Intro to Structured Prediction

NOV. 22, 2016
Outline

Introduction

Classical Approaches
- HMM
- CRF
- Structured Perceptron
- Structured SVM

Learning to Search
- SEARN
- DAgger
- AggreVaTe

Conclusions
Introduction

For many problems, making independent classifications is suboptimal
- Part-of-speech tagging
- Named Entity recognition
- Image segmentation
- Etc...

In each case, the output variables are all inter-related
- For POS/NER, the current word’s tag depends on the word occurring before/after it
- For images, the current pixel’s label depends on the pixels surrounding it

Structured prediction techniques explicitly model these kinds of relationships among output variables
Classical Solutions

Some of the most well-known techniques for structured prediction:

- Hidden Markov Model
- Conditional Random Field
- Structured Perceptron
- Structured SVM

We’ll go over each of these briefly before moving to more recent advances
Hidden Markov Model
Conditional Random Field

Fig. 2.7 Graphical model of a linear-chain CRF in which the transition factors depend on all of the observations.
Conditional Random Field

**Definition 2.2.** Let \( Y, X \) be random vectors, \( \theta = \{\theta_k\} \in \mathbb{R}^K \) be a parameter vector, and \( \mathcal{F} = \{f_k(y, y', x_t)\}_{k=1}^K \) be a set of real-valued feature functions. Then a *linear-chain conditional random field* is a distribution \( p(y|x) \) that takes the form:

\[
p(y|x) = \frac{1}{Z(x)} \prod_{t=1}^{T} \exp \left\{ \sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, x_t) \right\}, \tag{2.18}
\]

where \( Z(x) \) is an input-dependent normalization function

\[
Z(x) = \sum_y \prod_{t=1}^{T} \exp \left\{ \sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, x_t) \right\}. \tag{2.19}
\]
Structured Perceptron

**Inputs:** Training examples \((x_k, y_k)\)

**Initialization:** \(\bar{\lambda} = 0\)

**Algorithm:**

For \(l = 1\) to \(L\), \(k = 1\) to \(n\)

- Use Viterbi to get \(z_k = \arg \max_z \bar{\lambda} \cdot \Phi(x_k, z)\)
- If \(z_k \neq y_k\) then \(\bar{\lambda} = \bar{\lambda} + \Phi(x_k, y_k) - \Phi(x_k, z_k)\)

**Output:** weights \(\bar{\lambda}\)
Structured SVM

Generalization of SVM to structured outputs

Training:

$$\min_{w, \xi} \|w\|^2 + C \sum_{n=1}^{\ell} \xi_n$$

s.t.  \quad \langle w, \Psi(x_n, y_n) \rangle - \langle w, \Psi(x_n, y) \rangle + \xi_n \geq \Delta(y_n, y), \quad n = 1, \ldots, \ell, \quad \forall y \in \mathcal{Y}$$

Inference:

$$f(x) = \arg\max_{y \in \mathcal{Y}} \langle w, \Psi(x, y) \rangle$$
Structured SVM

Main problem:
- How to deal with this many constraints during training?

Solution: cutting-plane algorithm
- Iteratively construct a working set of constraints $S$ during training
  - Initialized to empty set
  - During each iteration, compute the solution given the current $S$, and add the most violating constraint to $S$
  - Stop once no constraint can be found that is violated by more than a desired precision $\epsilon$
Learning to Search
Frames the structured prediction problem as a reinforcement learning task

(Brief) Review of reinforcement learning:
- State – a summary of the agent’s experience, including observations, action history, and rewards
- Policy – a map from states to actions
  - Can be deterministic or stochastic
- Rewards are received by the agent upon taking actions
- Goal of the agent is to learn a policy that maximizes its future rewards
Learning to Search

In the structured prediction setting:

- A state represents the input data $x$, along with any classification decisions made thus far by the system.
- The policy maps from the state to the next classification decision to be made.
- Rewards are received for correct predictions.
- Goal of the agent is to learn a policy that makes as few errors as possible.
Learning to Search

As an example, consider part-of-speech tagging:

The dog chased the cat.
Learning to Search

As an example, consider part-of-speech tagging:

The  dog  chased  the  cat  .

