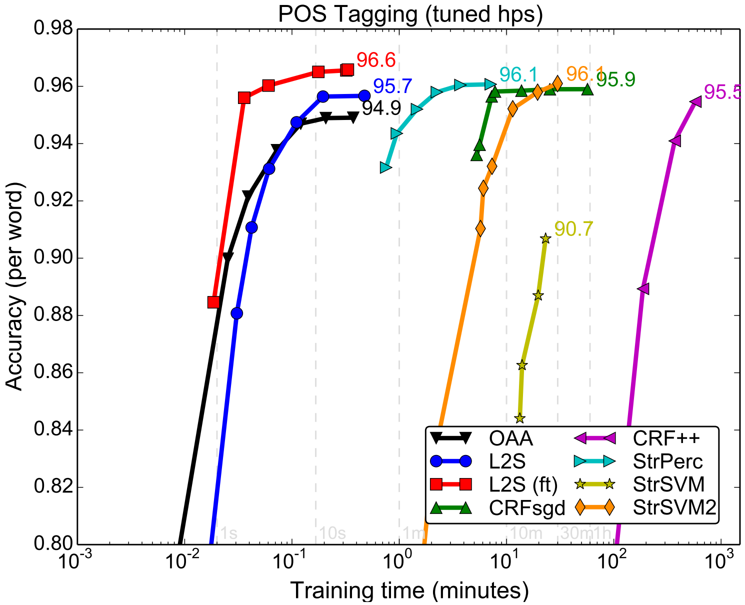
Outline

What part of speech are the words?



A demonstration

1 |w Despite
2 |w continuing
3 |w problems
1 |w in
4 |w its
5 |w newsprint
5 |w business

...

A demonstration

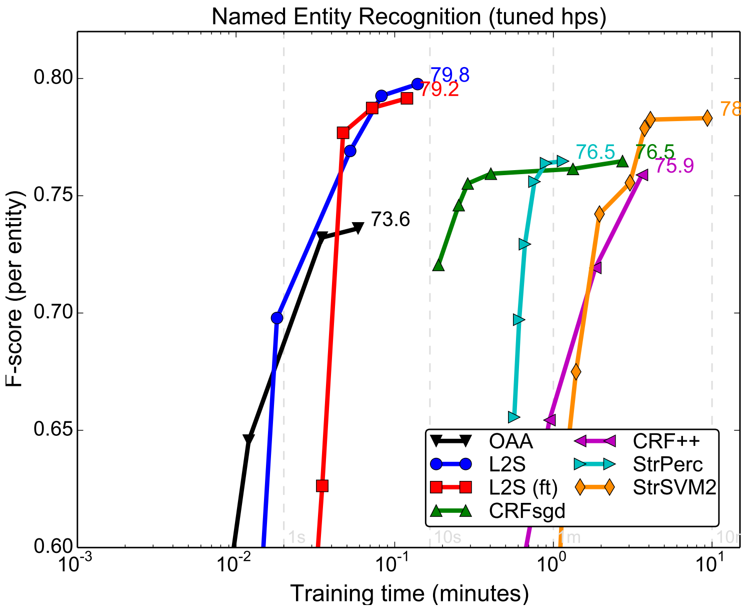
1 |w Despite
2 |w continuing
3 |w problems
1 |w in
4 |w its
5 |w newsprint
5 |w business

...

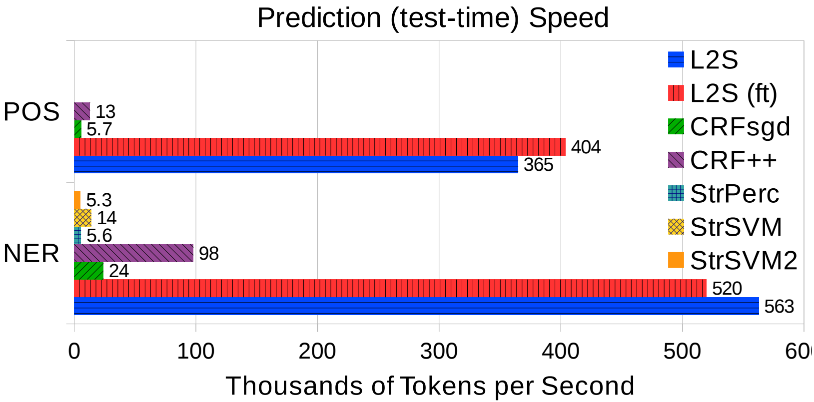
vw -b 24 -d wsj.train.vw -c –search_task sequence –search 45 –search_alpha 1e-8 –search_neighbor_features -1:w,1:w –affix -1w,+1w -f foo.reg

vw -t -i foo.reg wsj.test.vw

Is this word a name or not?



