Reducing structured prediction to classification

Richard Johansson

Machine learning seminar
March 3, 2016
acknowledgement

- this talk has drawn inspiration from the *learning to search* tutorial by Hal Daumé III and John Langford
- see http://hunch.net/~l2s/
overview

introduction

reduction to classification

training the action classifier

better training
structured prediction problems

- In classification, we learn classifiers that outputs an atomic label $y$ for a given input $x$
  - and $y$ comes from a finite (and typically small) set $\mathcal{Y}$
- In this talk, we will go beyond classification and predict a non-atomic **data structure**
  - for instance a sequence, a tree, a DAG, ... 
- This type of problem is common in language processing, vision, biology, and other fields
example: sequence tagging

- input: a sequence of symbols (e.g. words, gene sequence)
- output: another sequence of the same length

United Nations official Ekeus heads for Baghdad .
B-ORG I-ORG O B-PER O O B-LOC O

A C A T G G T C T G A A
N N C C C C C C C C C C C C N
example: parsing a sentence

- input: a sequence of words
- output: a graph representing the grammatical relations

output from Stanford CoreNLP
a bit more formally

- in structured prediction,
  - the output space $\mathcal{Y}(x)$ is defined by the input $x$
  - $\mathcal{Y}(x)$ is huge
  - the outputs $y \in \mathcal{Y}(x)$ consist of distinct but interdependent parts
- we have a **loss function** $L(y, \hat{y})$ that compares a predicted output to a gold standard
  - e.g. the **Hamming loss** for sequences: the number of errors
  - **attachment errors** for parse trees
- we want our learning process to find a good predictor $f$, so that the expected loss is as low as possible:

$$\mathbb{E}_{(x,y)} L(f(x), y)$$
two high-level approaches

▶ structured learning algorithms: modify the learning algorithm to attack the complex problem directly
  ▶ perceptron → structured perceptron [Collins, 2002]
  ▶ SVM → structured SVM [Taskar et al., 2004]
  ▶ logistic regression → CRF [Lafferty et al., 2001]
  ▶ etc
two high-level approaches

- **structured learning algorithms**: modify the learning algorithm to attack the complex problem directly
  - perceptron $\rightarrow$ structured perceptron [Collins, 2002]
  - SVM $\rightarrow$ structured SVM [Taskar et al., 2004]
  - logistic regression $\rightarrow$ CRF [Lafferty et al., 2001]
  - etc

- **reduction to classification** – this talk!
overview

introduction

reduction to classification

training the action classifier

better training
reduction to classification: general ideas

- break down the complex prediction problem into a sequence of simple decisions
  - think of it as a system that gradually consumes input and generates output
  - while doing that, the system has some notion of what it is doing: a state
  - formally: a state machine
- in each state, we have a finite set of actions to choose from
  - so we can use standard classifiers to select the action
  - the classifiers use features from the input and from the state
example: sequence tagging

United Nations official Ekeus heads for Baghdad.
example: sequence tagging

United Nations official Ekeus heads for Baghdad.
example: sequence tagging

United Nations official Ekeus heads for Baghdad.
example: sequence tagging

United Nations official Ekeus heads for Baghdad.
example: sequence tagging

United Nations official Ekeus heads for Baghdad.
example: sequence tagging

United Nations official Ekeus heads for Baghdad .

B-ORG I-ORG O B-PER O
example: sequence tagging

United Nations official Ekeus heads for Baghdad .

B-ORG I-ORG O B-PER O O
example: sequence tagging

United Nations official Ekeus heads for Baghdad.
example: sequence tagging

United Nations official Ekeus heads for Baghdad .
B-ORG I-ORG O B-PER O O B-LOC O
example: parsing a sentence

- **input**: a sequence of words
- **output**: a graph representing the grammatical relations

![Diagram of a parsing example](image)

- **transition-based parsing**:
  - **state**: stack and queue, and the relations we’ve output so far
  - **classifier** selects an action that modifies the stack or queue, and outputs relations
transition-based parsing example

\[
S \quad Q
\]

\[
\langle D \rangle \text{Then we met the cat.}
\]
transition-based parsing example

\[
\begin{array}{c}
S \\
<\text{D}> \\
Q \\
\text{Then} \quad \text{we} \quad \text{met} \quad \text{the} \quad \text{cat} \quad .
\end{array}
\]
transition-based parsing example

$S$

$Q$

$<D>$ Then

we met the cat .
transition-based parsing example

\[ S \]
\[ \text{Then we} \]

\[ Q \]
\[ \text{met the cat .} \]
transition-based parsing example
transition-based parsing example

Then we
transition-based parsing example

Then we met the cat.
transition-based parsing example

Then we met the cat.
transition-based parsing example

Then we met the cat.
transition-based parsing example

Then we met the cat.
transition-based parsing example

The diagram illustrates a transition-based parsing example, where the transition rules are applied to a sentence to parse it. The sentence is "Then we met the cat." The diagram shows the parsing process step by step, starting with the initial state S and ending with the final state Q, indicating the sentence is parsed correctly.
transition-based parsing example
advantages

- speed!

- flexibility: we can define any feature we want from the input and the state, no Markov assumptions etc
  - ...except that we can’t use features from future states
but won’t the greedy decisions be a problem?

▶ not necessarily, if we have good lookahead features [Liang et al., 2008]
▶ or we can keep a list of the $k$ best sequences seen so far – **beam search**
overview

introduction

reduction to classification

training the action classifier

better training
training the action classifier

- how can we train the action classifier?

