Daumé III, Langford, Marcu: Search-based Structured Prediction

SEARN

Julia Kreutzer

Seminar „Algorithms for Learning and Search in Structured Prediction“
Institut für Computerlinguistik
Universität Heidelberg

May 12, 2015
Overview

1. Introduction
2. The SEARN algorithm
3. SEARN analysed
4. SEARN in experiments
5. Conclusion
Introduction

Recap: What is structured prediction? Why/where it is challenging?
Structured Prediction

• Structured Prediction Problem
  \((x, c) \sim D\) with inputs \(x \in X\), cost vectors \(c \in (\mathbb{R}^+)^k\), \(k\) labels

• Goal
  Find \(h: X \rightarrow Y\) that minimizes \(L(D, h) = \mathbb{E}_{(x, c) \sim D} \{c_h(x)\}\)
Structured Prediction

• Structured Prediction Problem
  \((x, c) \sim D\) with inputs \(x \in X\), cost vectors \(c \in (\mathbb{R}^+)^k\), \(k\) labels

• Goal
  Find \(h: X \rightarrow Y\) that minimizes \(L(D, h) = \mathbb{E}_{(x,c) \sim D} \{c_h(x)\}\)

• Challenges
  Exact search is not always tractable
  Loss functions are not decomposable
  Complex feature functions
Structured Prediction

- Structured Prediction Problem
  \((x, c) \sim D\) with inputs \(x \in X\), cost vectors \(c \in (\mathbb{R}^+)^k\), \(k\) labels
- Goal
  Find \(h: X \to Y\) that minimizes \(L(D, h) = \mathbb{E}_{(x,c) \sim D}\{c_h(x)\}\)
- Challenges
  - Exact search is not always tractable
  - Loss functions are not decomposable
  - Complex feature functions
- [6] Under-generating vs. over-generating algorithms
Motivating Example

- POS-Tagging
  - Simplest setting: two words, two possible labels
Motivating Example

• POS-Tagging
  • Simplest setting: two words, two possible labels

Given: true labels
Motivating Example

• POS-Tagging
  • Simplest setting: two words, two possible labels

Given: loss function, not decomposable
Motivating Example

- POS-Tagging
  - Simplest setting: two words, two possible labels

Loss

<table>
<thead>
<tr>
<th></th>
<th>fliegen</th>
<th>fliegen</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>0.0</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>0.9</td>
<td></td>
</tr>
</tbody>
</table>

Missing: Cost for actions
Motivating Example

- POS-Tagging
  - Simplest setting: two words, two possible labels

How can we infer costs for local actions from global losses?
The SEARN algorithm

What kind of algorithm is it? Which problem does it solve? What is special?
Structured Prediction in Context

Source: [7]
Characteristics

• SEARN:
  • Meta-algorithm
    How can we learn from a teacher?
  • Search + Learn
    View the problem as a search problem
    Learn a classifier that walks through search space in a good way
    Instead of training on true path: train on path that is actually taken in practice
  • Any loss function
  • Any class of features
Components

- A search space $S$
- A cost-sensitive learning algorithm $A$
- Training data: structured, labeled
- A loss function $L(y, f(\hat{y}))$
- A good initial policy $\pi(s, c)$
Components

• A search space $S$
  
  State in the search space: $s = x \times (y_1, ..., y_T)$
  
  Final elements: sequence of choices $\hat{y}$
  
  Abstract: $f(\hat{y})$
  
  Concrete: $f(\hat{y}) = \hat{y}$
Components

• A search space $S$
• A cost-sensitive learning algorithm $A$
  Multiclass classifier $h(s)$ for location in search space $s$
  Trained on cost-sensitive training data
  „policy“ ($\to$ reinforcement learning)
Components

- A search space $S$
- A cost-sensitive learning algorithm $A$
- Training data: structured, labeled
  $(x, y) \in S^{SP}$
  $y \in Y$ decompose into vectors $(y_0, y_1, ..., y_T)$
  Arbitrary set of labels
Components