How fast in evaluation?



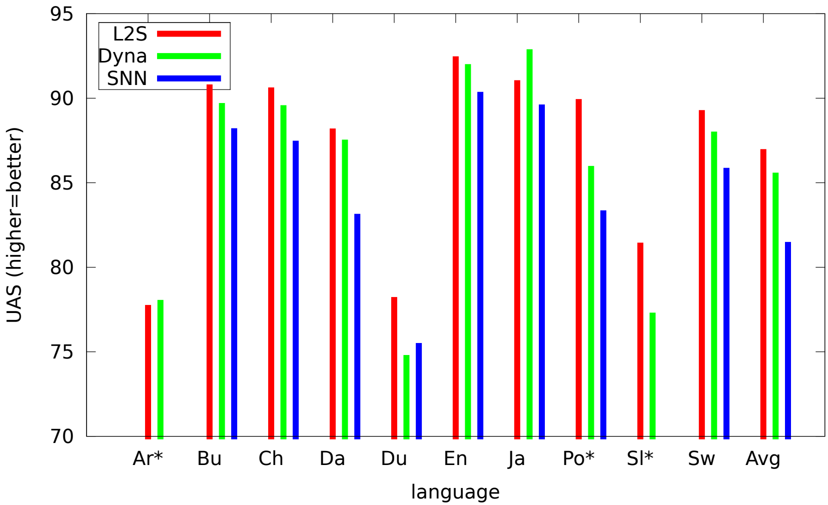
Entity Relation

Goal: find the Entities and then find their Relations

Method	Entity F1	Relation F1	Train Time
Structured SVM	88.00	50.04	300 seconds
L2S	92.51	52.03	13 seconds

L2S uses ~100 LOC.

Find dependency structure of sentences.



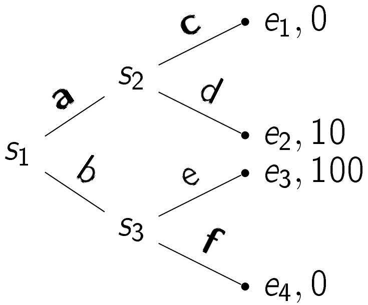
L2S uses ~300 LOC.

Outline

1. Empirics
2. Analysis
3. Programming
4. Others and Issues

Effect of Roll-in and Roll-out Policies

roll-out → / ↓ roll-in	Reference	Half-n-half	Learned
Reference	Inconsistent		
Learned	.		



Effect of Roll-in and Roll-out Policies

roll-out → / ↓ roll-in	Reference	Half-n-half	Learned
Reference	Inconsistent		
Learned	.		

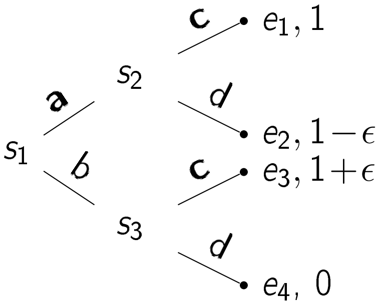
Theorem

Roll-in with ref:

0 *cost-sensitive regret* ⇒ *unbounded joint regret*

Effect of Roll-in and Roll-out Policies

roll-out → ↓ roll-in	Reference	Half-n-half	Learned
Reference	Inconsistent		
Learned	Consistent No local opt		



Effect of Roll-in and Roll-out Policies

roll-out → ↓ roll-in	Reference	Half-n-half	Learned
Reference	Inconsistent		
Learned	Consistent No local opt		

Theorem

Roll-out with Ref:

0 *cost-sensitive regret* ⇒ 0 *joint regret*

(but not local optimality)

Effect of Roll-in and Roll-out Policies

roll-out → / ↓ roll-in	Reference	Half-n-half	Learned
Reference	Inconsistent		
Learned	Consistent No local opt		Reinf. L.

