Treeler: Open-source Structured Prediction for NLP

Xavier Carreras

Universitat Politècnica de Catalunya

thanks to: M. Collins, A. Globerson, T. Koo, N. Ata, P. S. Madhyastha

acknowledgements: Pascal2 Harvest Programme
An open-source package for linear structured prediction

Released under GNU General Public License

Focus on NLP problems:
  - Everything is structured
  - Everything is large, performance is critical
  - High overlap of components across tasks

Origins at MIT CSAIL (2006-2009)

Redesigned to be more flexible

C++, polymorphism via templates
An Application: Extracting Financial Relations

Mr. Wayne bought shares of Acme Corp.

- Read texts from the web. For a new text:
An Application: Extracting Financial Relations

Mr. Wayne bought shares of Acme Corp.

- Read texts from the web. For a new text:
  1. Classify according to financial or not.
    - Use a binary classifier using bag-of-words representations
An Application: Extracting Financial Relations

Mr. Wayne bought shares of Acme Corp.

- Read texts from the web. For a new text:
  1. Classify according to financial or not.
  2. Extract named entities (persons and organizations)
     - Use a sequence tagger
An Application: Extracting Financial Relations

* Mr. Wayne bought shares of Acme Corp.

- Read texts from the web. For a new text:
  1. Classify according to financial or not.
  2. Extract named entities (persons and organizations)
  3. Parse text and extract grammatical relations.
     - Use a probabilistic dependency parser
     - Compute syntactic paths linking entities, weighted by their probability
An Application: Extracting Financial Relations

Read texts from the web. For a new text:

1. Classify according to financial or not.
2. Extract named entities (persons and organizations)
3. Parse text and extract grammatical relations.
4. Classify each pair of entities.
   - Use a multiclass classifier deciding the type of relation
   - Use grammatical relations as features
An Application: Extracting Financial Relations

Mr. Wayne bought shares of Acme Corp.

Read texts from the web. For a new text:

1. Classify according to financial or not.
2. Extract named entities (persons and organizations)
3. Parse text and extract grammatical relations.
4. Classify each pair of entities.

Treeler provides core algorithms for learning and using classifiers, taggers and parsers.
Linear (Structured) Prediction

Classification

Sequence Tagging

Parsing
Classification

- Not really structured prediction
- Linear Multiclass Classification:
  - $\mathcal{X} = \mathbb{R}^d$, an input domain with $d$ features
  - $\mathcal{Y} = \{1, \ldots, L\}$, a set of classes
  - Define parameters $w_l \in \mathbb{R}^d$, for $1 \leq l \leq L$
  - Classify new points $x \in \mathcal{X}$ with:

$$\arg \max_{l=1,\ldots,L} w_l \cdot x$$

- Learning algorithms: Perceptron, SVM, Maximum Entropy
Structured Prediction: Sequence Tagging

\[ \hat{y}: \text{PER PER - - LOC} \]
\[ x: \text{Jack London went to Paris} \]

- Goal: given input sequence \( x \), predict sequence \( y \)

- Approach 1: local classifiers
  - A multiclass classifier to predict individual tags
    \[ \hat{y}_i = \arg\max_{l=1,\ldots,L} w \cdot f(x, i, l) \]
  - Best sequence = concatenate best tag for each word
    \[ \hat{y} = \arg\max_{y \in \mathcal{Y}(x)} \sum_i w \cdot f(x, i, l) \]
Structured Prediction: Sequence Tagging

\[ y: \quad \text{PER} \quad \text{PER} \quad - \quad - \quad \text{LOC} \]
\[ x: \quad \text{Jack} \quad \text{London} \quad \text{went} \quad \text{to} \quad \text{Paris} \]

- Goal: given input sequence \( x \), predict sequence \( y \)
- **Approach 1**: local classifiers (limited features)
- **Approach 2**: global classifier
  - Multiclass classifier to predict full tag sequences
    \[ \hat{y} = \text{argmax}_{y \in \{1, \ldots, L\}^n} w \cdot f(x, y) \]
  - Unrestricted features, but too expensive
Structured Prediction: Sequence Tagging

\[ y: \text{PER PER - - LOC} \]
\[ x: \text{Jack London went to Paris} \]

- **Goal:** given input sequence \( x \), predict sequence \( y \)
- **Approach 1:** local classifiers (limited features)
- **Approach 2:** global classifier (too expensive in general)
- **Approach 3:** factored global classifier
  - Factor \( y \) into bigrams of tags
  \[ \hat{y} = \arg\max_{y \in \mathcal{Y}^*} \sum_i w \cdot f(x, i, y_{i-1}, y_i) \]
  - Extended locality by extending scope of \( n \)-grams
  - Fast inference using Viterbi algorithm
Structured Prediction: Parsing

- Directed arcs represent dependencies between a head word and a modifier word.

