Learning probability by comparison

Taisuke Sato

AIST

Outline

- PRISM: logic based probabilistic modeling language
  - Introduction
  - Sample sessions
- Rank learning
  - Parameter learning from pairwise comparison
  - Naïve Bayes and Bayesian network
  - Probabilistic context free grammar
  - Knowledge graph

Logic, Probability and Preference

Logic

Completeness theorem 1930

Probability

Foundations of the Theory of Probability 1933

Fenstad's representation theorem 1967

Probability distribution on possible worlds

Logic-based probabilistic modeling

Distribution semantics

PRISM

Tabling(DP)

Proposi-tionalized probability computation

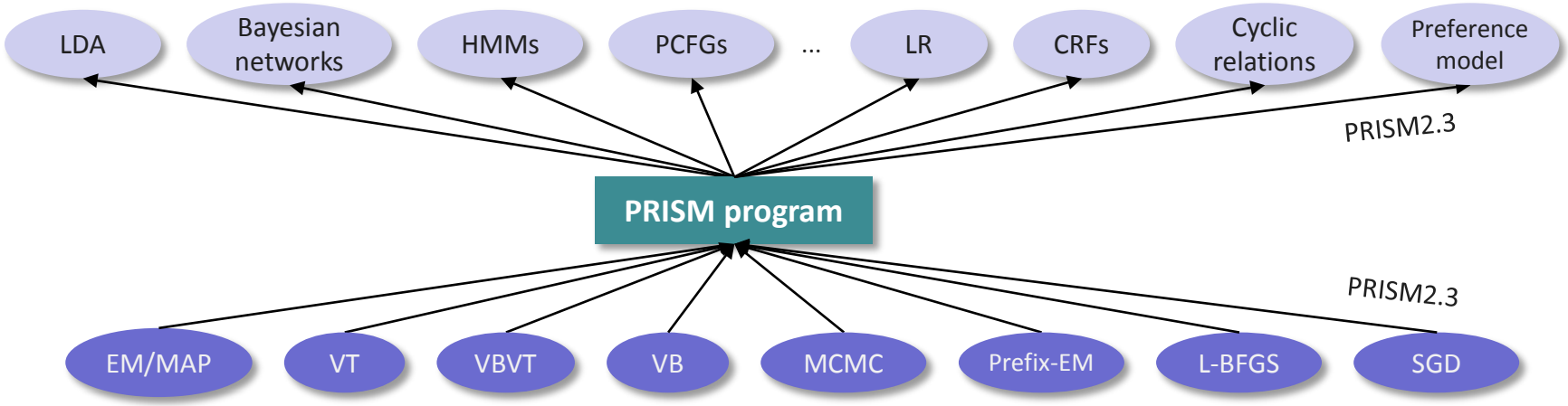
Machine learning

- Structure learning
- Parameter learning
- Viterbi inference
- Bayesian inference

Probabilistic preference modeling (PRISM2.3)

PRISM

- PRISM (http://rjida.meijo-u.ac.jp/prism/)
  - Logic-based probabilistic modeling language (Prolog(search)+C(probability))
  - Any learning method × any probabilistic model (program)



- PRISM2.2
  - Generative modeling (LDA, NB, BN, HMM, PCFG,…)
  - Discriminative modeling (logistic regression, linear-chain CRF, CRF-CFG,…)
- PRISM2.3 (released in Aug.)
  - Rank learning by SGD

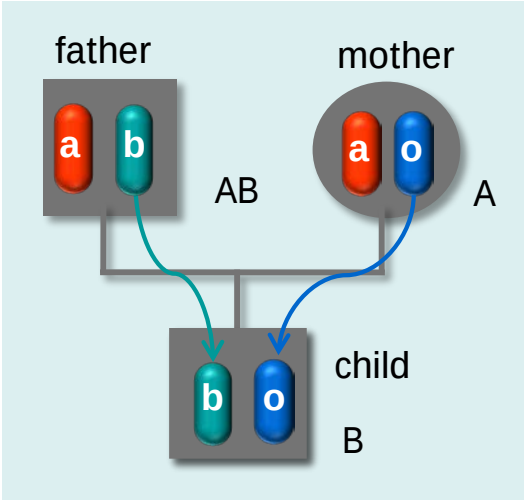
ABO blood type program

```
values(gene,[a,b,o],[0.5,0.2,0.3]).
```

msw(gene,a) is true with prob. 0.5

```
btype(P):-
   gtype(X,Y),
   ( X=Y → P=X ; X=o → P=Y ; Y=o → P=X ; P=ab ).

gtype(X,Y):-
   msw(gene,X),msw(gene,Y).
```



simulates gene inheritance from father (left) and mother (right)

