Machine Learning

Fall 2017

Structured Prediction

(structured perceptron, HMM, structured SVM)

Professor Liang Huang

(see Chap. 17 of CIML)
Structured Prediction

- **binary classification:** output is binary
- **multiclass classification:** output is a number (small # of classes)
- **structured classification:** output is a structure (seq., tree, graph)
  - part-of-speech tagging, parsing, summarization, translation
  - exponentially many classes: *search* (inference) efficiency is crucial!
Generic Perceptron

- online-learning: one example at a time
- learning by doing
  - find the best output under the current weights
  - update weights at mistakes

\[ x_i \rightarrow \text{inference} \rightarrow z_i \rightarrow y_i \rightarrow \text{update weights} \rightarrow w \]
Perceptron: from binary to structured classification

**Binary classification**

2 classes

\[
x \quad y=+1 \quad x \quad y=-1
\]

**Multiclass classification**

6 1 2 3 4 5 6 7 8 9

**Structured classification**

the man bit the dog

DT NN VBD DT NN

trivial

exact inference

update weights if \( y \neq z \)

easy

constant # of classes

exponential # of classes

hard

exact inference

update weights if \( y \neq z \)
Brief History of Perceptron

1959 Rosenblatt invention
1962 Novikoff proof
1969* Minsky/Papert book killed it
1997 Cortes/Vapnik SVM
1999 Freund/Schapire voted/avg: revived
2002 Collins structured
2005* McDonald/Crammer/Pereira structured MIRA
2006 Singer group aggressive
2007-2010* Singer group Pegasos

*mentioned in lectures but optional (others papers all covered in detail)
Multiclass Classification

- one weight vector ("prototype") for each class:
  \[ w = (w^{(1)}, w^{(2)}, \ldots, w^{(M)}) , \]

- multiclass decision rule:
  \[ \hat{y} = \arg \max_{z \in 1\ldots M} w^{(z)} \cdot x \]
  (best agreement w/ prototype)

Q1: what about 2-class?

Q2: do we still need augmented space?
Multiclass Perceptron

- on an error, penalize the weight for the wrong class, and reward the weight for the true class
Convergence of Multiclass

update rule:

\[ w \leftarrow w + \Delta \Phi(x, y, z) \]

separability:

\[ \exists u, \text{ s.t. } \forall (x, y) \in D, z \neq y \]
\[ u \cdot \Delta \Phi(x, y, z) \geq \delta \]

where \( w^{(i)} \) is used to calculate the functional margin for training example with label \( i \);

for a given training example \( x \) and a label \( y \), we define feature map function \( \Phi \) as

\[ \Phi(x, y) = (0^{(1)}, \ldots, 0^{(y-1)}, x, 0^{(y+1)}, \ldots, 0^{(M)}). \]

such that \( w \cdot \Phi(x, y) = w^{(y)} \cdot x \).

We also define that, with a given training example \( x \), the difference between two feature vectors for labels \( y \) and \( z \) as \( \Delta \Phi \):

\[ \Delta \Phi(x, y, z) = \Phi(x, y) - \Phi(x, z). \]
Example: POS Tagging

- **gold-standard:**
  - \textbf{the man bit the dog}
  - \textbf{DT NN VBD DT NN}

- **current output:**
  - \textbf{the man bit the dog}
  - \textbf{DT NN NN DT NN}

- **assume only two feature classes**
  - tag bigrams
  - word/tag pairs

- **weights ++:**
  - (NN, VBD)  (VBD, DT)  (VBD \rightarrow bit)

- **weights --:**
  - (NN, NN)  (NN, DT)  (NN \rightarrow bit)
Structured Perceptron

**Inputs:** Training set \((x_i, y_i)\) for \(i = 1 \ldots n\)

**Initialization:** \(W = 0\)

**Define:**
\[
F(x) = \arg\max_{y \in \text{GEN}(x)} \Phi(x, y) \cdot W
\]

**Algorithm:**
For \(t = 1 \ldots T, i = 1 \ldots n\)
\[
z_i = F(x_i)
\]
If \((z_i \neq y_i)\)
\[
W \leftarrow W + \Phi(x_i, y_i) - \Phi(x_i, z_i)
\]

**Output:** Parameters \(W\)
Inference: Dynamic Programming

\[ x(t-1) \rightarrow x(t) \rightarrow x(t+1) \]

\[ y(t-1) \rightarrow y(t) \rightarrow y(t+1) \]