- simplest idea: learn from an expert
  - walk the state machine so that it generates the gold-standard outputs
  - collect the states we observe along the way: these are our training instances
- this is similar to imitation learning in robotics
training the action classifier

- how can we train the action classifier?
- simplest idea: learn from an expert
  - “walk” the state machine so that it generates the gold-standard outputs
  - collect the states we observe along the way: these are our training instances
- this is similar to imitation learning in robotics
simple learning from an expert: formalized

initialize training set $\mathcal{D} = \emptyset$
for $(x, y) \in (X, Y)$
   for each state visited by $\pi^*$ in $x$
       add instance $(s, \pi^*(s))$ to $\mathcal{D}$
train classifier $\hat{\pi}$ on $\mathcal{D}$
return $\hat{\pi}$

- the expert $\pi^*$ is also called the **optimal policy**
- it uses the gold-standard output $y$
example: sequence tagging (with features)

United Nations official Ekeus heads for Baghdad.

B-ORG I-ORG

↑

- let’s keep things simple and extract just two features from each state:
  - the tag from the previous step
  - the word at the current position

[ (’<START>’, ’United’), ’B-ORG’),
 (’B-ORG’, ’Nations’), ’I-ORG’),
 (’I-ORG’, ’official’), ’O’),
...
]
examples of this training strategy

▶ a lot of name taggers following Ratinov and Roth [2009]
▶ MaltParser [Nivre et al., 2007] – probably the most widely used multilingual parser
▶ ... and the more recent Stanford neural-net dependency parser [Chen and Manning, 2014]
overview

introduction

reduction to classification

training the action classifier

better training
why is this problematic?
why is this problematic?

- when we are guided by the expert, we get no training in the unseen parts of the search space!
example: playing Super Mario

[Ross and Bagnell, 2010] prove a theorem showing that the expected number of errors can grow \textit{quadratically} as a function of the sequence length, because of error compounding
letting go of expert guidance

- we can address this problem by gradually letting the learned action classifier explore the state space
- a few different variants of this idea:
  - SEARN [Daumé III et al., 2009]
  - DAgger [Ross et al., 2011]
  - AggreVaTe [Ross and Bagnell, 2014]
  - LOCLS [Chang et al., 2015]
DAgger and AggreVaTe (preliminaries)

- in each iteration $i$ we will train a classifier $\hat{\pi}_i$ – a policy
- we have some weight decay scheme $\beta_1, \beta_2, \ldots$ that controls the probability of using the expert to generate states as training progresses
  - for instance $1, p, p^2, \ldots$
- in each state $s$, we can ask the expert for the best action $\pi^*(s)$ – even if $s$ is a messed-up state!
DAgger (Ross et al., 2011)

initialize training set \( \mathcal{D} = \emptyset \)
\( \hat{\pi}_0 \) = dummy classifier
for \( i = 1, \ldots, N \)
    let \( \pi_i = \beta_i \pi^* + (1 - \beta_i) \hat{\pi}_{i-1} \)
    for \( (x, y) \in (X, Y) \)
        for each state \( s \) visited by \( \pi_i \) in \( x \)
            add instance \( (s, \pi^*(s)) \) to \( \mathcal{D} \)
    train classifier \( \hat{\pi}_i \) on \( \mathcal{D} \)
return \( \hat{\pi}_N \)
AggreVaTe (Ross et al., 2014)

initialize training set $\mathcal{D} = \emptyset$

$\hat{\pi}_0 =$ dummy classifier

for $i = 1, \ldots, N$

let $\pi_i = \beta_i \pi^* + (1 - \beta_i) \hat{\pi}_{i-1}$

for $(x, y) \in (X, Y)$

for each state $s$ visited by $\pi_i$ in $x$

for each action $a$ available in $s$

compute cost $c_a$ of $a$

add instance $(s, a, c_a)$ to $\mathcal{D}$

train cost-sensitive classifier $\hat{\pi}_i$ on $\mathcal{D}$

return $\hat{\pi}_N$

- the cost $c_a$ of action $a$ is the difference in loss between executing $a$ and the best action if we “roll out” using $\pi^*$
so what about $\pi^*$?

- as mentioned, $\pi^*(s)$ is the best action in $s$ even if $s$ is bad
- how easy is it in practice to determine $\pi^*(s)$?
- it depends – on the prediction problem and the loss function!
  - it is trivial for sequence tagging with Hamming loss
  - for transition-based parsing, it wasn’t known until recently [Goldberg and Nivre, 2013]
results: handwriting recognition (DAgger)

Figure 5: Character accuracy as a function of iteration.
Table 4. The UAS score on dependency parsing data set; columns are roll-out and rows are roll-in. The best result is bold. **SEARN** achieves 84.0, 81.1, and 63.4 when the reference policy is optimal, suboptimal, and bad, respectively. **LOLS** is Learned/Mixture and highlighted in green.
Results: training a sequence tagger (LOLS)
results: sequence tagging efficiency (LOLS)
try this at home

- **Vowpal Wabbit** is a highly efficient classification library
  - [http://hunch.net/~vw/](http://hunch.net/~vw/)
- and it includes the reductions I’ve discussed today for sequence tagging and parsing
- it also has C and Python APIs, so that you can plug in your own problem
  - for a new prediction problem, you would have to define the search space and the loss
- see more at the learning to search tutorial:
  - [http://hunch.net/~l2s/](http://hunch.net/~l2s/)
references I


References II