Machine learning menu

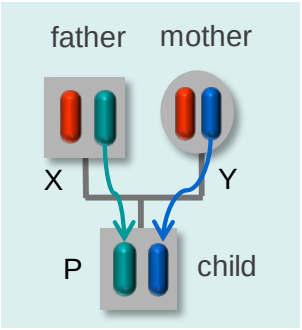
- Frequentist: a program defines $p(x,y|\theta)$ where hidden x generates observable y
  - Estimate θ from y
    - MLE $\theta^* = \mathrm{argmax}_\theta P(y|\theta)$ ← EM/MAP
    - $\theta^* = \mathrm{argmax}_\theta P(x^*,y|\theta)$ & $x^* = \mathrm{argmax}_x p(x,y|\theta^*)$ ← VT
- Bayesian: a program defines $p(x,y|\theta)p(\theta|\alpha)$
  - Compute marginal likelihood $p(y|\alpha) = \Sigma_x \int p(x,y,\theta|\alpha)\, d\theta$, posterior $p(\theta|y,\alpha)$ and predictive distribution $\int p(x|y_{new},\theta,\alpha)p(\theta|y,\alpha)d\theta$
    - variational Bayes (VB) ← VB, VB-VT
    - MCMC ← Metropolis-Hastings
- Conditional random field (CRF): a program defines $p(x|y,\theta)$
  - $\theta^* = \mathrm{argmax}_\theta p(x|y,\theta)$ ← LBFG
- Rank learning: a program defines $p(x,y|\theta)$
  - Given preference pairs $y_1 \succ y_2$, $y_3 \succ y_4$, ...
    learn θ s.t. $p(y_1|\theta)>p(y_2|\theta)$, $p(y_3|\theta)>p(y_4|\theta)$,… ← SGD

Built-in predicates

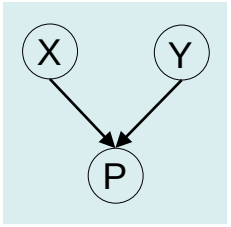
- Forward sampling -- sample, get_samples
- Probability computation -- prob,probf,…
- Parameter learning -- learn (with learn_mode=ml,vb,ml_vt,vb_vt)
  - EM/MAP
  - VT
- Bayesian inference
  - VB
  - VBVT
  - MCMC (by MH) -- mcmc,marg_mcmc_full,predict_mcmc_full,…
- Learning circular models
  - prefix-EM -- lin_prob,lin_learn,lin_viterbi,nonlin_prob,…
- Learning CRFs -- crf_prob,crf_learn,crf_viterbi,…
- Rank learning
  - SGD (with minibatch,adam.adadelta) -- rank_learn,rank
- Viterbi inference (most probable answer) -- viterbi,viterbig,n_viterbi,…
- model score (BIC, Cheesman-Stutz, VFE) -- learn_statistics
- smoothing -- hingsight,chindsight

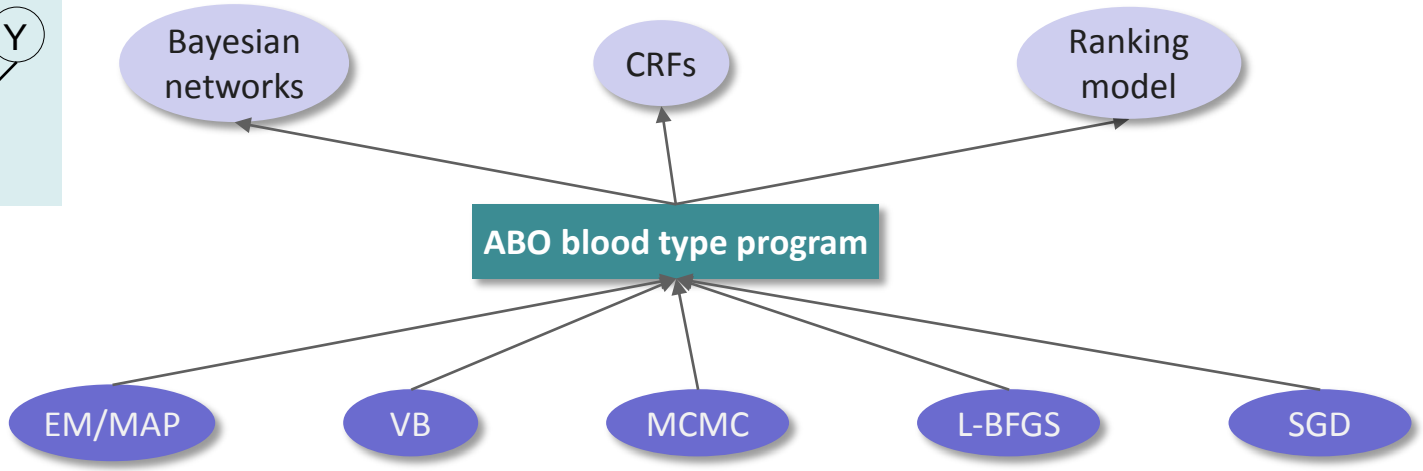
Write a program once, and apply everything!

Example




```
btype(P):-
   gtype(X,Y),
   ( X=Y → P=X ; X=o → P=Y
   ; Y=o → P=X ; P=ab )
gtype(X,Y):-
   msw(gene,X),msw(gene,Y)
```





Sample session

- Expl. graph and prob. computation

built-in predicate

```
?- prism(blood)
loading::blood.psm.out
```

```
btype(P):-
    gtype(X,Y),
    ( X=Y -> P=X ; X=o -> P=Y
    ; Y=o -> P=X ; P=ab ).
gtype(X,Y):- msw(gene,X),msw(gene,Y).
```

```
?- show_sw
Switch gene: unfixed_p: a (p: 0.5) b (p: 0.2) o (p: 0.3)
```

```
?- probf(btype(a))
btype(a) <=> gtype(a,a) v gtype(a,o) v gtype(o,a)
gtype(a,a) <=> msw(gene,a) & msw(gene,a)
gtype(a,o) <=> msw(gene,a) & msw(gene,o)
gtype(o,a) <=> msw(gene,o) & msw(gene,a)
```

```
?- prob(btype(a),P)
P = 0.550
```

