cat, Mouse}

- Target(X) if a car contains mouse

http://pages.cs.wisc.edu/~tushar/rdnboost/index.html

Possible rules

- Mode: animal(+c, -a)
- target(X) :- car(X, Y), animal(Y,A)
- We need to “ground” the variable A

http://pages.cs.wisc.edu/~tushar/rdnboost/index.html

#UseHash

- Mode: animal(+c, #a).
- Generated clauses
  - target(X) :- car(X, Y), animal(Y,"Dog")
  - target(X) :- car(X, Y), animal(Y,"Cat")
  - target(X) :- car(X, Y), animal(Y,"Mouse")

- Still need nodeSize=2

http://pages.cs.wisc.edu/~tushar/rdnboost/index.html

Problem 4

- A big car contains a mouse
  - Target(X) :- car(X,Y), big(Y), animal(Y,”Mouse”).
- Consider ILP search after target(X) <- car(X,Y), big(Y)
  - small(Y), animal(Y, “Dog”)
  - small(Y), animal(Y, “Cat”)
  - animal(Y, “Dog”), animal(Y, “Cat”)
  - …
- small, big, animal : are informative
- car is not

http://pages.cs.wisc.edu/~tushar/rdnboost/index.html

Use bridgers

- Bridgers connect facts
  - E.g. age, parents, segment
- Bridgers should not be counted
  - Infinite bridgers : car(X,Y), car(X,Z), car(X,A) …
- First bridger is free
  - car(X,Y) – size:0
  - car(X,Y), big(X) – size:1
  - car(X,Y), animal(X,”Dog”) – size:1
  - car(X,Y), car(X,Z) – size:1
- bridger: car/2.
- Keep nodeSize=1.

http://pages.cs.wisc.edu/~tushar/rdnboost/index.html

Citeseer

- Citation segmentation

- Given:
  - token(c, t)
  - punctuation(t)
  - wordString(t, w)
  - next(t, t)
- Todo:
  - field(t, f) f={‘author’, ‘title’, ‘venue’}

http://pages.cs.wisc.edu/~tushar/rdnboost/index.html

Multi-valued classification

- Learn one model for each label
- Change n-valued classification into n binary classification models
  - infield_title(t)
  - infield_author(t)
  - infield_venue(t)

http://pages.cs.wisc.edu/~tushar/rdnboost/index.html

Joint model

- Model/Rules for infield_title might be useful for infield_venue and vice versa
  - infield_title(T) <- next(T,P), punct(P), next(P,T1), infield_venue(T1)
- Specify all three predicates as query predicates

  -query infield_venue,infield_title,infield_author

- During inference, pick the most likely label
  - Has to be a post-processing step. Not their in code

http://pages.cs.wisc.edu/~tushar/rdnboost/index.html

Cora

- Citation clustering
- Given:
  - Title(b, t)
  - Author(b,a)
  - Venue(b, v)
  - TitleWord(t, w)
  - AuthorWord(a,w)
  - VenueWord(v,w)
- Todo:
  - sameBib(b, b)

http://pages.cs.wisc.edu/~tushar/rdnboost/index.html

Transitivity

- We might want the model to learn rules like
  - sameBib(X,Y) <- sameBib(X,Z), samebib(Z,Y)
- If we use sameBib(+b, -b)
  - sameBib(X,Y) <- sameBib(X,Y)
- The rule is perfect but not really useful for inference
- Force one variable to not be in head of clause
  - sameBib(`b, +b)
  - sameBib(+b, `b)

http://pages.cs.wisc.edu/~tushar/rdnboost/index.html

Code issues

- Cannot handle same predicates in head and body
- RDN-Boost will create recursive_<predicate> automatically
  - recursive_sameBib for Cora
- Specify modes as
  - recursive_sameBib(`b, +b)
  - recursive_sameBib(+b, `b)

Greedy search issues

- Intuitively a good rule would be
  - sameBib(X, Y) <- Title(X, T1), Title(Y, T2),
    sameTitle(T1, T2).
- No subset of predicates is "informative"
- Needs a node size of 3 or Title/2 as bridger

http://pages.cs.wisc.edu/~tushar/rdnboost/index.html

Mode overview

- + : variable must have appeared before
- - : variable can be new but *does not have to*
- # : ground the variable/use constant
- ` : variable must not be in the head
- @<val>: variable must take value <val>

New

http://pages.cs.wisc.edu/~tushar/rdnboost/index.html

Tree representations

- <train_folder>/models/
  - bRDNs/Trees/*tree : Trees as list of clauses
  - bRDNs/dotFiles/*dot: .dot files that can be used by graphViz to visualize
  - WILLtheories/*txt : Prolog format for trees. Also has human readable text version of all trees in one file