Theorem

Ignore Ref:
$\Rightarrow$ *Equivalent to reinforcement learning.*

Effect of Roll-in and Roll-out Policies

roll-out → ↓ roll-in	Reference	Half-n-half	Learned
Reference	Inconsistent		
Learned	Consistent No local opt	Consistent Local Opt	Reinf. L.

Theorem

Roll-out with $p = 0.5$ Ref and $p = 0.5$ Learned: 0 cost-sensitive regret $\Rightarrow$ 0 joint regret + locally optimal

See LOLS paper, Wednesday 11:20 Van Gogh

AggreVaTe Regret Decomposition

π^{ref} = reference policy
$\bar{\pi}$ = stochastic average learned policy
$J(\pi)$ = expected loss of π.

Theorem

$J(\bar{\pi}) - J(\pi^{ref}) \leq$

AggreVaTe Regret Decomposition

π^{ref} = reference policy
$\bar{\pi}$ = stochastic average learned policy
$J(\pi)$ = expected loss of π.

Theorem

$$J(\bar{\pi}) - J(\pi^{ref}) \leq T\mathbb{E}_{n,t}\mathbb{E}_{x\sim D^t_{\hat{\pi}_n}}\left[Q^{\pi^{ref}}(x, \hat{\pi}_n) - Q^{\pi^{ref}}(x, \pi^{ref})\right]$$

T = number of steps
$\hat{\pi}_n$ = nth learned policy
$D^t_{\hat{\pi}_n}$ = distribution over x at time t induced by $\hat{\pi}_n$
$Q^{\pi}(x, \pi')$ = loss of π' at x then π to finish

Proof

For all π let π^t play π for rounds $1 \ldots t$ then play π^{ref} for rounds $t+1 \ldots T$. So $\pi^T = \pi$ and $\pi^0 = \pi^{\text{ref}}$

Proof

For all π let π^t play π for rounds $1 \ldots t$ then play π^{ref} for rounds $t+1 \ldots T$. So $\pi^T = \pi$ and $\pi^0 = \pi^{\text{ref}}$

$J(\pi) - J(\pi^{\text{ref}})$

$= \sum_{t=1}^{T} J(\pi^t) - J(\pi^{t-1})$ (Telescoping sum)

Proof

For all π let π^t play π for rounds $1 \ldots t$ then play π^{ref} for rounds $t+1 \ldots T$. So $\pi^T = \pi$ and $\pi^0 = \pi^{\text{ref}}$

$$J(\pi) - J(\pi^{\text{ref}})$$
$$= \sum_{t=1}^{T} J(\pi^t) - J(\pi^{t-1}) \text{ (Telescoping sum)}$$
$$= \sum_{t=1}^{T} \mathbb{E}_{x \sim D_\pi^t} \left[Q^{\pi^{\text{ref}}}(x, \pi) - Q^{\pi^{\text{ref}}}(x, \pi^{\text{ref}}) \right]$$

since for all π, t, $J(\pi) = \mathbb{E}_{x \sim D_\pi^t} Q^{\pi}(x, \pi)$

Proof

For all π let π^t play π for rounds $1 \ldots t$ then play π^{ref} for rounds $t+1 \ldots T$. So $\pi^T = \pi$ and $\pi^0 = \pi^{\text{ref}}$

$$J(\pi) - J(\pi^{\text{ref}})$$
$$= \sum_{t=1}^{T} J(\pi^t) - J(\pi^{t-1}) \text{ (Telescoping sum)}$$
$$= \sum_{t=1}^{T} \mathbb{E}_{x \sim D_\pi^t} \left[Q^{\pi^{\text{ref}}}(x, \pi) - Q^{\pi^{\text{ref}}}(x, \pi^{\text{ref}}) \right]$$

since for all $\pi, t,\ J(\pi) = \mathbb{E}_{x \sim D_\pi^t} Q^{\pi}(x, \pi)$

$$= T \mathbb{E}_t \mathbb{E}_{x \sim D_\pi^t} \left[Q^{\pi^{\text{ref}}}(x, \pi) - Q^{\pi^{\text{ref}}}(x, \pi^{\text{ref}}) \right]$$