Sample session

- MLE and Viterbi inference

```
?- D=[btype(a),btype(a),btype(ab),btype(o)],learn(D)
Exporting switch information to the EM routine ... done
#em-iters: 0(4) (Converged: -4.965)
Statistics on learning:
        Graph size: 18
        Number of switches: 1
        Number of switch instances: 3
        Number of iterations: 4
        Final log likelihood: -4.965

?- prob(btype(a),P)
P = 0.598

?- viterbif(btype(a))
btype(a)  <= gtype(a,a)
gtype(a,a)  <= msw(gene,a) & msw(gene,a)
```

Sample session
- Bayes inference by VB & MCMC

```
?- D=[btype(a), btype(a), btype(ab), btype(o)], set_prism_flag(learn_mode,vb),learn(D).
#vbem-iters: 0(3) (Converged: -6.604)
Statistics on learning:
        Graph size: 18
        …
        Final variational free energy: -6.604

?- D=[btype(a), btype(a), btype(ab), btype(o)],
marg_mcmc_full(D,[burn_in(1000),end(10000),skip(5)],[VFE,ELM]), marg_exact(D,LogM)
VFE = -6.604
ELM = -6.424
LogM = -6.479

?- D=[btype(a), btype(a), btype(ab),btype(o)], predict_mcmc_full(D,[btype(a)],[[_,E,_]]),
print_graph(E,[lr('<=')])
btype(a)  <= gtype(a,a)
gtype(a,a)  <= msw(gene,a) & msw(gene,a)
```

Sample session

- Rank learning of {o≻a, o≻b, b≻ab}

```
?- show_sw.
Switch gene: unfixed: a (p: 0.5) b (p: 0.2) o (p: 0.3)

?- D=[[btype(o),btype(a)],[btype(o),btype(b)],[btype(b),btype(ab)]],rank_learn(D).
#sgd-iters: 0.........100.......(10000) (Stopped: 0.105)
Graph size: 36
        Number of switches: 1
        Number of switch instances: 3
        Number of iterations: 10000
        Final log likelihood: 0.191

?- show_sw.
Switch gene: unfixed:  a (p: 0.112) b (p: 0.131) o (p: 0.757)

?- rank([btype(ab),btype(o)],X).
X = [btype(o),btype(ab)]
```

Sample session

- Conditional random field (CRF): P(Ge|P)

```
btype(P):- btype(P,_).
btype(P,Ge):- Ge=[X,Y],
   gtype(X,Y),
   ( X=Y -> P=X ; X=o -> P=Y ; Y=o -> P=X ; P=ab ).
gtype(X,Y):- msw(gene,X),msw(gene,Y).
```

incomplete data P observed

complete data (P,Ge) observed
let's learn most likely Ge for P

```
?- get_samples(100,btype(_,_),D),crf_learn(D),crf_viterbig(btype(a,Ge)).
#crf-iters: 0#12
(7) (Converged: -92.443)
Statistics on learning:
        Graph size: 36
        Number of switches: 1
        Number of switch instances: 3
        Number of iterations: 7
        Final log likelihood: -92.443
Ge = [a,a]
```

$$Ge = \mathrm{argmax}_Z P(\mathrm{btype}(a,Z) \mid \mathrm{btype}(a))$$
$$= \mathrm{argmax}_Z P(Z|Pe=a)$$

PRISM's performance

- [Turliuc+, 2016]* compared LDA in various PPLs (Stan,R(topicmodels),PRISM..)
- Stan: major imperative probabilistic modeling language based on MCMC
- Likelihood and execution time measured

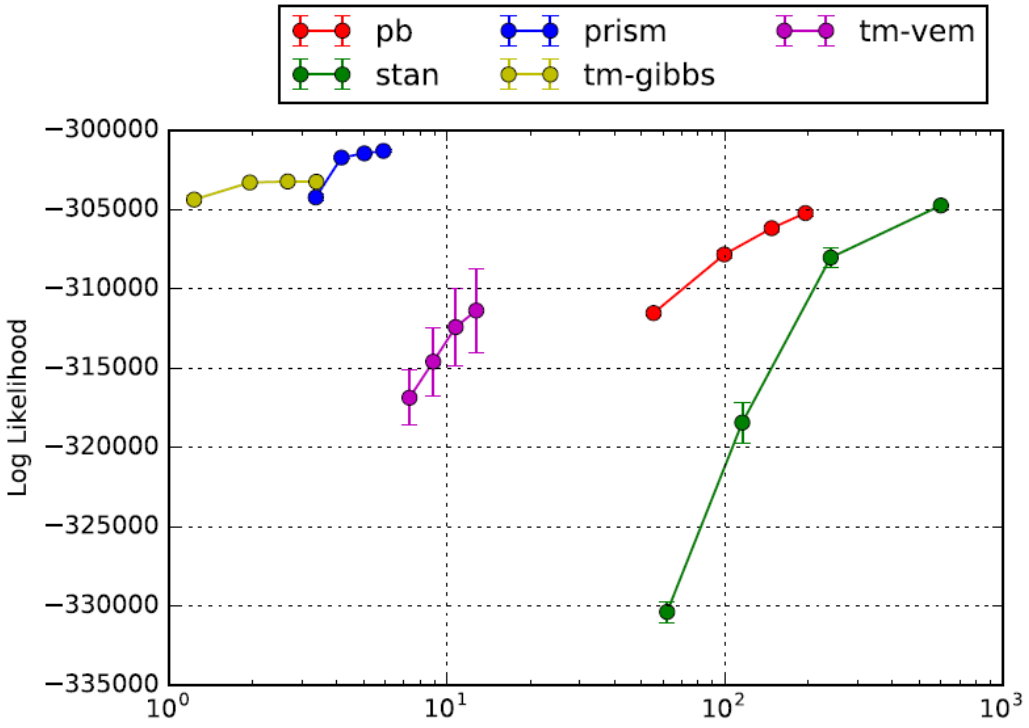


Fig. 6. Likelihood against log average execution time. Higher is better.