- E.g.:
  - shares modifies bought,
  - Wayne modifies bought,
  - Mr. modifies Wayne
Dependency Parsing: arc-factored models

(McDonald et al. 2005)

Parse trees decompose into single dependencies \( \langle h, m \rangle \)

\[
\arg\max_{y \in \mathcal{Y}(x)} \sum_{\langle h, m \rangle \in y} w \cdot f(x, h, m)
\]

Some features:
\[
\begin{align*}
    f_1(x, 3, 4) &= \left[ \text{"bought"} \rightarrow \text{"shares"} \right] \\
    f_2(x, 3, 4) &= \left[ \text{distance} = +1 \right]
\end{align*}
\]

Tractable inference exists (e.g. variants of CKY)
Linear Structured Prediction

- **Classification**
  \[
  \arg\max_{y \in \{1, \ldots, L\}} w \cdot f(x, y)
  \]

- **Sequence prediction (bigram factorization)**
  \[
  \arg\max_{y \in Y(x)} \sum_i w \cdot f(x, i, y_{i-1}, y_i)
  \]

- **Dependency parsing (arc factorization)**
  \[
  \arg\max_{y \in Y(x)} \sum_{\langle h, m \rangle \in y} w \cdot f(x, h, m)
  \]

- **In general, we can enumerate parts** \( r \in y \)
  \[
  \arg\max_{y \in Y(x)} \sum_{r \in y} w \cdot f(x, r)
  \]
Linear Structured Prediction Framework

- Generic Structured Prediction
  - Input domain $\mathcal{X}$, output domain $\mathcal{Y}$
  - A choice of factorization, $r \in \mathcal{Y}$
  - Features: $f(x, r) \rightarrow \mathbb{R}^d$

- The linear prediction model, with $w \in \mathbb{R}^d$

$$\argmax_{y \in \mathcal{Y}(x)} \sum_{r \in y} w \cdot f(x, r)$$

- Inference, i.e. how to solve the argmax?
  - Depends on the factorization

- Learning, i.e. how to obtain $w$?
  - Perceptron, SVM, CRF
  - Generic with respect to factorization
Structured Prediction Framework

Factorizations

Features

Inference

Learning
Factorizations

- $X$ : a generic type for input patterns
- $Y$ : a generic type for output structures

- $R$ is a factorization providing:
  - $r_t$ : a type for parts
  - $\text{parts}(x,y)$ : the set of parts in $x$ and $y$
  - $\text{parts}(x)$ : the set of parts assignable to $x$
Factorizations: enumerating parts

(tagging via bigram factorizations)

\[ r_t = \text{tuple of} \]
\[ \text{int } i; \quad // \text{position of bigram} \]
\[ \text{tag } a; \quad // \text{tag at } i-1 \]
\[ \text{tag } b; \quad // \text{tag at } i \]

<table>
<thead>
<tr>
<th>parts(x,y)</th>
<th>parts(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, - ,PER)</td>
<td>(1, - , - , - )</td>
</tr>
<tr>
<td>(2,PER,PER)</td>
<td>(1, - ,PER)</td>
</tr>
<tr>
<td>(3,PER, - )</td>
<td>(1, - ,LOC)</td>
</tr>
<tr>
<td>(4, - , - )</td>
<td>(2, - , - )</td>
</tr>
<tr>
<td>(5, - ,LOC)</td>
<td>(2, - ,PER)</td>
</tr>
<tr>
<td>(2, - ,LOC)</td>
<td></td>
</tr>
<tr>
<td>(2,PER,PER)</td>
<td></td>
</tr>
</tbody>
</table>

(parsing via arc factorizations)

\[ r_t = \text{tuple of} \]
\[ \text{int } h; \quad // \text{position of head} \]
\[ \text{int } m; \quad // \text{position of mod} \]

