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

http://nlp.lsi.upc.edu/treeler

- 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.

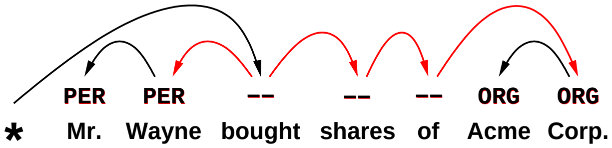
- Read texts from the web. For a new text:
  1. Classify according to financial or not.
     - Use a binary classifier using bag-of-words representations

An Application: Extracting Financial Relations

PER	PER	--	--	--	ORG	ORG
Mr.	**Wayne**	**bought**	**shares**	**of**	**Acme**	**Corp.**

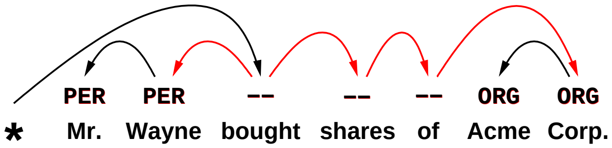
- Read texts from the web. For a new text:
  1. Classify according to financial or not.
  2. Extract named entities (persons and organizations)
     - Use a sequence tagger

An Application: Extracting Financial Relations



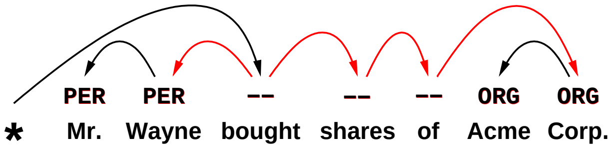
- Read texts from the web. For a new text:
  1. Classify according to financial or not.
  2. Extract named entities (persons and organizations)
  3. Parse text and extract grammatical relations.
     - Use a probabilistic dependency parser
     - Compute syntactic paths linking entities, weighted by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



- 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
  - $\mathcal{Y} = \{1, ..., L\}$, a set of classes
  - Define parameters $\mathbf{w}_l \in \mathbb{R}^d$, for $1 \leq l \leq L$
  - Classify new points $\mathbf{x} \in \mathcal{X}$ with:

$$\operatorname*{argmax}_{l=1,...,L} \quad \mathbf{w}_l \cdot \mathbf{x}$$

- Learning algorithms: Perceptron, SVM, Maximum Entropy

Structured Prediction: Sequence Tagging

$\mathbf{y}$:	PER	PER	-	-	LOC
$\mathbf{x}$:	Jack	London	went	to	Paris

- Goal: given input sequence $\mathbf{x}$, predict sequence $\mathbf{y}$
- Approach 1: local classifiers
  - A multiclass classifier to predict individual tags

$$\hat{y}_i = \operatorname*{argmax}_{l=1,\ldots,L} \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, i, l)$$

  - Best sequence = concatenate best tag for each word

$$\hat{\mathbf{y}} = \operatorname*{argmax}_{\mathbf{y} \in \mathcal{Y}(x)} \sum_i \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, i, l)$$

Structured Prediction: Sequence Tagging

$\mathbf{y}$:	PER	PER	-	-	LOC
$\mathbf{x}$:	Jack	London	went	to	Paris

- Goal: given input sequence $\mathbf{x}$, predict sequence $\mathbf{y}$
- Approach 1: local classifiers (limited features)
- Approach 2: global classifier
  - Multiclass classifier to predict full tag sequences

$$\hat{\mathbf{y}} = \underset{y=\{1,\ldots,L\}^n}{\operatorname{argmax}} \ \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, \mathbf{y})$$

  - Unrestricted features, but too expensive

Structured Prediction: Sequence Tagging

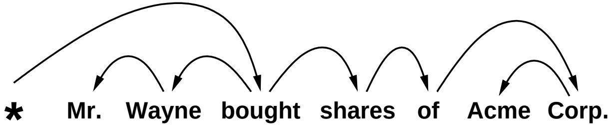
$\mathbf{y}$:	PER	PER	-	-	LOC
$\mathbf{x}$:	Jack	London	went	to	Paris

- Goal: given input sequence $\mathbf{x}$, predict sequence $\mathbf{y}$
- Approach 1: local classifiers (limited features)
- Approach 2: global classifier (too expensive in general)
- Approach 3: factored global classifier
  - Factor $\mathbf{y}$ into bigrams of tags

$$\hat{\mathbf{y}} = \operatorname*{argmax}_{\mathbf{y} \in \mathcal{Y}^*} \sum_i \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, i, \mathbf{y}_{i-1}, \mathbf{y}_i)$$