From [Turliuc+, 2016]*
Data: 1000 documents (100 word/document) generated by LDA with |V|=25, 10 topics

```
values(word,[1,2,..,25]).
values(topic,[1,2,..,10]).
doc(D):- (D=[W|R] ->
    msw(topic,Z),
    msw(word(Z),W),
    doc(R) ; D=[]).
```

↖ PRISM program for LDA

* Probabilistic abductive logic programming using Dirichlet priors, Turliuc et al. IJAR (78) pp.223–240, 2016

Parameter learning by comparison
for classification, parsing, link prediction

- Ranking entities (URLs,restaurants,universities…)
  - used in information retrieval, recommendation system
  - ranked by features such as TFIDF and PageRank
  - evaluated by Hit@10, MRR, nDCG,…
- Ranking propositions by a probabilistic model (new)
  - If “airplanes fly” $\succ$ “penguins fly”, learn θ s.t.

    P(“airplanes fly”|θ) > P(“penguins fly”|θ)

  - applicable to probabilistic models for structured objects

    e.g. parse trees ➔ new approach to probabilistic parsing

Rank learning in PRISM2.3

- Rank goals g by $l(g) = \log P(g|\theta)$
- Introduce re-parameterization of θ by $\boldsymbol{w}$:

$$\theta_{i,v} = \frac{\exp(w_{i,v})}{\sum_{v'} \exp(w_{i,v'})}$$

- Learn $\boldsymbol{w}$: for a pair: $g_1 \succ g_2$, minimize hinge-loss $f(g_1, g_2)$ where

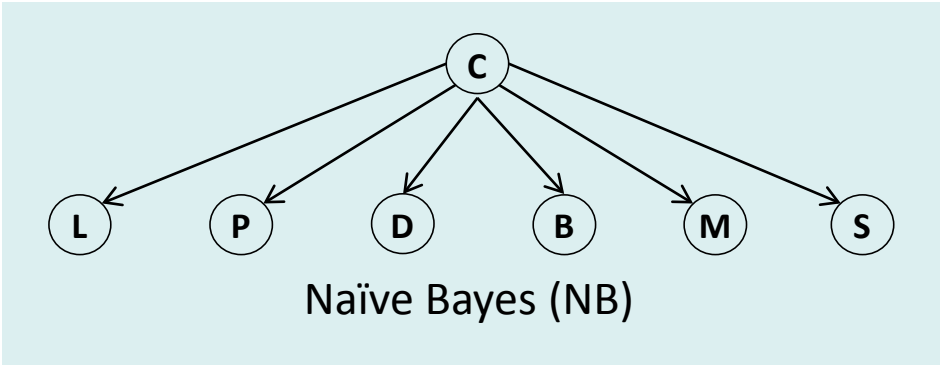
$$\begin{aligned} f(g_1, g_2) &= \max(0, z) \\ z &= l(g_2) - l(g_1) + c \end{aligned}$$

by SGD using

$$\frac{\partial \theta_{i,v}}{\partial w_{i',v'}} = \begin{cases} \theta_{i,v}(1 - \theta_{i,v}) & (i = i', v = v') \\ -\theta_{i,v}\theta_{i,v'} & (i = i', v \neq v') \\ 0 & (i \neq i') \end{cases}$$

- Learning rate automatically adjusted by {AdaDelta, Adam}

Naïve Bayes: UCI-car data



Naïve Bayes (NB)

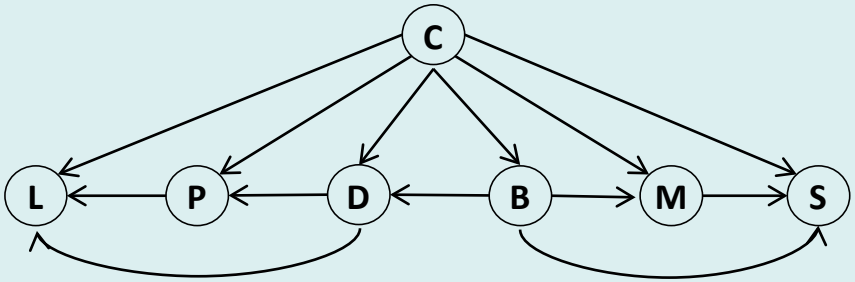
- size = 1728
- #class = 4
- #attrs = 6
- no missing value

```
values(class,[unacc,acc,good,vgood]).
values(attr(buying,_),[vhigh,high,med,low]).
values(attr(maint,_),[vhigh,high,med,low]).
values(attr(safety,_),[low,med,high]).
values(attr(doors,_),[2,3,4,more5]).
values(attr(persons,_),[2,4,more]).
values(attr(lug_boot,_),[small,med,big]).
```

```
nb(Attrs,C):-
    Attrs = [B,M,S,D,P,L],
    msw(class,C),
    msw(attr(buying,     [C]), B),
    msw(attr(maint,      [C]), M),
    msw(attr(safety,     [C]), S),
    msw(attr(doors,      [C]), D),
    msw(attr(persons,    [C]), P),
    msw(attr(lug_boot,   [C]), L).
```