<table>
<thead>
<tr>
<th>parts(x,y)</th>
<th>parts(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(*,3)</td>
<td>(*,1)</td>
</tr>
<tr>
<td>(3,2)</td>
<td>(*,2)</td>
</tr>
<tr>
<td>(3,4)</td>
<td>(*,3)</td>
</tr>
<tr>
<td>(2,1)</td>
<td>(*,4)</td>
</tr>
<tr>
<td>(4,5)</td>
<td>(*,5)</td>
</tr>
<tr>
<td>(5,7)</td>
<td>(*,6)</td>
</tr>
<tr>
<td>(7,6)</td>
<td>(*,7)</td>
</tr>
</tbody>
</table>

\[ \ldots \]

\[ \ldots \]
Scores

- Scores<\(X,R\)> provides scores for parts
  - \(\text{score}(x,r)\): score of part \(r\) assigned to \(x\)

- We can define the following generic algorithm:
  
  ```
  function score(X x, Y y, Score<X,R> s)
      sum = 0
      foreach r in parts(x,y)
          sum += s.score(x,r)
      return sum
  ```
Scores, Features and Parameters

- **Features\(X,R\)** provides feature vectors for parts
  - \(\text{fvec}_t\): a type for feature vectors
  - \(f(x,r)\): the \(\text{fvec}\) for \(r\) assigned to \(x\)

- **WFScores\(X,R,F\)**: implements a scorer based on features
  - \(w_t\): a type for parameters
  - \(\text{score}(x,r)\): the inner product of \(f(x,r)\) and \(w\)

- The form of WFScores can be tailored to \(R\) and \(F\)
  - Sparse or dense \(\text{fvec}_t\) and \(w_t\)
  - Polymorphic inner products
Inference

- Inference\(<X,Y,R>\) provides inference algorithms
  - Let \(s\) be a scoring of type Scores\(<X,R>\)
  - \(\text{max}(x,s)\) computes the best structure for \(x\), i.e.
    \[
    \hat{y} = \arg\max_{y \in \mathcal{Y}(x)} \sum_{r \in y} \text{score}(x, r)
    \]
  - \(\text{partition}(x,s)\) computes the partition function for \(x\), i.e.
    \[
    Z = \sum_{y \in \mathcal{Y}(x)} \exp \left\{ \sum_{r \in y} \text{score}(x, r) \right\}
    \]
  - \(\text{sum}(x,s)\) computes marginals for parts, i.e.
    \[
    \mu(r) = \sum_{y \in \mathcal{Y}(x): r \in y} \exp \left\{ \sum_{r \in y} \text{score}(x, r) \right\} \ast Z^{-1}
    \]
- Actual implementations depend on \(Y\) and \(R\)
Learners

- Learner<X,Y,R>, a concept class for learning algorithms
  - learn(trainset, params): learns a weight vector from a training set

- A learner will use the following components, implicitly defined by X, Y and R:
  - Features<X,R>
  - WFScores<X,R,Features<X,R>>
  - Inference<X,Y,R>

- Available methods: Perceptron, MaxMargin, LogLinear
Averaged Perceptron (Freund and Schapire ’98, Collins ’03)