• A search space $S$
• A cost-sensitive learning algorithm $A$
• Training data: structured, labeled
• A loss function $L(\mathbf{y}, f(\hat{\mathbf{y}}))$
  - Computable for any full-length prediction sequence $(y_0, y_1, ..., y_T)$
  - Does not have to be decomposable
Components

- A search space $S$
- A cost-sensitive learning algorithm $A$
- Training data: structured, labeled
- A loss function $L(y, f(\hat{y}))$
- A good initial policy $\pi(s, c)$
  - Achieves low loss on training data
  - „the teacher“
Prediction

• At test time:
  • Use returned policy
  • Compute $y_0$ on basis of $x$
    Compute $y_1$ on basis of $y_0$ and $x$
    ...
    Compute $y_T$ on basis of $x, y_0, y_1, \ldots, y_{T-1}$
At test time:
- Use returned policy
- Compute $y_0$ on basis of $x$
  - Compute $y_1$ on basis of $y_0$ and $x$
  - ...
  - Compute $y_T$ on basis of $x$, $y_0$, $y_1$, ..., $y_{T-1}$

- No Markov assumption
- Feature function is essential
Training

Algorithm SEARN($S_{SP}$, $\pi$, $A$)
1: Initialize policy $h \leftarrow \pi$
2: while $h$ has a significant dependence on $\pi$ do
3: Initialize the set of cost-sensitive examples $S \leftarrow \emptyset$
4: for $(x, y) \in S_{SP}$ do
5: Compute predictions under the current policy $\hat{y} \sim x, h$
6: for $t = 1 \ldots T_x$ do
7: Compute features $\Phi = \Phi(s_t)$ for state $s_t = (x, y_1, \ldots, y_t)$
8: Initialize a cost vector $c = \langle \rangle$
9: for each possible action $a$ do
10: Let the cost $\ell_a$ for example $x, c$ at state $s$ be $\ell_h(c, s, a)$
11: end for
12: Add cost-sensitive example $(\Phi, \ell)$ to $S$
13: end for
14: end for
15: Learn a classifier on $S$: $h' \leftarrow A(S)$
16: Interpolate: $h \leftarrow \beta h' + (1 - \beta) h$
17: end while
18: return $h_{\text{last}}$ without $\pi$

Source: [1]
Motivating Example re-visited

• POS-Tagging
  • Simplest setting: two words, two possible labels

How can we infer costs for local actions from global losses?
Motivating Example re-visited

• POS-Tagging
  • Simplest setting: two words, two possible labels

We assume that all following choices will be made optimally. Follow the local regret.
Motivating Example re-visited

• POS-Tagging
  • Simplest setting: two words, two possible labels

![Diagram showing POS-tagging example with two words and two labels, N and V.]

<table>
<thead>
<tr>
<th>t=1</th>
<th>N</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>0.4</td>
<td>0.0</td>
</tr>
<tr>
<td>V</td>
<td>0.1</td>
<td>0.9</td>
</tr>
</tbody>
</table>

Loss Computing the regret

- t=1
  - a=“N“ \( l_a = 0.2 \)
  - a=“V“ \( l_a = 0.5 \)
  \( l_1 = <0.2, 0.5> \)

- t=2
  - a=“N“ \( l_a = 0.4 \)
  - a=“V“ \( l_a = 0.0 \)
  \( l_2 = <0.0, 0.4> \)
Motivating Example re-visited

• POS-Tagging
  • Simplest setting: two words, two possible labels

Creating cost-sensitive examples

\[ I_1 = \langle 0.2, 0.5 \rangle \]

features for state: \( \Phi_1 = \Phi(x) \)

\[ I_2 = \langle 0.0, 0.4 \rangle \]

features for state: \( \Phi_2 = \Phi(x, y_1) \)

add to \( S \):
\[ \{ \langle \Phi_1, I_1 \rangle, \langle \Phi_2, I_2 \rangle \} \]
Motivating Example re-visited