NB program

BAN : UCI-car data



BAN (BN Augmented Naïve Bayes)

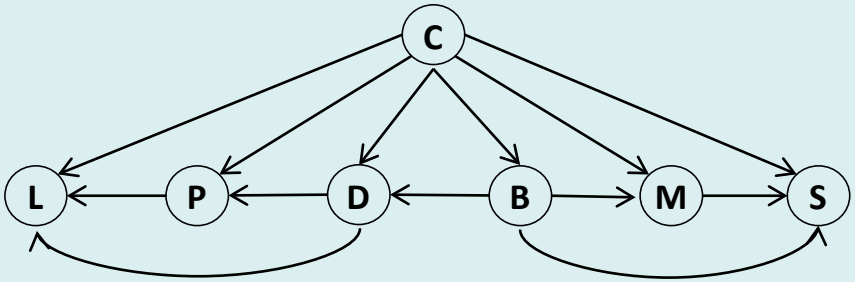
- size = 1728
- class = 4
- #attrs = 6
- no missing value

```
values(class,[unacc,acc,good,vgood]).
values(attr(buying,_),[vhigh,high,med,low]).
values(attr(maint,_),[vhigh,high,med,low]).
values(attr(safety,_),[low,med,high]).
values(attr(doors,_),[2,3,4,more5]).
values(attr(persons,_),[2,4,more]).
values(attr(lug_boot,_),[small,med,big]).
```

```
bn(Attrs,C):-
    Attrs = [B,M,S,D,P,L],
    msw(class,C),
    msw(attr(buying,    [C]),     B),
    msw(attr(maint,     [C,B]),   M),
    msw(attr(safety,    [C,B,M]), S),
    msw(attr(doors,     [C,B]),   D),
    msw(attr(persons,   [C,D]),   P),
    msw(attr(lug_boot,  [C,D,P]), L).
```

BAN program

CRF-BAN : UCI-car data



BAN (BN Augmented Naïve Bayes)

- size = 1728
- class = 4
- #attrs = 6
- no missing value

```
values(class,[unacc,acc,good,vgood]).
values(attr(buying,_),[vhigh,high,med,low]).
values(attr(maint,_),[vhigh,high,med,low]).
values(attr(safety,_),[low,med,high]).
values(attr(doors,_),[2,3,4,more5]).
values(attr(persons,_),[2,4,more]).
values(attr(lug_boot,_),[small,med,big]).
```

```
bn(Attrs,C):-
    Attrs = [B,M,S,D,P,L],
    msw(class,C),
    msw(attr(buying,    [C]),     B),
    msw(attr(maint,     [C,B]),   M),
    msw(attr(safety,    [C,B,M]), S),
    msw(attr(doors,     [C,B]),   D),
    msw(attr(persons,   [C,D]),   P),
    msw(attr(lug_boot,  [C,D,P]), L).
bn(Attrs):- bn(Attrs,_).
```

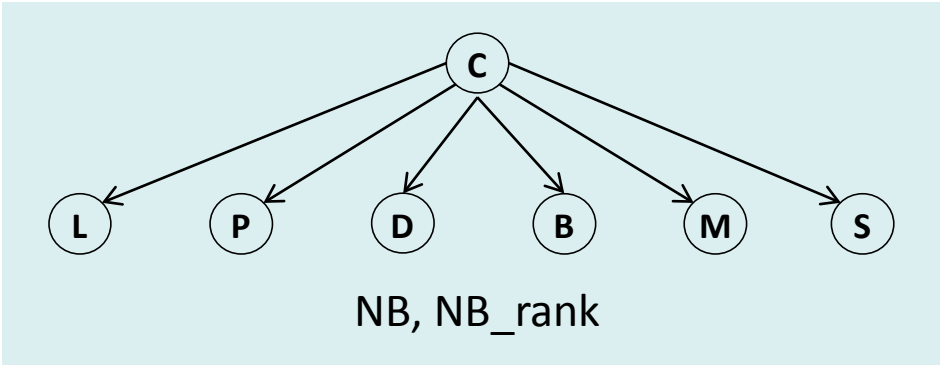
CRF-BAN program

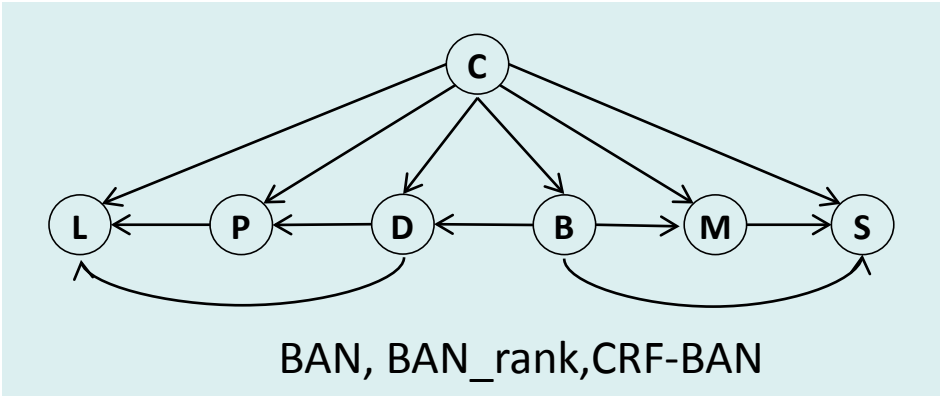
Accuracy : UCI-car data

- Task: to learn parameters and predict a class of cars
- preference pair: $nb([a_1..,a_6],c_{correct}) \succ nb([a_1..,a_6],c')$ $(c' \neq c_{correct})$

