Outline

1. Empirics
2. Analysis
3. Programming
4. Others and Issues
What part of speech are the words?

POS Tagging (tuned hps)

Accuracy (per word)

Training time (minutes)

POS Tagging (tuned hps)
OAA
L2S
L2S (ft)
CRFsgd
CRF++
StrPerc
StrSVM
StrSVM2
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?

**Named Entity Recognition (tuned hps)**

- OAA
- L2S
- L2S (ft)
- CRFsgd
- CRF++
- StrPerc
- StrSVM2

**Training time (minutes)**
- 0.60
- 0.65
- 0.70
- 0.75
- 0.80

**F-score (per entity)**
- 73.6
- 79.8
- 79.2
- 76.5
- 75.9
- 76.5
- 78.3

**Units**
- 1s
- 10s
- 1m
- 10m
How fast in evaluation?

Prediction (test-time) Speed

- **POS**
  - L2S: 404
  - L2S (ft): 365
  - CRFsgd: 13
  - CRF++: 5.7
  - StrPerc: 98
  - StrSVM: 14
  - StrSVM2: 5.6

- **NER**
  - L2S: 520
  - L2S (ft): 563
  - CRFsgd: 24
  - CRF++: 5.3
  - StrPerc: 98
  - StrSVM: 14
  - StrSVM2: 5.6

Thousands of Tokens per Second
Goal: find the Entities and then find their Relations

<table>
<thead>
<tr>
<th>Method</th>
<th>Entity F1</th>
<th>Relation F1</th>
<th>Train Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structured SVM</td>
<td>88.00</td>
<td>50.04</td>
<td>300 seconds</td>
</tr>
<tr>
<td>L2S</td>
<td>92.51</td>
<td>52.03</td>
<td>13 seconds</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>roll-out →</th>
<th>Reference</th>
<th>Half-n-half</th>
<th>Learned</th>
</tr>
</thead>
<tbody>
<tr>
<td>↓ roll-in</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reference</td>
<td></td>
<td>Inconsistent</td>
<td></td>
</tr>
<tr>
<td>Learned</td>
<td>.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diagram:
```
    c → e1, 0
   /   
  a     d
 /     /
|      |
s2     e2, 10
/     /
|      |
s1     e
/     /
|      |
s3     e3, 100
     /  
    f   e4, 0
```
## Effect of Roll-in and Roll-out Policies

<table>
<thead>
<tr>
<th></th>
<th>Reference</th>
<th>Half-n-half</th>
<th>Learned</th>
</tr>
</thead>
<tbody>
<tr>
<td>roll-out →</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>↓ roll-in</td>
<td>Reference</td>
<td>Inconsistent</td>
<td></td>
</tr>
<tr>
<td>Reference</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Learned</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Theorem

*Roll-in with ref:*

\[ 0 \text{ cost-sensitive regret} \Rightarrow \text{unbounded joint regret} \]
## Effect of Roll-in and Roll-out Policies

<table>
<thead>
<tr>
<th>roll-out →</th>
<th>Reference</th>
<th>Half-n-half</th>
<th>Learned</th>
</tr>
</thead>
<tbody>
<tr>
<td>↓ roll-in</td>
<td>Inconsistent</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reference</td>
<td>Consistent</td>
<td>No local opt</td>
<td></td>
</tr>
<tr>
<td>Learned</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Diagram]

- **Node a** connects to **node b** with probability $s_1$.
- **Node b** connects to **nodes c and d** with probabilities $s_2$ and $s_3$, respectively.

- **Node c** connects to **node e_1** with probability $1$.
- **Node d** connects to **node e_2** with probability $1 - \epsilon$ and **node e_4** with probability $0$.
- **Node e_3** connects to **node e_2** with probability $1 + \epsilon$.
Effect of Roll-in and Roll-out Policies

<table>
<thead>
<tr>
<th>roll-out →</th>
<th>Reference</th>
<th>Half-n-half</th>
<th>Learned</th>
</tr>
</thead>
<tbody>
<tr>
<td>↓ roll-in</td>
<td>Inconsistent</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reference</td>
<td></td>
<td>Inconsistent</td>
<td></td>
</tr>
<tr>
<td>Learned</td>
<td>Consistent</td>
<td></td>
<td>No local opt</td>
</tr>
</tbody>
</table>

**Theorem**

*Roll-out with Ref:*

0 cost-sensitive regret $\Rightarrow$ 0 joint regret
(but not local optimality)
### Effect of Roll-in and Roll-out Policies