- POS-Tagging
  - Simplest setting: two words, two possible labels

```
<table>
<thead>
<tr>
<th>Word</th>
<th>t=1</th>
<th>t=2</th>
</tr>
</thead>
<tbody>
<tr>
<td>fliege</td>
<td>N</td>
<td>V</td>
</tr>
<tr>
<td>liege</td>
<td>V</td>
<td>N</td>
</tr>
</tbody>
</table>
```

Loss

- $l_1 = <0.2, 0.5>$
- Features for state: $\Phi_1 = \Phi(x)$
- $l_2 = <0.0, 0.4>$
- Features for state: $\Phi_2 = \Phi(x, y_1)$

Adding to $S$:

- $\{<\Phi_1, l_1>, <\Phi_2, l_2>\}$

Creating cost-sensitive examples:

- add to $S$: $\{<\Phi_1, l_1>, <\Phi_2, l_2>\}$

Train classifier on these tuples!
Details

• Initial policy
  • Takes full advantage of training data labels
  • Use search to create the initial policy (if not available analytically):
    • Given a node in the search space with cost vector, compute the best step to take
    • = given a node in the search space, find the shortest way to a goal
  • Optimal approximation: assume all further decisions will be made optimally
  • “greedy“ search
  • Choice of search algorithm influences bias in learning algorithm

\[ \pi(s, c) = \arg \min_{s} \min_{y_{t+1}, y_{t+2}, \ldots, y_T} c(y_1, \ldots, y_T) \]
Details

• Cost-sensitive examples
  • Run the given policy \( h \) over the training data
  • Prediction is sequence \( \hat{y} \) with loss \( c_{\hat{y}} \)
  • Compute (arbitrary) features \( \varphi = \varphi(s) \) for state \( s \) on sequence
  • Compute cost ("regret") for each state \( s \) and each action \( a \):

\[
l_h(c, s, a) = \mathbb{E}_{\hat{y} \sim (s, a, h)} c_{\hat{y}} - \min_{a'} \mathbb{E}_{\hat{y} \sim (s, a', h)} c_{\hat{y}}
\]

\( \rightarrow (\varphi, l) \in S \) is the input data structure for the learner
SEARN analysed

What can we tell about SEARN from an analytical perspective?
SEARN analysed

• Moving from initial policy to fully learned policy
• Each iteration „degrades“ current policy
• Analysis shows: the degradation is small
  • Theorem 2: Upper bound on the loss of a learned classifier
  • Lemma 1: Upper bound on the loss of a classifier after first iteration
  • Lemma 2: Upper bound after several iterations
• Proof for lemma 1:
  • Interpolation
    \[ h^{\text{new}} \leftarrow \beta h' + (1 - \beta)h \]
  • Maximal cost:
    \[ c_{\max} = \mathbb{E}_{(x,c) \sim D} \max_i c_i \]
  • Cases:
    1. Learned policy is never called
    2. Called once
    3. Called more than once
  • Assumption:
    \[ \beta < 1/T \]

\[
L(D, h^{\text{new}}) = Pr(c = 0)L(D, h^{\text{new}} | c = 0) \\
+ Pr(c = 1)L(D, h^{\text{new}} | c = 1) \\
+ Pr(c \geq 2)L(D, h^{\text{new}} | c \geq 2)
\]  

(6)

\[
\leq (1 - \beta)^TL(D, h) + T\beta(1 - \beta)^{-1}\left[L(D, h) + \ell_{h}^{\text{CS}}(h')\right] \\
+ \left[1 - (1 - \beta)^T - T\beta(1 - \beta)^{-1}\right]c_{\max}
\]  

(7)

\[
= L(D, h) + T\beta(1 - \beta)^{-1}\ell_{h}^{\text{CS}}(h') + \left(\sum_{i=2}^{T}(-1)^{i}\beta^{i}\binom{T}{i}\right)L(D, h) \\
+ \left[1 - (1 - \beta)^T - T\beta(1 - \beta)^{-1}\right]c_{\max}
\]  