Model	Accuracy	Learning
NB	86.1%	MLE
NB_rank	88.2%	SGD
BAN	91.5%	MLE
BAN_rank	97.7%	SGD
CRF-BAN	99.8%	LBFG

Accuracy:10-fold CV
MLE: learn/1
SGD: rank_learn/1
LBFG: crf_learn/1





- Parameter learning by rank_learn/1 outperforms learn/1
- CRF-BAN gives best accuracy but crf_learn/1 takes time

Probabilistic context free grammar

PCFG = CFG + probability

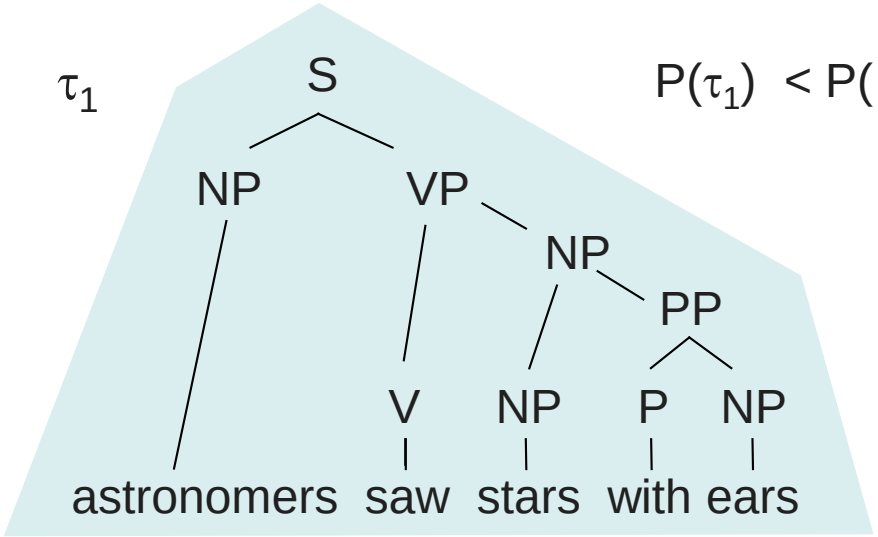
S → NP VP (1.0)
NP → NP PP (0.2) | ears (0.1) | stars (0.2) | telescopes (0.3) | astronomers (0.2)
PP → P NP (1.0)
VP → VP PP (0.4) | V NP (0.6)
V → see (0.5) | saw (0.5)
P → in (0.3) | at (0.4) | with (0.3)

S →* astronomers saw stars with ears

Viterbi parse tree

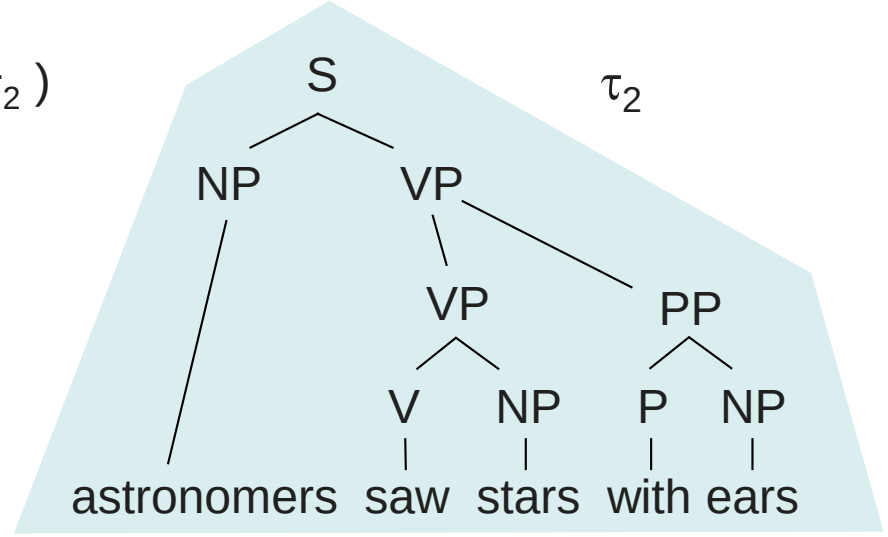
S → NP VP (1.0)
NP → NP PP (0.2) | ears (0.1) | stars (0.2)
telescopes (0.3) | astronomers (0.2)
PP → P NP (1.0)
VP → VP PP (0.4) | V NP (0.6)
V → see (0.5) | saw (0.5)
P → in (0.3) | at (0.4) | with (0.3)