Sample tree

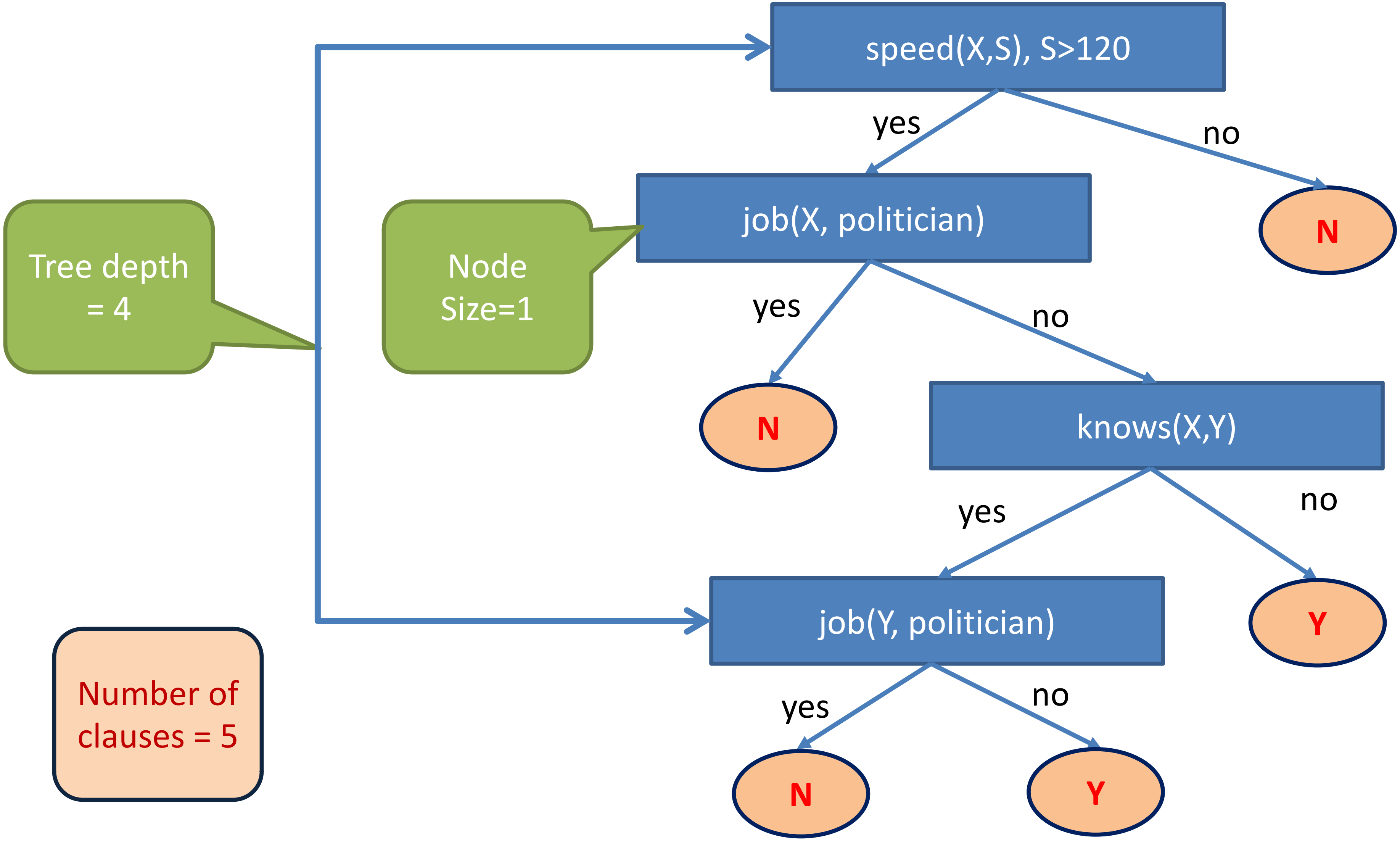
```
%%%%%  WILL-Produced Tree #1 @ 0:51:19 10/20/10.  [Using 13,141,856 memory cells.]  %%%%%

% FOR advisedby(A, B):
%   if ( professor(B) )
%   then if ( professor(A) )
%   | then return -0.1418510649004878;  // std dev = 0.000, 7.000 (wgt'ed) examples reached here.  /* #neg=7 */
%   | else if ( publication(C, B) )
%   | | then return 0.739727882467934;  // std dev = 0.323, 76.000 (wgt'ed) examples reached here.  /* #neg=9 #pos=67 */
%   | | else return 0.3781489350995123;  // std dev = 0.500, 25.000 (wgt'ed) examples reached here.  /* #neg=12 #pos=13 */
%   else return -0.1418510649004879;  // std dev = 0.000, 132.000 (wgt'ed) examples reached here.  /* #neg=132 */
```

http://pages.cs.wisc.edu/~tushar/rdnboost/index.html

Additional flags

- modelSuffix
  - Run multiple experiments with different values for this flag to prevent overwriting
- negPosRatio(default=2)
  - Each boosting iteration samples negative examples so that negative:positive ratio is 2:1
  - Most datasets have too many negatives

Tree parameters



Tree depth = 4

Node Size=1

Number of clauses = 5

http://pages.cs.wisc.edu/~tushar/rdnboost/index.html