function Perceptron<X,Y,R>(trainset, T)
    typedef WFScores<X,R,Features<X,R>> WF_t;
    WF_t::w_t w = 0;       // initialize weights
    WF_t::w_t wavg = 0;    // initialize averaged weights
    for t = 1 .. T
        foreach (x,y) in trainset
            // create scorer for x using w
            WF_t scores(w,x);
            // get max solution under w
            Y z = Inference<X,Y,R>::max(x, scores);
            // update w
            if (z != y)
                foreach r in parts(x,y)
                    w = w + Features<X,R>::f(x,r);
                foreach r in parts(x,z)
                    w = w - Features<X,R>::f(x,r)
            // update averaged w
            wavg = wavg + w
    return (w,wavg)
Learners in Treeler

```
Learner<X,Y,R>
F: Features<X,R>
WF: WFscore<X,R,F>
I: Inference<X,Y,R>
+learn(trainset, params): WF::w_t

DualExpGradient<X,Y,R,O>
+objective: 0
<<requires>> max()
<<requires>> sum()
+learn(trainset, params): WF::w_t

Perceptron<X,Y,R>
<<requires>> I::max()

LogLinearEG<X,Y,R>
<<requires>> I::max()
<<requires>> I::sum()
O=LogLinearObjective

Pegasos<X,Y,R>
<<requires>> I::max()

MaxMarginEG<X,Y,R>
<<requires>> I::max()
<<requires>> I::sum()
O=MaxMarginObjective
```
<table>
<thead>
<tr>
<th></th>
<th>X</th>
<th>Y</th>
<th>R</th>
<th>I::max</th>
<th>I::sum</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class.</strong></td>
<td>$\mathbb{R}^d$</td>
<td>${1, \ldots, L}$</td>
<td>1</td>
<td>one-vs-all</td>
<td>explicit</td>
</tr>
<tr>
<td></td>
<td>$\mathbb{R}^d$</td>
<td>${1, \ldots, L}$</td>
<td>1,1'</td>
<td>pairwise</td>
<td>explicit</td>
</tr>
<tr>
<td><strong>tagging</strong></td>
<td>sent.</td>
<td>$L^*$</td>
<td>2-gram</td>
<td>Viterbi&lt;1&gt;</td>
<td>FwdBack&lt;1&gt;</td>
</tr>
<tr>
<td></td>
<td>sent.</td>
<td>$L^*$</td>
<td>3-gram</td>
<td>Viterbi&lt;2&gt;</td>
<td>FwdBack&lt;2&gt;</td>
</tr>
<tr>
<td><strong>parsing</strong></td>
<td>sent.</td>
<td>proj.</td>
<td>h,m</td>
<td>Eisner&lt;1&gt;</td>
<td>IO-Eisner&lt;1&gt;</td>
</tr>
<tr>
<td></td>
<td>sent.</td>
<td>non-proj.</td>
<td>h,m</td>
<td>C-L-E</td>
<td>matrix-tree</td>
</tr>
<tr>
<td></td>
<td>sent.</td>
<td>proj.</td>
<td>h,m,c</td>
<td>Eisner&lt;2&gt;</td>
<td>IO-Eisner&lt;2&gt;</td>
</tr>
</tbody>
</table>

+ feature functions commonly used in the state-of-the-art

+ methods for reading/writing using standard formats

+ scripts for training models and running them on new data
experiments:

Dependency Parsing
Comparing Learners for Dependency Parsing

Dataset: English “WSJ” Penn Treebank

![Validation Accuracy Graph]

<table>
<thead>
<tr>
<th></th>
<th>Averaged Perceptron</th>
<th>LogLinear</th>
<th>LogLinear</th>
<th>MaxMargin</th>
<th>MaxMargin</th>
</tr>
</thead>
<tbody>
<tr>
<td>C=0.001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C=0.01</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C=0.0001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>R</th>
<th>I: :max</th>
<th>I: :sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>sent.</td>
<td>proj.</td>
<td>h,m</td>
<td>Eisner&lt;1&gt;</td>
<td>IO-Eisner&lt;1&gt;</td>
</tr>
</tbody>
</table>

- **Learners:** Perceptron vs. LogLinear vs. MaxMargin
Comparing Factorizations for Dependency Parsing

Dataset: English “WSJ” Penn Treebank

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>R</th>
<th>I::max</th>
<th>I::sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>sent.</td>
<td>proj.</td>
<td>h,m</td>
<td>Eisner&lt;1&gt;</td>
<td>IO-Eisner&lt;1&gt;</td>
</tr>
<tr>
<td>sent.</td>
<td>proj.</td>
<td>h,m,c</td>
<td>Eisner&lt;2&gt;</td>
<td>IO-Eisner&lt;2&gt;</td>
</tr>
</tbody>
</table>

- Learner: Averaged Perceptron
CoNLL-2007: Multilingual Dependency Parsing

![Graph showing performance of different languages in multilingual dependency parsing. The x-axis represents languages: Arabic, Basque, Catalan, Chinese, Czech, English, Greek, Hungarian, Italian, and Turkish. The y-axis represents the percentage score, ranging from 0 to 100. The graph compares the performance of Treeler (blue bars) and the best result per language (red bars).]
Treeler: Summary

- Open-source library for Structured Prediction
  http://nlp.lsi.upc.edu/treeler

- Focus: tagging and parsing in NLP

- Abstract interfaces between models and learners:
  - New models can be easily plugged to learners
  - New learners can be used across different structured tasks

- C++ templates, effective and efficient polymorphism