Possible actions:
- DET
- NOUN
- VERB
- PUNCT
- ADJ

Current state:
The  dog  chased  the  cat  .
Learning to Search

As an example, consider part-of-speech tagging:

Possible actions

Current state

The dog chased the cat .
Learning to Search

As an example, consider part-of-speech tagging:

```
The dog chased the cat .
```

Possible actions:

- DET
- NOUN
- VERB
- PUNCT
- ADJ

Current state:

- DET

The dog chased the cat .
Learning to Search

As an example, consider part-of-speech tagging:

The dog chased the cat.

Possible actions:
- DET
- NOUN
- VERB
- PUNCT
- ADJ

Current state:
- DET
- dog
- chased
- the
- cat
- .
Learning to Search

As an example, consider part-of-speech tagging:

The dog chased the cat .

Possible actions
DET NOUN VERB PUNCT ADJ

Current state
The dog chased the cat .
DET NOUN
Learning to Search

As an example, consider part-of-speech tagging:

The dog chased the cat .

Possible actions

Current state

DET NOUN VERB PUNCT ADJ

DET NOUN
Learning to Search

As an example, consider part-of-speech tagging:

The dog chased the cat.

Possible actions: DET, NOUN, VERB, PUNCT, ADJ

Current state: DET, NOUN, VERB, the, cat.
Learning to Search

As an example, consider part-of-speech tagging:

The dog chased the cat .

Possible actions: DET, NOUN, VERB, PUNCT, ADJ

Current state: The DET dog NOUN chased VERB the DET cat PUNCT .
Learning to Search

As an example, consider part-of-speech tagging:

The dog chased the cat.

Possible actions:
- DET
- NOUN
- VERB
- PUNCT
- ADJ

Current state:
- The DET
- dog NOUN
- chased VERB
- the DET
- cat

.
Learning to Search

As an example, consider part-of-speech tagging:

As an example, consider part-of-speech tagging:

The dog chased the cat.

Possible actions

Current state

DET NOUN VERB PUNCT ADJ

DET NOUN VERB DET

The dog chased the cat.
Learning to Search

As an example, consider part-of-speech tagging:

The dog chased the cat.
Learning to Search

As an example, consider part-of-speech tagging:

The dog chased the cat.

Current state:
- The: DET
- dog: NOUN
- chased: VERB
- the: DET
- cat: NOUN

Possible actions:
- DET
- NOUN
- VERB
- PUNCT
- ADJ
Learning to Search

As an example, consider part-of-speech tagging:

Possible actions:
- DET
- NOUN
- VERB
- PUNCT
- ADJ

Current state:
- The
  - DET
- dog
  - NOUN
- chased
  - VERB
- the
  - DET
- cat
  - NOUN
- .
  - PUNCT
Learning to Search

To use reinforcement learning for structured prediction, we need:

- Training data: \((x_n, y_n) \sim D\)
- A search space \(S\) of possible outputs
- A cost sensitive learning algorithm
- A known loss function
- A good initial policy that can achieve low loss on the training data
Cost-Sensitive Classification

In binary classification, we treat all errors are equal
◦ Overall goal is to minimize the total number of errors

In cost-sensitive classification, each error has an associated cost
◦ Overall goal is to minimize the total cost of errors

Methods for cost-sensitive classification include:
◦ Algorithms designed specifically for the task
◦ Extensions to existing cost-insensitive algorithms (typically based on either thresholding the output or resampling the input examples)
SEARN

Motivation:
- If a model is trained on just perfect data, it won’t know what to do if a mistake is encountered
- Allow the current policy to “explore” the training data, so it will not be caught (too) off-guard at test-time

Example:
The full SEARN algorithm is below:

1. Initialize the current policy to the optimal policy

2. Repeat:
   (a) Use the current policy to generate paths over all training examples
   (b) For each example, for each step in the path traversed by the current policy:
       i. Generate a multiclass example whose classes are possible decisions and whose losses
          are based on the loss of the current policy (see below)
   (c) Learn a new multiclass classifier on the basis of the examples
   (d) Find an interpolation constant $\beta$ on development data that improves performance
   (e) Set the current policy to $\beta$ times the new policy plus $1 - \beta$ times the old policy

3. Return the current policy without the optimal policy
SEARN – Generating Examples

With current policy

rollin

one-step deviations

rollout

With current policy
The full SEARN algorithm is below:

1. Initialize the current policy to the optimal policy

2. Repeat:
   (a) Use the current policy to generate paths over all training examples
   (b) For each example, for each step in the path traversed by the current policy:
       i. Generate a multiclass example whose classes are possible decisions and whose losses are based on the loss of the current policy (see below)
   (c) Learn a new multiclass classifier on the basis of the examples
   (d) Find an interpolation constant $\beta$ on development data that improves performance
   (e) Set the current policy to $\beta$ times the new policy plus $1 - \beta$ times the old policy

3. Return the current policy without the optimal policy
The full SEARN algorithm is below:

1. Initialize the current policy to the optimal policy
2. Repeat:
   (a) Use the current policy to generate paths over all training examples
   (b) For each example, for each step in the path traversed by the current policy:
      i. Generate a multiclass example whose classes are possible decisions and whose losses are based on the loss of the current policy (see below)
   (c) Learn a new multiclass classifier on the basis of the examples
   (d) Find an interpolation constant $\beta$ on development data that improves performance
   (e) Set the current policy to $\beta$ times the new policy plus $1 - \beta$ times the old policy
3. Return the current policy without the optimal policy

Example: POS Tagging

1.) Optimal policy is to just directly use the available gold labels
The full SEARN algorithm is below:

1. Initialize the current policy to the optimal policy

2. Repeat:
   
   (a) Use the current policy to generate paths over all training examples
   (b) For each example, for each step in the path traversed by the current policy:
       
       i. Generate a multiclass example whose classes are possible decisions and whose losses are based on the loss of the current policy (see below)
   (c) Learn a new multiclass classifier on the basis of the examples
   (d) Find an interpolation constant $\beta$ on development data that improves performance
   (e) Set the current policy to $\beta$ times the new policy plus $1 - \beta$ times the old policy

3. Return the current policy without the optimal policy

Example: POS Tagging

2.) Generate paths using the optimal policy (i.e. gold labels), train a new policy, and interpolate this with the original
The full SEARN algorithm is below:

1. Initialize the current policy to the optimal policy
2. Repeat:
   (a) Use the current policy to generate paths over all training examples
   (b) For each example, for each step in the path traversed by the current policy:
      i. Generate a multiclass example whose classes are possible decisions and whose losses are based on the loss of the current policy (see below)
   (c) Learn a new multiclass classifier on the basis of the examples
   (d) Find an interpolation constant $\beta$ on development data that improves performance
   (e) Set the current policy to $\beta$ times the new policy plus $1 - \beta$ times the old policy
3. Return the current policy without the optimal policy

Example: POS Tagging

3.) Generate paths using the interpolated policy. This means some tags will be correct, some will be incorrect.
The full SEARN algorithm is below:

1. Initialize the current policy to the optimal policy

2. Repeat:

   (a) Use the current policy to generate paths over all training examples
   (b) For each example, for each step in the path traversed by the current policy:
       i. Generate a multiclass example whose classes are possible decisions and whose losses are based on the loss of the current policy (see below)
   (c) Learn a new multiclass classifier on the basis of the examples
   (d) Find an interpolation constant $\beta$ on development data that improves performance
   (e) Set the current policy to $\beta$ times the new policy plus $1 - \beta$ times the old policy

3. Return the current policy without the optimal policy

4.) Train a new policy using the generated paths. New policy should be more robust to unseen situations than the previous policy
Experiments conducted on several tasks:
- Handwriting recognition
- Spanish named entity recognition
- Syntactic chunking
- Joint chunking + POS tagging