**exact inference**

update weights if \( y \neq z \)
Viterbi for argmax

Viterbi search for argmax $p(t_1 \ldots t_n) \cdot p(w_1 \ldots w_1 t_1 \ldots t_n)$:

for $j = 1$ to $m$
  \[ Q[i, j] = p(t_j) \cdot p(w_i | t_j) \]

for $i = 2$ to $n$
  for $j = 1$ to $m$
    \[ Q[i, j] = \begin{cases} 0 & \text{best-pred} [i, j] = 0 \\ \infty & \text{best-score} = -\infty \\ \text{for } k = 1 \text{ to } m \\ r = p(t_j | t_k) \cdot p(w_i | t_j) \cdot Q[i-1, k] \end{cases} \]
    if $r > \text{best-score}$
      \[ \text{best-score} = r \]
      \[ \text{best-pred} [i, j] = k \]
      \[ Q[i, j] = r \]

final-best = 0
final-score = -∞
for $j = 1$ to $m$
  if $Q[n, j] > \text{final-score}$
    \[ \text{final-score} = Q[n, j] \]
    \[ \text{final-best} = j \]

print $t_{\text{final-best}}$
current = $t_{\text{final-best}}$
for $i = n-1$ down to $1$
  current = $\text{best-pred} [i+1, \text{current}]$
print $t_{\text{current}}$

how about unigram?
Python implementation

Complete this Python code implementing the Viterbi algorithm for part-of-speech tagging. It should print a list of word/tag pairs, e.g. [('a', 'D'), ('can', 'N'), ('can', 'A'), ('can', 'V'), ('a', 'D'), ('can', 'N')].

```python
from collections import defaultdict

best = defaultdict(lambda : defaultdict(float))
best[0]["<s>"] = 1
back = defaultdict(dict)

words = "<s> a can can can a can </s>".split()

tags = {"a": ["D"], "can": ["N", "A", "V"], "</s>": ["</s>"]}  # possible tags for each word
ptag = {"D": {"N": 1}, "V": {"</s>": 0.5, "D":0.5}, ... }  # ptag[x][y] = p(y | x)
pword = {"D": {"a": 0.5}, "N": {"can": 0.1}, ... }  # pword[x][w] = p(w | x)

for i, word in enumerate(words[1:], 1):
    for tag in tags[word]:
        for prev in best[i-1]:
            if tag in ptag[prev]:
                score = best[i-1][prev] * ptag[prev][tag] * pword[tag][word]
                if score > best[i][tag]:
                    best[i][tag] = score
                    back[i][tag] = prev

def backtrack(i, tag):
    if i == 0:
        return []
    return backtrack(i-1, back[i][tag]) + [(words[i], tag),]

print backtrack(len(words)-1, "</s>")[:-1]
```

Q: what about top-down recursive + memoization?
Trigram HMM

time complexity: $O(nT^3)$
in general: $O(nT^g)$ for g-gram
Efficiency vs. Expressiveness

- The inference (argmax) must be efficient
  - Either the search space $\text{GEN}(x)$ is small, or factored
  - Features must be local to $y$ (but can be global to $x$)
    - E.g. bigram tagger, but look at all input words (cf. CRFs)
Averaged Perceptron

Inputs: Training set \((x_i, y_i)\) for \(i = 1 \ldots n\)

Initialization: \(W_0 = 0\)

Define: \(F(x) = \arg \max_{y \in GEN(x)} \Phi(x, y) \cdot W\)

Algorithm:
For \(t = 1 \ldots T, i = 1 \ldots n\)
\[z_i = F(x_i)\]
If \((z_i \neq y_i)\)
\[W_{j+1} \leftarrow W_j + \Phi(x_i, y_i) - \Phi(x_i, z_i)\]

Output: Parameters \(W = \sum_j W_j\)

- more stable and accurate results
- approximation of voted perceptron
  (Freund & Schapire, 1999)
Averaging Tricks

- Daume (2006, PhD thesis)