  - Extended locality by extending scope of n-grams
  - Fast inference using Viterbi algorithm

Structured Prediction: Parsing

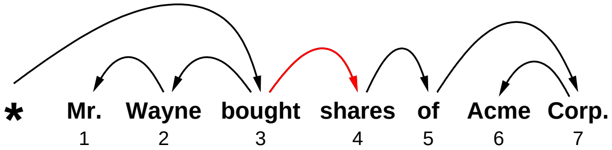


- Directed arcs represent dependencies between a head word and a modifier word.
- E.g.:

  shares *modifies* bought,
  Wayne *modifies* bought,
  Mr. *modifies* Wayne

Dependency Parsing: arc-factored models

(McDonald et al. 2005)



- Parse trees decompose into single dependencies $\langle h, m \rangle$

$$\underset{\mathbf{y} \in \mathcal{Y}(\mathbf{x})}{\operatorname{argmax}} \sum_{\langle h,m \rangle \in y} \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, h, m)$$

- Some features: $\mathbf{f}_1(\mathbf{x}, 3, 4) = [\ \text{"bought"} \ \rightarrow \text{"shares"}\ ]$
  $\mathbf{f}_2(\mathbf{x}, 3, 4) = [\ \text{distance} = +1\ ]$
- Tractable inference exists (e.g. variants of CKY)

Linear Structured Prediction

- Classification

$$\operatorname*{argmax}_{\mathbf{y} \in \{1, \ldots, L\}} \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, \mathbf{y})$$

- Sequence prediction (bigram factorization)

$$\operatorname*{argmax}_{\mathbf{y} \in \mathcal{Y}(\mathbf{x})} \sum_{i} \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, i, \mathbf{y}_{i-1}, \mathbf{y}_i)$$

- Dependency parsing (arc factorization)

$$\operatorname*{argmax}_{\mathbf{y} \in \mathcal{Y}(\mathbf{x})} \sum_{\langle h, m \rangle \in y} \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, h, m)$$

- In general, we can enumerate parts $r \in \mathbf{y}$

$$\operatorname*{argmax}_{\mathbf{y} \in \mathcal{Y}(\mathbf{x})} \sum_{r \in \mathbf{y}} \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, r)$$

Linear Structured Prediction Framework

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

$$\underset{\mathbf{y} \in \mathcal{Y}(\mathbf{x})}{\operatorname{argmax}} \sum_{r \in y} \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, r)$$

- Inference, i.e. how to solve the argmax?
  - Depends on the factorization
- Learning, i.e. how to obtain $\mathbf{w}$?
  - Perceptron, SVM, CRF
  - Generic with respect to factorization

Structured Prediction Framework

Factorizations

Features

Inference

Learning

Factorizations

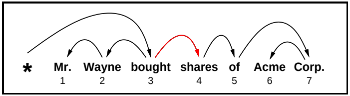
- `X` : a generic type for input patterns
- `Y` : a generic type for output structures

- `R` is a factorization providing:
  - `r_t` : a type for parts
  - `parts(x,y)` : the set of parts in `x` and `y`
  - `parts(x)` : the set of parts assignable to `x`

Factorizations: enumerating parts

PER	PER	-	-	LOC
Jack	London	went	to	Paris
1	2	3	4	5

(tagging via bigram factorizations)



(parsing via arc factorizations)

```
r_t = tuple of
  int i; // position of bigram
  tag a; // tag at i-1
  tag b; // tag at i
```

```
r_t = tuple of
  int h; // position of head
  int m; // position of mod
```

parts(x,y)	parts(x)
(1, - ,PER)	(1, - , -)
(2,PER,PER)	(1, - ,PER)
(3,PER, -)	(1, - ,LOC)
(4, - , -)	(2, - , -)
(5, - ,LOC)	(2, - ,PER)
	(2, - ,LOC)
	(2,PER,PER)
	...

parts(x,y)	parts(x)
(∗,3)	(∗,1)
(3,2)	(∗,2)
(3,4)	(∗,3)
(2,1)	(∗,4)
(4,5)	(∗,5)
(5,7)	(∗,6)
(7,6)	(∗,7)
	...