Proof

For all π let π^t play π for rounds $1 \dots t$ then play π^{ref} for rounds $t+1 \dots T$. So $\pi^T = \pi$ and $\pi^0 = \pi^{\text{ref}}$

$$J(\pi) - J(\pi^{\text{ref}})$$
$$= \sum_{t=1}^{T} J(\pi^t) - J(\pi^{t-1}) \text{ (Telescoping sum)}$$
$$= \sum_{t=1}^{T} \mathbb{E}_{x \sim D_\pi^t} \left[Q^{\pi^{\text{ref}}}(x, \pi) - Q^{\pi^{\text{ref}}}(x, \pi^{\text{ref}}) \right]$$

since for all π, t, $J(\pi) = \mathbb{E}_{x \sim D_\pi^t} Q^\pi(x, \pi)$

$$= T \mathbb{E}_t \mathbb{E}_{x \sim D_\pi^t} \left[Q^{\pi^{\text{ref}}}(x, \pi) - Q^{\pi^{\text{ref}}}(x, \pi^{\text{ref}}) \right]$$

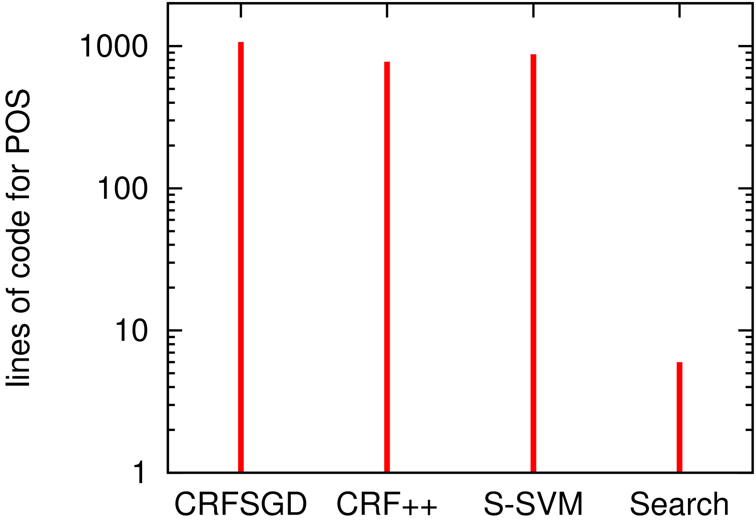
So $J(\bar{\pi}) - J(\pi^{\text{ref}})$

$$= T \mathbb{E}_{t,n} \mathbb{E}_{x \sim D_{\hat{\pi}_n}^t} \left[Q^{\pi^{\text{ref}}}(x, \hat{\pi}_n) - Q^{\pi^{\text{ref}}}(x, \pi^{\text{ref}}) \right]$$

Outline

1. Empirics
2. Analysis
3. Programming
4. Others and Issues

Lines of Code



How?

Sequential_RUN(*examples*)

for $i = 1$ **to** len(*examples*) **do**
    prediction $\leftarrow$ **predict**(*examples*[*i*], *examples*[*i*].*label*)
    loss(*prediction* $\neq$ *examples*[*i*].*label*)
end for

How?

Sequential_RUN(*examples*)

```
for i = 1 to len(examples) do
    prediction ← predict(examples[i], examples[i].label)
    loss(prediction ≠ examples[i].label)
end for
```

Decoder + loss + reference advice

RunParser(*sentence*)

```
stack S ← {Root}
buffer B ← [words in sentence]
arcs A ← ∅
while B ≠ ∅ or |S| > 1 do
    ValidActs ← GetValidActions(S, B)
    features ← GetFeat(S, B, A)
    ref ← GetGoldAction(S, B)
    action ← predict(features, ref, ValidActs)
    S, B, A ← Transition(S, B, A, action)
end while
loss(A[w] ≠ A*[w], ∀w ∈ sentence)
return output
```

Program/Search equivalence

Theorem: Every algorithm which:

1. Always terminates.
2. Takes as input relevant feature information X.
3. Make 0+ calls to predict.
4. Reports loss on termination.

defines a search space, and such an algorithm exists for every search space.