**Algorithm AVERAGEDSTRUCTUREDPERCEPTRON\(x_{1:N}, y_{1:N}, I\)**

1: \(w_0 \leftarrow \langle 0, \ldots, 0 \rangle\)
2: \(w_a \leftarrow \langle 0, \ldots, 0 \rangle\)
3: \(c \leftarrow 1\)
4: **for** \(i = 1 \ldots I\) **do**
5:     **for** \(n = 1 \ldots N\) **do**
6:         \(\hat{y}_n \leftarrow \arg \max_{y \in \mathcal{Y}} w_0^\top \Phi(x_n, y_n)\)
7:         **if** \(y_n \neq \hat{y}_n\) **then**
8:             \(w_0 \leftarrow w_0 + \Phi(x_n, y_n) - \Phi(x_n, \hat{y}_n)\)
9:             \(w_a \leftarrow w_a + c\Phi(x_n, y_n) - c\Phi(x_n, \hat{y}_n)\)
10: **end if**
11: \(c \leftarrow c + 1\)
12: **end for**
13: **end for**
14: **return** \(w_0 - w_a/c\)

**Figure 2.3:** The averaged structured perceptron learning algorithm.
Do we need smoothing?

- smoothing is much easier in discriminative models
- just make sure for each feature template, its subset templates are also included
  - e.g., to include \((t_0 w_0 w_{-1})\) you must also include
  - \((t_0 w_0) (t_0 w_{-1}) (w_0 w_{-1})\)
  - and maybe also \((t_0 \ t_{-1})\) because \(t\) is less sparse than \(w\)
Geometry of Convergence Proof pt 1

1: repeat
2: for each example \((x, y)\) in \(D\) do
3: \[ z \leftarrow \text{EXACT}(x, w) \]
4: if \(z \neq y\) then
5: \[ w \leftarrow w + \Delta \Phi(x, y, z) \]
6: until converged

\[ w^{(k+1)} = w^{(k)} + \Delta \Phi(x, y, z) \]

perceptron update:

\[ u \cdot w^{(k+1)} = u \cdot w^{(k)} + u \cdot \Delta \Phi(x, y, z) \]

(by induction)

\[ u \cdot w^{(k+1)} \geq k\delta \]

(part 1: upperbound)
Geometry of Convergence Proof pt 2

1: repeat
2: for each example \((x, y)\) in \(D\) do
3: \(z \leftarrow \text{EXACT}(x, w)\)
4: if \(z \neq y\) then
5: \(w \leftarrow w + \Delta \Phi(x, y, z)\)
6: until converged

Violation: incorrect label scored higher

perceptron update:

\[
\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \Delta \Phi(x, y, z)
\]

by induction: \(\|\mathbf{w}^{(k+1)}\|^2 \leq kR^2\)

(parts 1+2 => update bounds:

\[
k \leq \frac{R^2}{\delta^2}
\]
Experiments
Experiments: Tagging

• (almost) identical features from (Ratnaparkhi, 1996)

• trigram tagger: current tag $t_i$, previous tags $t_{i-1}, t_{i-2}$

• current word $w_i$ and its spelling features

• surrounding words $w_{i-1} w_{i+1} w_{i-2} w_{i+2}..$

<table>
<thead>
<tr>
<th>Method</th>
<th>Error rate/%</th>
<th>Numits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perc, avg, cc=0</td>
<td>2.93</td>
<td>10</td>
</tr>
<tr>
<td>Perc, noavg, cc=0</td>
<td>3.68</td>
<td>20</td>
</tr>
<tr>
<td>Perc, avg, cc=5</td>
<td>3.03</td>
<td>6</td>
</tr>
<tr>
<td>Perc, noavg, cc=5</td>
<td>4.04</td>
<td>17</td>
</tr>
<tr>
<td>ME, cc=0</td>
<td>3.4</td>
<td>100</td>
</tr>
<tr>
<td>ME, cc=5</td>
<td><strong>3.28</strong></td>
<td>200</td>
</tr>
</tbody>
</table>
Experiments: NP Chunking

- **B-I-O scheme**

- Rockwell International Corp. 's Tulsa unit said it signed a tentative agreement...