τ_1



$P(\tau_1) < P(\tau_2)$

τ_2



$P(\tau_1)=1.0\times0.2\times0.6\times0.5\times0.2\times0.2\times1.0\times0.3\times0.1$
$=0.000072$

$P(\tau_2)=1.0\times0.2\times0.4\times0.6\times0.5\times0.2\times1.0\times0.3\times0.1$
$=0.000144$

PCFG program

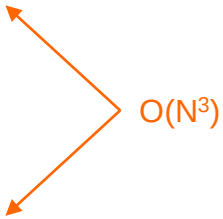
```
values(s,[[np,vp]],set@[1.0]).
values(np,[[np,pp],[ears], [stars],
        [telescope], [astronomars]],
        set@[0.2, 0.1, 0.2, 0.3, 0.2]).
values(pp,[[p,np]],set@[1.0]).
values(vp,[[vp,pp],[v,np]],set@[0.4,0.6]).
values(v,[see,saw],set@0.5,0.5]).
values(p,[in,at,with],set@[0.3,0.4,0.3]).
```

```
pcfg(L):- C=s,pcfg([C],L,[]).
pcfg([A|R],L0,L2):-
   ( get_values(A,_) ->
       msw(A,RHS), pdcg(RHS,L0,L1)
   ; L0=[A|L1] ),
   pdc(R,L1,L2).
pdcg([],L,L).
```

common to all grammars

```
?- viterbi(pdcg([astronomars,saw,stars,with,ears])).
Viterbi_P = 0.000144
```

```
?- viterbif(pdcg([astronomars,saw,stars,with,ears])).
pdcg([astronomars,saw,stars,with,ears])
   <= pdcg([s],[astronomars,saw,stars,with,ears],[])
…
pdcg([ears],[ears],[])
   <= pdcg([],[],[])
pdcg([],[],[])
```



Rank learning of parse trees

- We use ATR corpus (10,996 parse trees) and a PCFG (861 rules)
- Task: to make Viterbi parse trees to coincide with the corpus trees by learning good parameters θ ($|\theta| = 842$)
  1. For sentence s, put $\tau_{ATR}(s)$ = parse tree of s in the corpus &
     $\tau_{V1}(s)$ = 1st Viterbi parse tree by $P_0 = \mathrm{argmax}_\tau P_0(\tau \mid s)$ &
     $\tau_{V2}(s)$ = 2nd Viterbi parse tree by P_0
  2. Put $\tau_V(s) = \tau_{V2}(s)$ if $P_0(\tau_{ATR}(s)) \geq P_0(\tau_{V1}(s))$, $\tau_V(s) = \tau_{V1}(s)$ o.w.
  3. Learn θ from preference pairs ($\tau_{ATR}(s) \succ \tau_V(s)$) by rank_learn/1 (SGD)

- This approach yields better parameters than MLE (10-fold CV)

model	learning	TreeAccuracies (std.)
PCFG	learning to rank, P_0 =uniform	80.66% (1.10)
PCFG	learning to rank, P_0 =random	80.58% (0.81)
PCFG	learning to rank, P_0 =MLE	80.49% (0.99)
PCFG	MLE	78.87% (1.13)
PCFG	no-learning, P_0 =uniform	72.59% (1.27)
CRF-CFG	discriminative learning	81.93% (1.03)

rank learning outperforms MLE

baseline

Learning sentence preference

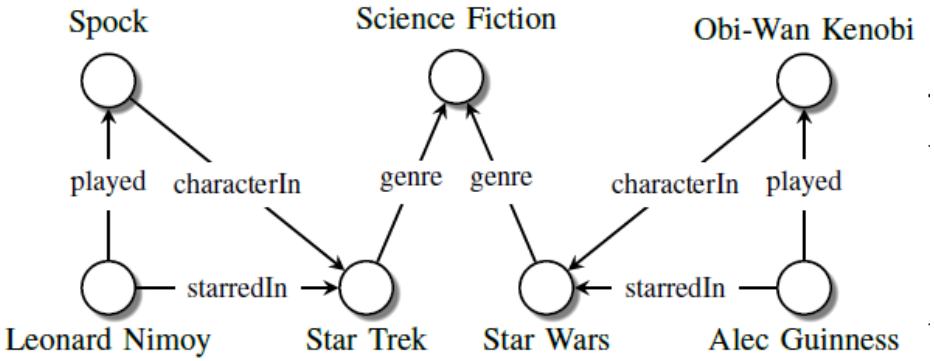
- We prefer sentences w.o. interjections to ones with them : $(s_{\text{no\_interj}} \succ s_{\text{interj}})$
- Task: learn θ so that $P(s_{\text{no\_interj}}|\theta) > P(s_{\text{interj}}|\theta)$ holds
  1. sample $Size$-sentences w.o. interjections and those with them respectively from PCFG(θ_0) where θ_0 denotes parameters for P_0
  2. construct preference pairs D= { $[s_{\text{no\_interj}}, s_{\text{interj}}]$ } ($|D| = Size^2$)
  3. estimate the ratio of "$P(s_{\text{no\_interj}}|\theta) > P(s_{\text{interj}}|\theta)$" in D using θ learned by ?-rank_learn(D) using10-fold CV

P_0	$Size = 50$	$Size = 100$	$Size = 150$
uniform	62.04%(4.99)	63.87%(1.14)	59.59%(1.14)
random	61.04%(2.16)	57.68%(1.58)	59.33%(0.80)
no-learning	34.11%(1.02)	34.11%(1.02)	34.11%(1.02)

- PCFG(θ) now produces $s_{\text{no\_interj}}$ much more frequently than PCFG(θ_0)

Knowledge graph

- Big knowledge graphs (KGs) available
  (FreeBase,Wikidata,Knowledge Vault,..)
- Set of triplets (*s*,*rel*,*o*) (= proposition *rel*(*s*,*o*))
- Q.A. on (*s*,*rel*,*o*) ：Who played Spock?



