Advances in Structured Prediction

John Langford
Microsoft Research
jl@hunch.net

Hal Daumé III
U Maryland
me@hal3.name

Slides and more: http://hunch.net/~l2s
Examples of structured prediction
Sequence labeling

\[ x = \text{the monster ate the sandwich} \]
\[ y = Dt \quad Nn \quad Vb \quad Dt \quad Nn \]

\[ x = \text{Yesterday I traveled to Lille} \]
\[ y = \quad - \quad \text{PER} \quad - \quad - \quad \text{LOC} \]
NLP algorithms use a kitchen sink of features.
(Bipartite) matching

What is the anticipated cost of collecting fees under the new proposal?
This text has been automatically translated from Arabic:

Moscow stressed tone against Iran on its nuclear program. He called Russian Foreign Minister Tehran to take concrete steps to restore confidence with the international community, to cooperate fully with the IAEA. Conversely Tehran expressed its willingness...
Image segmentation
Protein secondary structure prediction
Standard solution methods

1. Each prediction is independent
2. Shared parameters via “multitask learning”
3. Assume tractable graphical model; optimize
4. Hand-crafted
Predicting independently

- $h : \text{features of nearby voxels} \rightarrow \text{class}$
- Ensure output is coherent at test time

✔ Very simple to implement, often efficient

✗ Cannot capture correlations between predictions
✗ Cannot optimize a joint loss
Prediction with multitask bias

- $h : \text{features} \rightarrow (\text{hidden representation})$
  $\rightarrow \text{yes/no}$
- Share (hidden representation) across all classes

- All advantages of predicting independently
- May implicitly capture correlations

- Learning may be hard (… or not?)
- Still not optimizing a joint loss
Optimizing graphical models

- Encode output as a graphical model
- Learn parameters of that model to optimize:
  - $p(\text{true labels} \mid \text{input})$
  - cvx u.b. on $\text{loss(\text{true labels}, \text{predicted labels})}$

- Guaranteed consistent outputs
- Can capture correlations explicitly

- Assumed independence assumptions may not hold
- Computationally intractable with too many “edges” or non-decomposable loss function
Back to the original problem...

- How to optimize a discrete, joint loss?

- **Input:** \( x \in X \)
- **Truth:** \( y \in Y(x) \)
- **Outputs:** \( Y(x) \)
- **Predicted:** \( \hat{y} \in Y(x) \)
- **Loss:** \( \text{loss}(y, \hat{y}) \)
- **Data:** \( (x,y) \sim D \)
Back to the original problem...

- How to optimize a discrete, joint loss?

- **Input:** \( x \in X \)
- **Truth:** \( y \in Y(x) \)
- **Outputs:** \( Y(x) \)
- **Predicted:** \( \hat{y} \in Y(x) \)
- **Loss:** \( \text{loss}(y, \hat{y}) \)
- **Data:** \( (x, y) \sim D \)

**Goal:**

\[
\text{find } h \in H \\
\text{such that } h(x) \in Y(x) \\
\text{minimizing } \\
E_{(x,y) \sim D} \left[ \text{loss}(y, h(x)) \right] \\
\text{based on } N \text{ samples} \\
(x_n, y_n) \sim D
\]
Challenges

- Output space is too big to exhaustively search:
  - Typically exponential in size of input
  - implies $y$ must decompose in some way
    (often: $x$ has many pieces to label)

- Loss function has combinatorial structure:
  - Intersection over union
  - Edit Distance
Decomposition of label

- Decomposition of $y$ often implies an ordering
  
  I | can | can | a | can
  Pro | Md | Vb |Dt | Nn

- But sometimes not so obvious....

(we'll come back to this case later...)
Search spaces

- When $y$ decomposes in an ordered manner, a sequential decision making process emerges.
Search spaces

- When $y$ decomposes in an ordered manner, a sequential decision making process emerges.

Encodes an output $\hat{y} = \hat{y}(e)$ from which $\text{loss}(y, \hat{y})$ can be computed (at training time).
Policies

- A policy maps observations to actions

\[ \pi(x, t, \tau, \ldots \text{anything else}) = a \]
Versus reinforcement learning

In learning to search (L2S):

- *Labeled data* at training time
  - \( \Rightarrow \) can construct good/optimal policies
- Can “reset” and try the same example many times

\[
\begin{align*}
o_1 & \rightarrow a_1 \rightarrow o_2 \rightarrow a_2 \rightarrow o_3 \rightarrow a_3 \rightarrow \ldots \rightarrow o_T \rightarrow a_T \rightarrow \text{loss} \\
\text{Classifier} & = \pi(o_1) \\
\text{Goal:} & \min_{\pi} E \left[ \text{loss}(\pi) \right]
\end{align*}
\]
Labeled data $\rightarrow$ Reference policy

Given partial traj. $a_1, a_2, \ldots, a_{t-1}$ and true label $y$

The minimum achievable loss is:

$$\min \text{ loss}(y, \hat{y}(\tilde{a}))$$

The optimal action is the corresponding $a_t$

The optimal policy is the policy that always selects the optimal action
Ingredients for learning to search

- **Training data**: \((x_n, y_n) \sim D\)
- **Output space**: \(Y(x)\)
- **Loss function**: \(\text{loss}(y, \hat{y})\)
- **Decomposition**: \(\{o\}, \{a\}, \ldots\)
- **Reference policy**: \(\pi^{\text{ref}}(o, y)\)
An analogy from playing Mario