- **features:**
  - unigram model
  - surrounding words and POS tags

<table>
<thead>
<tr>
<th>Current word</th>
<th>$w_i$ &amp; $t_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous word</td>
<td>$w_{i-1}$ &amp; $t_i$</td>
</tr>
<tr>
<td>Word two back</td>
<td>$w_{i-2}$ &amp; $t_i$</td>
</tr>
<tr>
<td>Next word</td>
<td>$w_{i+1}$ &amp; $t_i$</td>
</tr>
<tr>
<td>Word two ahead</td>
<td>$w_{i+2}$ &amp; $t_i$</td>
</tr>
<tr>
<td>Bigram features</td>
<td>$w_{i-2}, w_{i-1}$ &amp; $t_i$</td>
</tr>
<tr>
<td></td>
<td>$w_{i-1}, w_i$ &amp; $t_i$</td>
</tr>
<tr>
<td></td>
<td>$w_i, w_{i+1}$ &amp; $t_i$</td>
</tr>
<tr>
<td></td>
<td>$w_{i+1}, w_{i+2}$ &amp; $t_i$</td>
</tr>
<tr>
<td>Current tag</td>
<td>$p_i$ &amp; $t_i$</td>
</tr>
<tr>
<td>Previous tag</td>
<td>$p_{i-1}$ &amp; $t_i$</td>
</tr>
<tr>
<td>Tag two back</td>
<td>$p_{i-2}$ &amp; $t_i$</td>
</tr>
<tr>
<td>Next tag</td>
<td>$p_{i+1}$ &amp; $t_i$</td>
</tr>
<tr>
<td>Tag two ahead</td>
<td>$p_{i+2}$ &amp; $t_i$</td>
</tr>
<tr>
<td>Bigram tag features</td>
<td>$p_{i-2}, p_{i-1}$ &amp; $t_i$</td>
</tr>
<tr>
<td></td>
<td>$p_{i-1}, p_i$ &amp; $t_i$</td>
</tr>
<tr>
<td></td>
<td>$p_i, p_{i+1}$ &amp; $t_i$</td>
</tr>
<tr>
<td></td>
<td>$p_{i+1}, p_{i+2}$ &amp; $t_i$</td>
</tr>
<tr>
<td>Trigram tag features</td>
<td>$p_{i-2}, p_{i-1}, p_i$ &amp; $t_i$</td>
</tr>
<tr>
<td></td>
<td>$p_{i-1}, p_i, p_{i+1}$ &amp; $t_i$</td>
</tr>
<tr>
<td></td>
<td>$p_i, p_{i+1}, p_{i+2}$ &amp; $t_i$</td>
</tr>
</tbody>
</table>
Experiments: NP Chunking

- Results

<table>
<thead>
<tr>
<th>Method</th>
<th>F-Measure</th>
<th>Numits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perceptron, avg, cc=0</td>
<td>93.53</td>
<td>13</td>
</tr>
<tr>
<td>Perceptron, noavg, cc=0</td>
<td>93.04</td>
<td>35</td>
</tr>
<tr>
<td>Perceptron, avg, cc=5</td>
<td>93.33</td>
<td>9</td>
</tr>
<tr>
<td>Perceptron, noavg, cc=5</td>
<td>91.88</td>
<td>39</td>
</tr>
<tr>
<td>Max-ent, cc=0</td>
<td>92.34</td>
<td>900</td>
</tr>
<tr>
<td>Max-ent, cc=5</td>
<td>92.65</td>
<td>200</td>
</tr>
</tbody>
</table>

- (Sha and Pereira, 2003) **trigram** tagger
  - Voted perceptron: 94.09% vs. CRF: 94.38%
Structured Perceptron (Collins 02)

- challenge: search efficiency (exponentially many classes)
  - often use dynamic programming (DP)
  - but still too slow for repeated use, e.g. parsing is $O(n^3)$
  - and can’t use non-local features in DP
Learning w/ Inexact Inference (Huang et al 2012)

- routine use of inexact inference in NLP (e.g. beam search)
- how does structured perceptron work with inexact search?
  - so far most structured learning theory assume exact search
  - would search errors break these learning properties?
  - if so how to modify learning to accommodate inexact search?

the man bit the dog

DT NN VBD DT NN

x x \rightarrow \text{inexact inference} \rightarrow z

y y

update weights if $y \neq z$

w

does it still work???

beam search

greedy search
Idea: Search-Error-Robust Model

- train a “search-specific” or “search-error-robust” model
- we assume the same “search box” in training and testing
- model should “live with” search errors from search box
- exact search => convergence; greedy => no convergence
- how can we make perceptron converge w/ greedy search?
Convergence with Exact Search

- linear classification: converges iff. data is separable
- structured: converges iff. data separable & search exact
  - there is an oracle vector that correctly labels all examples
  - one vs the rest (correct label better than all incorrect labels)
- theorem: if separable, then \# of updates \leq \frac{R^2}{\delta^2}
  