From: A Review of Relational Machine Learning for Knowledge Graphs
Maximilian Nickel, Kevin Murphy, Volker Tresp, Evgeniy Gabrilovich,
Proceedings of the IEEE 104(1): 11-33 (2016)

Embedding

- Embedding in a low-dimensional space for scalability
  - entities: *s*,*o* ➔ K-dimension vector **s**,**o** (K ≪ |entities|)
  - relation: *rel* ➔ K-dimension vector **rel**, K x K matrix **D**
- Translation model
  - TransE [Bordes+,13],..,HOLE [Nickel+,16]
  - *rel*(*s*,*o*) ≈ true ⇔ ‖**s** + **rel** - **o**‖ ≈ 0
- Bi-linear model
  - RESCAL [Nickel,13],..,DistMult [Yang+,15], [Trouillon+,16]
  - *rel*(*s*,*o*) ≈ true ⇔ $\mathbf{s}^{\mathsf{T}}\mathbf{Do}$ = **(s•Do)** ≈ 1 (**D** is diagonal)
- PRISM-DistMult model (entity vector = prob. distribution)

```
values(cluster(_),[1-K]):- K=50.
values(rel(_),[true,false]).

rel(S,O,V):-  msw(cluster(S),C),msw(cluster(O),C),msw(rel(C),V).
```

Connecting logic to vector spaces

rel(s,o,true) ⇔
　　∃c msw(cluster(s),c) & msw(cluster(o),c) & msw(rel(c),true))

s = (P(msw(cluster(s),1)),..,P(msw(cluster(s),K)))^T
o = (P(msw(cluster(o),1)),..,P(msw(cluster(o),K)))^T
D[i,j] = P(msw(rel(i),true))) if i=j, =0 o.w.

P(rel(s,o,true))
= P(∃c msw(cluster(s),c) & msw(cluster(o),c) & msw(rel(c),true)))
= $\sum_{\{c \text{ in } [1..K]\}}$ P(msw(cluster(s),c) & msw(cluster(o),c) & msw(rel(c),true))
= $\sum_{\{c \text{ in } [1..K]\}}$ P(msw(cluster(s),c))P(msw(cluster(o),c)) P(msw(rel(c),true))
= **s**[1]**D**[1,1]**o**[1]+…+ **s**[K]**D**[K,K]**o**[K]
= **s**^{T}**D**o = **(s•Do)**

Rank learning of triplets for classification

Rank learning requires negative data!

- We learn θ so that $P((s,rel,o)|\theta) > P(rel(s', rel,o')|\theta)$ holds for pairs $((s,rel,o) \succ (s',rel,o'))$ of positive (s,rel,o) and negative (s',rel,o')
- $(s,rel,o) \in KG$ is considered positive $\succ$ other triplets considered negative
- We need (positive $\succ$ negative) pairs for rank learning but KG only contains positive data
- Create negative data following [Socher+,13]
  - Given (s,rel,o) in KG, randomly sample entities s', o' such that (s',rel,o), (s,rel,o') not in KG (🡐 over-simplified)
  - Use pairs $((s, rel,o) \succ rel(s', rel,o))$, $(rel(s, rel,o) \succ rel(s, rel,o'))$ as training data for rank learning
- (positive $\succ$ negative) pairs are also used in the testing phase

Triplet classification: FB15k data

- FB15k[Bordes+,13]: subset of FB, 14,951 entities and 1,345 relations
- Top-10 largest relations {rel_0,..,rel_9} used in the experiment

Data set	rel_0	rel_1	rel_2	rel_3	rel_4	rel_5	rel_6	rel_7	rel_8	rel_9
training	15998	12893	12166	12175	11547	11636	9466	9494	9439	9465
valid	1706	1353	1304	1191	1195	1200	1049	976	965	969
test	2060	1591	1451	1555	1478	1384	1123	1168	1190	1160

- Task: learn parameters for PRISM-DistMult model, classify triplets in {rel_0,..,rel_9} as positive/negative and evaluate accuracy over 5 runs.

	rel_0	rel_1	rel_2	rel_3	rel_4	rel_5	rel_6	rel_7	rel_8	rel_9
Rank learning	87.4%	79.8%	74.1%	72.7%	62.0%	62.7%	54.8%	55.0%	69.4%	69.5%
MLE	85.9%	60.4%	68.7%	72.6%	61.8%	56.1%	52.0%	50.0%	66.5%	69.7%
T-Test	=	>	=	=	=	>	=	>	=	=

- Rank learning performs better than or equally to MLE

Summary

- We reach the best model only after exhaustive trials
- PRISM2.3 offers built-in predicates for diverse learning methods to ease this laborious process
  - MLE,VT,VB,MCMC for generative modeling
  - LBFG for conditional random fields (CRFs)
  - SGD for preference modeling (🡐 new)
- They all applicable to the same program!
- Available for download at http://rjida.meijo-u.ac.jp/prism/