From Mario AI competition 2009

Input:

Output:

Jump in \{0,1\}
Right in \{0,1\}
Left in \{0,1\}
Speed in \{0,1\}

High level goal:
Watch an expert play and learn to mimic her behavior
Training (expert)
Warm-up: Supervised learning

1. Collect trajectories from expert $\pi^{\text{ref}}$
2. Store as dataset $D = \{ (o, \pi^{\text{ref}}(o,y)) \mid o \sim \pi^{\text{ref}} \}$
3. Train classifier $\pi$ on $D$

- Let $\pi$ play the game!
Test-time execution (sup. learning)
What's the (biggest) failure mode?

The expert never gets stuck next to pipes
⇒ Classifier doesn't learn to recover!
Warm-up II: Imitation learning

1. Collect trajectories from expert $\pi^{\text{ref}}$
2. Dataset $D_0 = \{ (o, \pi^{\text{ref}}(o, y)) \mid o \sim \pi^{\text{ref}} \}$
3. Train $\pi_1$ on $D_0$
4. Collect new trajectories from $\pi_1$
   - But let the expert steer!
5. Dataset $D_1 = \{ (o, \pi^{\text{ref}}(o, y)) \mid o \sim \pi_1 \}$
6. Train $\pi_2$ on $D_0 \cup D_1$

- In general:
  - $D_n = \{ (o, \pi^{\text{ref}}(o, y)) \mid o \sim \pi_n \}$
  - Train $\pi_{n+1}$ on $\bigcup_{i \leq n} D_i$

If $N = T \log T$, $L(\pi_n) < T \varepsilon_N + O(1)$ for some $n$
Test-time execution (DAgger)
What's the biggest failure mode?

Classifier only sees *right* versus *not-right*

- No notion of *better* or *worse*
- No *partial credit*
- Must have a single *target* answer
Aside: cost-sensitive classification

Classifier: \( h : x \rightarrow [K] \)

Multiclass classification

- **Data:** \( (x,y) \in X \times [K] \)
- **Goal:** \( \min_h \text{Pr}( h(x) \neq y ) \)

Cost-sensitive classification

- **Data:** \( (x,c) \in X \times [0,\infty)^K \)
- **Goal:** \( \min_h E_{(x,c)}[ c_h(x) ] \)
Learning to search: AggraVaTe

1. Let learned policy $\pi$ drive for $t$ timesteps to obs. $o$

2. For each possible action $a$:
   - Take action $a$, and let expert $\pi^{\text{ref}}$ drive the rest
   - Record the overall loss, $c_a$

3. Update $\pi$ based on example: $(o, \langle c_1, c_2, \ldots, c_K \rangle)$

4. Goto (1)
Learning to search: AggraVaTe

1. **Generate an initial trajectory** using the current policy

2. Foreach decision on that trajectory with obs. $o$:
   a) Foreach possible action $a$ (one-step deviations)
      i. Take that action
      ii. Complete this trajectory using reference policy
      iii. Obtain a final loss, $c_a$
   b) Generate a cost-sensitive classification example: $(o, \bar{c})$
Learning to search: AggraVaTe

1. Generate an initial trajectory using the current policy

2. Foreach decision on that trajectory with obs. \( o \):
   a) Foreach possible action \( a \) (one-step deviations)
      i. Take that action
      ii. Complete this trajectory using reference policy
      iii. Obtain a final loss, \( c_a \)
   b) Generate a cost-sensitive classification example:
      \( (o, \tilde{c}) \)

Often it's possible to analytically compute this loss without having to execute a roll-out!
Example I: Sequence labeling

- Receive input:
  \[ \begin{align*}
  x &= \text{the monster ate the sandwich} \\
  y &= \text{Dt Nn Vb Dt Nn}
  \end{align*} \]

- Make a sequence of predictions:
  \[ \begin{align*}
  x &= \text{the monster ate the sandwich} \\
  \hat{y} &= \text{Dt Dt Dt Dt Dt Dt Dt}
  \end{align*} \]

- Pick a timestep and try all perturbations there:
  \[ \begin{align*}
  x &= \text{the monster ate the sandwich} \\
  \hat{y}_\text{Dt} &= \text{Dt Dt} \\
  \hat{y}_\text{Nn} &= \text{Dt Nn} \\
  \hat{y}_\text{Vb} &= \text{Dt Vb}
  \end{align*} \]

- Compute losses and construct example:
  \[
  ( \{ w=\text{monster}, p=\text{Dt}, \ldots \}, \\
  [1,0,1] )
  \]
Example II: Graph labeling

- **Task:** label nodes of a graph given node features (and possibly edge features)
- **Example:** WebKB webpage labeling

- **Node features:** text on web page
- **Edge features:** text in hyperlinks
Example II: Graph labeling

- How to linearize? Like belief propagation might!
- Pick a starting node (A), run BFS out
- Alternate outward and inward passes

Linearization:

ABCDEFGHI
HGFEDCBA
BCDEFGHI
HGFEDCBA
\ldots
Example II: Graph labeling

1. Pick a node (= timestep)
2. Construct example based on neighbors' labels
3. Perturb current node's label to get losses