Rosenblatt => Collins
  1957            2002
Convergence with Exact Search

current model

$w^{(k)}$

$w^{(k+1)}$

update

VV V

N

V N

NV

correct label

training example

time flies

N V

output space

$\{N,V\} \times \{N,V\}$

standard perceptron converges with exact search
No Convergence w/ Greedy Search

standard perceptron does not converge with greedy search
Early update (Collins/Roark 2004) to rescue

\[ w^{(k)} \rightarrow VV \quad V \rightarrow NV \]

\[ w^{(k+1)} \rightarrow V \rightarrow N \]

training example

\[
\begin{array}{c}
\text{time} \\
N \\
\text{flies} \\
V
\end{array}
\]

output space

\[ \{N,V\} \times \{N,V\} \]

standard perceptron does not converge with greedy search

stop and update at the first mistake
• why does inexact search break convergence property?
  • what is required for convergence? exactness?
• why does early update (Collins/Roark 04) work?
  • it works well in practice and is now a standard method
  • but there has been no theoretical justification
• we answer these Qs by inspecting the convergence proof
Geometry of Convergence Proof pt 1

1: repeat
2: for each example \((x, y)\) in \(D\) do
3: \(z \leftarrow \text{EXACT}(x, w)\)
4: if \(z \neq y\) then
5: \(w \leftarrow w + \Delta \Phi(x, y, z)\)
6: until converged

exact inference

update weights if \(y \neq z\)

perceptron update:

\[
\begin{align*}
\mathbf{w}^{(k+1)} &= \mathbf{w}^{(k)} + \Delta \Phi(x, y, z) \\
\mathbf{u} \cdot \mathbf{w}^{(k+1)} &= \mathbf{u} \cdot \mathbf{w}^{(k)} + \mathbf{u} \cdot \Delta \Phi(x, y, z) \\
&\geq \delta \quad \text{margin (by induction)}
\end{align*}
\]

\[
\begin{align*}

\mathbf{u} \cdot \mathbf{w}^{(k+1)} &\geq k \delta \\

\| \mathbf{u} \| \| \mathbf{w}^{(k+1)} \| &\geq \mathbf{u} \cdot \mathbf{w}^{(k+1)} \geq k \delta \\

\| \mathbf{w}^{(k+1)} \| &\geq k \delta
\end{align*}
\]
Geometry of Convergence Proof pt 2

1: repeat
2: for each example \((x, y)\) in \(D\) do
3:   \(z \leftarrow \text{EXACT}(x, w)\)
4: if \(z \neq y\) then
5:   \(w \leftarrow w + \Delta \Phi(x, y, z)\)
6: until converged

**perceptron update:**
\[
\begin{align*}
\|w^{(k+1)}\|^2 &= \|w^{(k)} + \Delta \Phi(x, y, z)\|^2 \\
&= \|w^{(k)}\|^2 + \|\Delta \Phi(x, y, z)\|^2 + 2 w^{(k)} \cdot \Delta \Phi(x, y, z) \\
&\leq R^2 \\
&\leq 0
\end{align*}
\]

violation: incorrect label scored higher

by induction: \(\|w^{(k+1)}\|^2 \leq kR^2\)

\((\text{part 2: upperbound})\)

parts 1+2 => update bounds:
\[
k \leq \frac{R^2}{\delta^2}
\]
Violation is All we need!

- exact search is **not** really required by the proof
- rather, it is only used to ensure violation!

- violation: incorrect label scored higher

the proof only uses 3 facts:
1. separation (margin)
2. diameter (always finite)
3. violation (but no need for exact)
Violation-Fixing Perceptron

- if we guarantee violation, we don’t care about exactness!
- violation is good b/c we can at least fix a mistake

same mistake bound as before!