(8)

Source: [1]
SEARN analysed

• Proof for lemma 1:
  • Interpolation
    \[ h^{\text{new}} \leftarrow \beta h' + (1 - \beta)h \]
  • Maximal cost:
    \[ c_{\text{max}} = \mathbb{E}_{(x,c) \sim D} \max_i c_i \]
  • Cases:
    1. Learned policy is never called
    2. Called once
    3. Called more than once
  • Assumption:
    \[ \beta < 1/T \]

\[ \leq L(D, h) + T \beta \ell_h^{\text{CS}}(h') \]
\[ + \left[ 1 - (1 - \beta)^T - T \beta (1 - \beta)^{T-1} \right] (c_{\text{max}} - L(D, h)) \]

\[ \leq L(D, h) + T \beta \ell_h^{\text{CS}}(h') \]
\[ + \left[ 1 - (1 - \beta)^T - T \beta (1 - \beta)^{T-1} \right] c_{\text{max}} \]

\[ = L(D, h) + T \beta \ell_h^{\text{CS}}(h') + \left( \sum_{i=2}^{T} (-1)^i \beta^i \binom{T}{i} \right) c_{\text{max}} \]

\[ \leq L(D, h) + T \beta \ell_h^{\text{CS}}(h') + \frac{1}{2} T^2 \beta^2 c_{\text{max}} \]

Source: [1]
SEARN in comparison

Why/where is SEARN superior to other structured prediction algorithms?
SEARN in comparison

- Independent models
  - Ignore structure or constrain membership
    - No complex features, limited to Hamming loss
    - \( = \) SEARN with features independent of history
  - Maximum Entropy Markov Model (MEMM)
    - Prediction on basis of \( k \) previous predictions
    - Assumption: previous predictions are correct \( \rightarrow \) can perform arbitrarily bad
    - Stacked MEMM’s: SEARN with \( \beta = 1 \) limited to sequence labeling
SEARN in comparison

- Perceptron-based models
  - Structured Perceptron
    Assumption: \( \text{argmax} \) is tractable
    SEARN in reverse: moving from incorrect towards true output
  - Incremental Perceptron
    Replace \( \text{argmax} \) with beam search
    Limitations: beam-search applications, decomposable loss function

- Global models
  - Conditional Random Field & Max-Margin Markov Network (\( M^3N \))
    In application limited to linear chain models with Markov assumption
    SEARN is more general
Experiments

How can we apply SEARN to structured prediction tasks? Does it perform well?
1. Sequence Labeling
   • Handwriting recognition
   • Spanish NER
   • Syntactic chunking
   • Joint chunking and POS tagging

SEARN:
   • Loss per label: Hamming loss
   • Loss per chunk: F1
   • Left-to-right greedy search
   • Chunk-at-a-time decoding (BIO)
   • Reduction to binary classification

\[ l^{Ham}(y, \hat{y}) = \sum_{n=1}^{N} 1[y_n \neq \hat{y}_n] \]
\[ l^{F}(y, \hat{y}) = \frac{2|y \cap \hat{y}|}{|y| + |\hat{y}|} \]
1. Sequence Labeling
   • Results