Methods:
- Non-structured approaches: perceptron, logistic regression, SVM
- Structured baselines: structured perceptron, CRF, structured SVM, M³N
- SEARN + perceptron
- SEARN + logistic regression
- SEARN + SVM
## SEARN – Experiments

<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\text{\text{struct}}</td>
<td>–</td>
<td>–</td>
<td>94.90</td>
<td>~</td>
</tr>
<tr>
<td>M\text{\text{N-Lin}}</td>
<td>81.00</td>
<td>~</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>M\text{\text{N-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>

Table 1. Empirical comparison of performance of alternative structured prediction algorithms against SEARN on sequence labeling tasks. (Top) Comparison for whole-sequence 0/1 loss; (Bottom) Comparison for individual losses: Hamming for handwriting and Chunking+Tagging and F for NER and Chunking. SEARN is always optimized for the appropriate loss.
DAgger

Initialize $\mathcal{D} \leftarrow \emptyset$.
Initialize $\hat{\pi}_1$ to any policy in $\Pi$.

for $i = 1$ to $N$ do
    Let $\pi_i = \beta_i \pi^* + (1 - \beta_i) \hat{\pi}_i$.
    Sample $T$-step trajectories using $\pi_i$.
    Get dataset $\mathcal{D}_i = \{(s, \pi^*(s))\}$ of visited states by $\pi_i$ and actions given by expert.
    Aggregate datasets: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_i$.
    Train classifier $\hat{\pi}_{i+1}$ on $\mathcal{D}$.
end for
Return best $\hat{\pi}_i$ on validation.

Algorithm 3.1: DAGGER Algorithm.
Dagger vs SEARN

SEARN uses roll-out with the current policy to determine the loss of each action.

In Dagger, the “correct” action is explicitly given by the expert.

However, no sense of cost associated with each action – all errors treated equally:

- In some settings, some errors are more costly than others.
- E.g. a self-driving car should have a much higher cost to drive off a cliff than to cross into another lane.
DAgger – Generating Examples
DAgger – Experiments

Experiments conducted on 3 tasks:
- Super Tux Kart – 3D racing game similar to Mario Kart
- Super Mario Bros.
- Handwriting recognition
DAgger – Experiments

Super Tux Kart Results
DAgger – Experiments

Super Mario Bros. Results
DAgger – Experiments

Handwriting Recognition Results
Algorithm 1 Aggrevate: Imitation Learning with Cost-To-Go

Initialize $D \leftarrow \emptyset$, $\hat{\pi}_1$ to any policy in $\Pi$.

for $i = 1$ to $N$ do

Let $\pi_i = \beta_i \pi^* + (1 - \beta_i) \hat{\pi}_i$ #Optionally mix in expert's own behavior.
Collect $m$ data points as follows:

for $j = 1$ to $m$ do

Sample uniformly $t \in \{1, 2, \ldots, T\}$.
Start new trajectory in some initial state drawn from initial state distribution
Execute current policy $\pi_i$ up to time $t - 1$.
Execute some exploration action $a_t$ in current state $s_t$ at time $t$
Execute expert from time $t + 1$ to $T$, and observe estimate of cost-to-go $\hat{Q}$ starting at time $t$

end for

Get dataset $D_i = \{(s, t, a, \hat{Q})\}$ of states, times, actions, with expert's cost-to-go.
Aggregate datasets: $D \leftarrow D \cup D_i$.
Train cost-sensitive classifier $\hat{\pi}_{i+1}$ on $D$

(Alternately: use any online learner on the data-sets $D_i$ in sequence to get $\hat{\pi}_{i+1}$)

end for

Return best $\hat{\pi}_i$ on validation.
AggreVaTe – Generating Examples

With current policy

Using expert policy
Conclusions

While classical techniques for structured prediction remain strong baselines, there are more recent developments that are worth considering.

The learning to search framework draws connections between structured prediction and reinforcement learning, reframing the problem as a matter of finding an optimal policy for making predictions.

Empirical results show strong (and scalable) performance on a variety of tasks, including tagging, image recognition, and syntactic chunking problems.