1: repeat
2: for each example \((x, y)\) in \(D\) do
3: \((x, y', z)\) = FINDVIOLATION\((x, y, w)\)
4: if \(z \neq y\) then \(\triangleright (x, y', z)\) is a violation
5: \(w \leftarrow w + \Delta \Phi(x, y', z)\)
6: until converged
What if can’t guarantee violation

- this is why perceptron doesn’t work well w/ inexact search
  - because not every update is guaranteed to be a violation
  - thus the proof breaks; no convergence guarantee

- example: beam or greedy search
  - the model might prefer the correct label (if exact search)
  - but the search prunes it away
  - such a non-violation update is “bad” because it doesn’t fix any mistake
  - the new model still misguides the search
Standard Update: No Guarantee

Correct label scores higher. Non-violation: bad update!
Early Update: Guarantees Violation

training example

```
time     flies
N         V
```

output space

```
{N,V} x {N,V}
```

standard update doesn’t converge b/c it doesn’t guarantee violation

early update: incorrect prefix scores higher: a violation!
Experiments

trigram part-of-speech tagging

```
the man bit the dog
```

```
DT NN VBD DT NN
```

local features only, exact search tractable (proof of concept)

incremental dependency parsing

```
the man bit the dog
```

```
man
```

```
bit
dog
```

```
the
```

```
the
```

non-local features, exact search intractable (real impact)
1) Trigram Part of Speech Tagging

- standard update performs terribly with greedy search ($b=1$)
  - because search error is severe at $b=1$: half updates are bad!
  - no real difference beyond $b=2$: search error becomes rare

<table>
<thead>
<tr>
<th>beam size</th>
<th>max-violation</th>
<th>early</th>
<th>standard</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>97.2</td>
<td>97</td>
<td>96.8</td>
</tr>
<tr>
<td>2</td>
<td>97</td>
<td>97</td>
<td>96.6</td>
</tr>
<tr>
<td>3</td>
<td>97</td>
<td>97</td>
<td>96.4</td>
</tr>
<tr>
<td>4</td>
<td>97</td>
<td>97</td>
<td>97</td>
</tr>
<tr>
<td>5</td>
<td>97</td>
<td>97</td>
<td>97</td>
</tr>
<tr>
<td>6</td>
<td>97</td>
<td>97</td>
<td>97</td>
</tr>
<tr>
<td>7</td>
<td>97</td>
<td>97</td>
<td>97</td>
</tr>
<tr>
<td>8</td>
<td>97</td>
<td>97</td>
<td>97</td>
</tr>
<tr>
<td>9</td>
<td>97</td>
<td>97</td>
<td>97</td>
</tr>
<tr>
<td>10</td>
<td>97</td>
<td>97</td>
<td>97</td>
</tr>
</tbody>
</table>

% of bad (non-violation)

- standard updates: 53%, 10%, 1.5%, 0.5%
Max-Violation Reduces Training Time

- max-violation peaks at $b=2$, greatly reduced training time
- early update achieves the highest dev/test accuracy
- comparable to best published accuracy (Shen et al ‘07)
- future work: add non-local features to tagging

<table>
<thead>
<tr>
<th>beam size</th>
<th>beam</th>
<th>iter</th>
<th>time</th>
<th>test</th>
</tr>
</thead>
<tbody>
<tr>
<td>standard</td>
<td>-</td>
<td>6</td>
<td>162m</td>
<td>97.28</td>
</tr>
<tr>
<td>early</td>
<td>4</td>
<td>6</td>
<td>37m</td>
<td>97.27</td>
</tr>
<tr>
<td>max-violation</td>
<td>2</td>
<td>3</td>
<td>26m</td>
<td>97.27</td>
</tr>
<tr>
<td>Shen et al (2007)</td>
<td></td>
<td></td>
<td></td>
<td>97.33</td>
</tr>
</tbody>
</table>
2) Incremental Dependency Parsing

- DP incremental dependency parser (Huang and Sagae 2010)
- non-local history-based features rule out exact DP
  - we use beam search, and search error is severe
  - baseline: early update. extremely slow: 38 iterations
Max-violation converges much faster

- **early update**: 38 iterations, 15.4 hours (92.24)
- **max-violation**: 10 iterations, 4.6 hours (92.25)
  12 iterations, 5.5 hours (92.32)
Comparison b/w tagging & parsing

- search error is much more severe in parsing than in tagging
- standard update is OK in tagging except greedy search (b=1)
- but performs **horribly** in parsing even at large beam (b=8)
- because ~50% of standard updates are bad (non-violation)!

**take-home message:**
our methods are more helpful
for harder search problems!

<table>
<thead>
<tr>
<th>test</th>
<th>standard</th>
<th>early</th>
<th>max-violation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>79.1</td>
<td>92.1</td>
<td>92.2</td>
</tr>
</tbody>
</table>

% of bad standard updates
Annotated Bibliography


46