<table>
<thead>
<tr>
<th>ALGORITHM</th>
<th>Handwriting</th>
<th>NER</th>
<th>Chunk</th>
<th>C+T</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Small</td>
<td>Large</td>
<td>Small</td>
<td>Large</td>
</tr>
<tr>
<td><strong>CLASSIFICATION</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Perceptron</td>
<td>65.56</td>
<td>70.05</td>
<td>91.11</td>
<td>94.37</td>
</tr>
<tr>
<td>Log Reg</td>
<td>68.65</td>
<td>72.10</td>
<td>93.62</td>
<td>96.09</td>
</tr>
<tr>
<td>SVM-Lin</td>
<td>75.75</td>
<td>82.42</td>
<td>93.74</td>
<td>97.31</td>
</tr>
<tr>
<td>SVM-Quad</td>
<td>82.63</td>
<td>82.52</td>
<td>85.49</td>
<td>85.49</td>
</tr>
<tr>
<td><strong>STRUCTURED</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Str. Perc.</td>
<td>69.74</td>
<td>74.12</td>
<td>93.18</td>
<td>95.32</td>
</tr>
<tr>
<td>CRF</td>
<td>~</td>
<td>~</td>
<td>94.94</td>
<td>~</td>
</tr>
<tr>
<td>SVM_struct</td>
<td>~</td>
<td>~</td>
<td>94.90</td>
<td>~</td>
</tr>
<tr>
<td>M³N-Lin</td>
<td>81.00</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>M³N-Quad</td>
<td>87.00</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td><strong>SEARN</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Perceptron</td>
<td>70.17</td>
<td>76.88</td>
<td>95.01</td>
<td>97.67</td>
</tr>
<tr>
<td>Log Reg</td>
<td>73.81</td>
<td>79.28</td>
<td>95.90</td>
<td>98.17</td>
</tr>
<tr>
<td>SVM-Lin</td>
<td>82.12</td>
<td>90.58</td>
<td>95.91</td>
<td>98.11</td>
</tr>
<tr>
<td>SVM-Quad</td>
<td>87.55</td>
<td>90.91</td>
<td>89.31</td>
<td>90.01</td>
</tr>
</tbody>
</table>

~: could not scale
—: not reported
F1 on Chunk, C+T
Hamming on Handwriting, NER

Source: [1]
2. Automatic Document Summarization
   • Greedily extract sentences of a document until word limit reached
   • Vine-growth model on syntactic dependency parse tree
   • Actions: add root of new tree or child of already added node
   • Loss: Rouge
   • Initial Policy: argmax intractable (constraints), beam search approximation
2. Automatic Document Summarization

- Results

Rouge score

<table>
<thead>
<tr>
<th></th>
<th>ORACLE</th>
<th></th>
<th>SEARN</th>
<th></th>
<th>BAYESUM</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Vine</td>
<td>Extr</td>
<td>Vine</td>
<td>Extr</td>
<td>D05</td>
<td>D03</td>
<td>Base</td>
</tr>
<tr>
<td>100 w</td>
<td>.0729</td>
<td>.0362</td>
<td>.0415</td>
<td>.0345</td>
<td>.0340</td>
<td>.0316</td>
<td>.0181</td>
</tr>
<tr>
<td>250 w</td>
<td>.1351</td>
<td>.0809</td>
<td>.0824</td>
<td>.0767</td>
<td>.0762</td>
<td>.0698</td>
<td>.0403</td>
</tr>
</tbody>
</table>

Source: [1]
Conclusion

What did we learn about SEARN? What did we not learn?
Summary

• Core idea: combining search and learning
  „Instead of accounting for search in the process of learning, I treat the structured prediction problem as being defined by a search process.” [2]

• Meta-algorithm for structured prediction
  • Minimal requirements for structure and loss function
  • Start from good initial policy and generalize

• Competitive results for sequence labeling and summarization task
Problems and Questions

• The algorithm
  • Heavily relies on quality of initial policy
    • Efficiency
    • Bias
    • Noise
  • Definition of convergence criterion?
  • Missing details for SEARN in test time
  • Policy might be stochastic
Problems and Questions

• The application
  • Documented experiments lack interesting details
    • Iteration numbers
    • Observed speed of convergence
    • Interpolation and storage of classifiers
    • Use, integration and parametrization of base classifiers
  • Only a few experiments (by the same person)
  • (Un-)popularity in practice?
  • Machine Translation?
Problems and Questions

Your opinion! 😊
Problems and Questions

Your opinion! 😊

Thank you!
References