<table>
<thead>
<tr>
<th>roll-out →</th>
<th>Reference</th>
<th>Half-n-half</th>
<th>Learned</th>
</tr>
</thead>
<tbody>
<tr>
<td>↓ roll-in</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reference</td>
<td></td>
<td>Inconsistent</td>
<td></td>
</tr>
<tr>
<td>Learned</td>
<td>Consistent</td>
<td>No local opt</td>
<td>Reinf. L.</td>
</tr>
</tbody>
</table>

**Theorem**

*Ignore Ref:*

⇒ *Equivalent to reinforcement learning.*
**Effect of Roll-in and Roll-out Policies**

<table>
<thead>
<tr>
<th>roll-out →</th>
<th>Reference</th>
<th>Half-n-half</th>
<th>Learned</th>
</tr>
</thead>
<tbody>
<tr>
<td>↓ roll-in</td>
<td>Inconsistent</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reference</td>
<td>Consistent</td>
<td>Consistent</td>
<td>Reinf. L.</td>
</tr>
<tr>
<td>Learned</td>
<td>No local opt</td>
<td>Local Opt</td>
<td></td>
</tr>
</tbody>
</table>

**Theorem**

Roll-out with $p = 0.5$ Ref and $p = 0.5$ Learned: 0 cost-sensitive regret $\Rightarrow$ 0 joint regret + locally optimal
$\pi^\text{ref}$ = reference policy
$\bar{\pi}$ = stochastic average learned policy
$J(\pi) = \text{expected loss of } \pi$.

**Theorem**

$J(\bar{\pi}) - J(\pi^\text{ref}) \leq \frac{1}{T} \sum_{t=1}^{T} E_{D_t \pi_n} [Q_{\pi^\text{ref}}(x, \hat{\pi}_n) - Q_{\pi^\text{ref}}(x, \pi^\text{ref})]$ 
$T$ = number of steps
$\hat{\pi}_n$ = $n^{th}$ learned policy
$D_t \hat{\pi}_n$ = distribution over $x$ at time $t$ induced by $\hat{\pi}_n$
$Q_{\pi}(x, \pi') = \text{loss of } \pi' \text{ at } x \text{ then } \pi$ to finish
**AggreVaTe Regret Decomposition**

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

**Theorem**

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

\( T \) = 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 \( \pi' \) at \( x \) then \( \pi \) to finish
Proof

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

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

\[
J(\pi) - J(\pi^{ref}) = \sum_{t=1}^{T} J(\pi^t) - J(\pi^{t-1}) \quad \text{(Telescoping sum)}
\]
Proof

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

- CRFSGD
- CRF++
- S-SVM
- Search

Lines of code for POS
Sequential\_RUN(\textit{examples})

1: \textbf{for} $i = 1$ \textbf{to} \textit{len(examples)} \textbf{do}
2: \hspace{1em} \textit{prediction} $\leftarrow$ \textbf{predict}(\textit{examples}[\textit{i}], \textit{examples}[\textit{i}].\textit{label})
3: \hspace{1em} \textbf{loss}(\textit{prediction} \neq \textit{examples}[\textit{i}].\textit{label})
4: \textbf{end for}
How?

Sequential_RUN($\text{examples}$)

1: \textbf{for} $i = 1$ \textbf{to} \text{len($\text{examples}$)} \textbf{do}
2: \hspace{1em} $\text{prediction} \leftarrow \text{predict}(\text{examples}[i], \text{examples}[i].\text{label})$
3: \hspace{1em} $\text{loss}(\text{prediction} \neq \text{examples}[i].\text{label})$
4: \hspace{1em} \textbf{end for}

Decoder + loss + reference advice
RunParser(*sentence*)

1: stack $S \leftarrow \{\text{Root}\}$  
2: buffer $B \leftarrow \text{[words in sentence]}$  
3: arcs $A \leftarrow \emptyset$  
4: while $B \neq \emptyset$ or $|S| > 1$ do  
5:    ValidActs $\leftarrow$ GetValidActions($S$, $B$)  
6:    features $\leftarrow$ GetFeat($S$, $B$, $A$)  
7:    ref $\leftarrow$ GetGoldAction($S$, $B$)  
8:    action $\leftarrow$ predict(features, ref, ValidActs)  
9:    $S$, $B$, $A \leftarrow$ Transition($S$, $B$, $A$, action)  
10: end while  
11: loss($A[w] \neq A^*[w]$, $\forall w \in \text{sentence}$)  
12: return output
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.
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.
1 Empirics
2 Analysis
3 Programming
4 Others and Issues

- Families of algorithms.
- 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,

Learning to Search

When the reference policy is optimal

Searn  Daume III et al., MLJ 2009.

Aggra  Ross & Bagnell,

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,

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.

Inverse Reinforcement Learning

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

Propose: Kalman, 1968.

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.


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

2. Prediction accuracy. It had better work well.

3. Train speed. Debug/development productivity + maximum data input.

4. Test speed. Application efficiency