It even works in Python

```
def _run(self, sentence):
    output = []
    for n in range(len(sentence)):
        pos,word = sentence[n]
        with self.vw.example('w': [word],
                  'p': [prev_word]) as ex:
            pred = self.sch.predict(examples=ex,
                        my_tag=n+1, oracle=pos,
                        condition=[(n,'p'), (n-1, 'q')])
            output.append(pred)
    return output
```

Bugs you cannot have

1. Never train/test mismatch.

Bugs you cannot have

1. Never train/test mismatch.
2. Never unexplained slow.

Bugs you cannot have

1. Never train/test mismatch.
2. Never unexplained slow.
3. Never fail to compensate for cascading failure.

Outline

1. Empirics
2. Analysis
3. Programming
4. Others and Issues
    1. Families of algorithms.
    2. What's missing from learning to search?

Imitation Learning

Use perceptron-like update when learned deviates from gold standard.

Inc. P.	Collins & Roark, ACL 2004.
LaSo	Daume III & Marcu, ICML 2005.
Local	Liang et al, ACL 2006.
Beam P.	Xu et al., JMLR 2009.
Inexact	Huang et al, NAACL 2012.

Imitation Learning

Use perceptron-like update when learned deviates from gold standard.

Inc. P. Collins & Roark, ACL 2004.

LaSo Daume III & Marcu, ICML 2005.

Local Liang et al, ACL 2006.

Beam P. Xu et al., JMLR 2009.

Inexact Huang et al, NAACL 2012.

Train a classifier to mimic an expert's behavior

DAgger Ross et al., AIStats 2011.

Dyna O Goldberg et al., TACL 2014.

Learning to Search

When the reference policy is optimal

Searn Daume III et al., MLJ 2009.

Aggra Ross & Bagnell, `http://arxiv.org/pdf/1406.5979`

Learning to Search

When the reference policy is optimal

Searn Daume III et al., MLJ 2009.

Aggra Ross & Bagnell, `http://arxiv.org/pdf/1406.5979`

When it's not

LOLS Chang et al., ICML 2015.

Learning to Search

When the reference policy is optimal

- **Searn** Daume III et al., MLJ 2009.
- **Aggra** Ross & Bagnell, `http://arxiv.org/pdf/1406.5979`

When it's not

- **LOLS** Chang et al., ICML 2015.

Code in Vowpal Wabbit `http://hunch.net/~vw`

Inverse Reinforcement Learning

Given observed expert behavior, infer the underlying reward function the expert seems to be optimizing

Inverse Reinforcement Learning

Given observed expert behavior, infer the underlying reward function the expert seems to be optimizing

propose Kalman, 1968.

1st sol. Boyd, 1994.

Inverse Reinforcement Learning

Given observed expert behavior, infer the underlying reward function the expert seems to be optimizing

propose Kalman, 1968.

1st sol. Boyd, 1994.

from sample trajectories only
Ng & Russell, ICML 2000

Inverse Reinforcement Learning

Given observed expert behavior, infer the underlying reward function the expert seems to be optimizing

- **propose** Kalman, 1968.
- **1st sol.** Boyd, 1994.

from sample trajectories only
Ng & Russell, ICML 2000

for apprenticeship learning

- **Apprent.** Abbeel & Ng, ICML 2004
- **Maxmar.** Ratliff et al., NIPS 2005
- **MaxEnt** Ziebart et al., AAAI 2008

What's missing? Automatic Search order

Learning to search $\simeq$ dependency + search order.
Graphical models "work" given dependencies only.

What's missing? The reference policy

A good reference policy is often nonobvious... yet critical to performance.

What's missing?
Efficient Cost-Sensitive Learning

When choosing 1-of-k things, $O(k)$ time is not exciting for machine translation.

What's missing? GPU fun

Vision often requires a GPU. Can that be done?

How to optimize discrete joint loss?

How to optimize discrete joint loss?

1. Programming complexity.

How to optimize discrete joint loss?

1. Programming complexity. Most complex problems addressed independently—too complex to do otherwise.

How to optimize discrete joint loss?

1. Programming complexity. Most complex problems addressed independently—too complex to do otherwise.
2. Prediction accuracy. It had better work well.

How to optimize discrete joint loss?

1. Programming complexity. Most complex problems addressed independently—too complex to do otherwise.
2. Prediction accuracy. It had better work well.
3. Train speed. Debug/development productivity + maximum data input.

How to optimize discrete joint loss?

1. Programming complexity. Most complex problems addressed independently—too complex to do otherwise.
2. Prediction accuracy. It had better work well.
3. Train speed. Debug/development productivity + maximum data input.
4. Test speed. Application efficiency