Scores

- `Scores<X,R>` provides scores for parts
  - `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
  - `fvec_t` : a type for feature vectors
  - `f(x,r)` : the `fvec` for `r` assigned to `x`
- `WFScores<X,R,F>` : implements a scorer based on features
  - `w_t` : a type for parameters
  - `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 `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>`
  - `max(x,s)` computes the best structure for `x`, i.e.

$$\hat{\mathbf{y}} = \operatorname*{argmax}_{\mathbf{y} \in \mathcal{Y}(\mathbf{x})} \sum_{r \in \mathbf{y}} \mathrm{score}(\mathbf{x}, \mathrm{r})$$

  - `partition(x,s)` computes the partition function for `x`, i.e.

$$Z = \sum_{\mathbf{y} \in \mathcal{Y}(\mathbf{x})} \exp \left\{ \sum_{r \in \mathbf{y}} \mathrm{score}(\mathbf{x}, \mathrm{r}) \right\}$$

  - `sum(x,s)` computes marginals for parts, i.e.

$$\mu(r) = \sum_{\mathbf{y} \in \mathcal{Y}(\mathbf{x}) : r \in \mathbf{y}} \exp \left\{ \sum_{r \in \mathbf{y}} \mathrm{score}(\mathbf{x}, \mathrm{r}) \right\} * Z^{-1}$$

- Actual implementations depend on `Y` and `R`

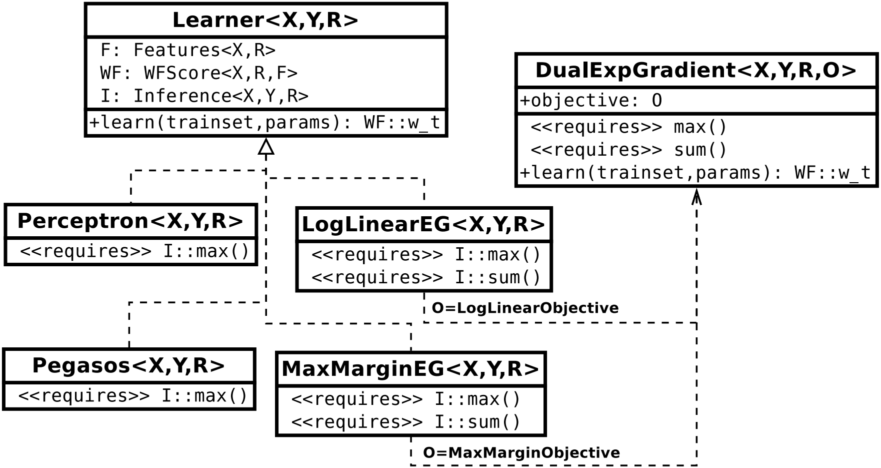
Learners

- `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



Structured Prediction Models in Treeler

	X	Y	R	I::max	I::sum
class.	$\mathbb{R}^d$	$\{1,\ldots,L\}$	l	one-vs-all	explicit
	$\mathbb{R}^d$	$\{1,\ldots,L\}$	l,l'	pairwise	explicit
tagging	sent.	L^*	2-gram	Viterbi<1>	FwdBack<1>
	sent.	L^*	3-gram	Viterbi<2>	FwdBack<2>
parsing	sent.	proj.	h,m	Eisner<1>	IO-Eisner<1>
	sent.	non-proj.	h,m	C-L-E	matrix-tree
	sent.	proj.	h,m,c	Eisner<2>	IO-Eisner<2>

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

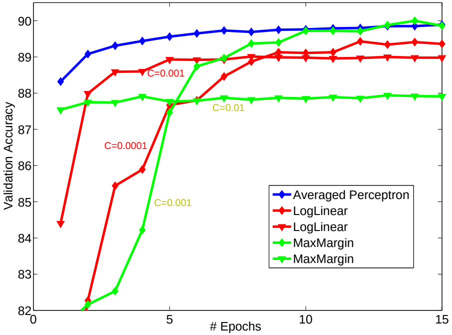
\+ methods for reading/writing using standard formats

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

Dependency Parsing

Comparing Learners for Dependency Parsing

Dataset: English "WSJ" Penn Treebank

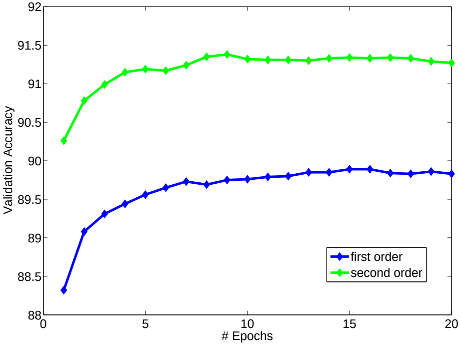


X	Y	R	I::max	I::sum
sent.	proj.	h,m	Eisner<1>	IO-Eisner<1>

- Learners: Perceptron vs. LogLinear vs. MaxMargin

Comparing Factorizations for Dependency Parsing

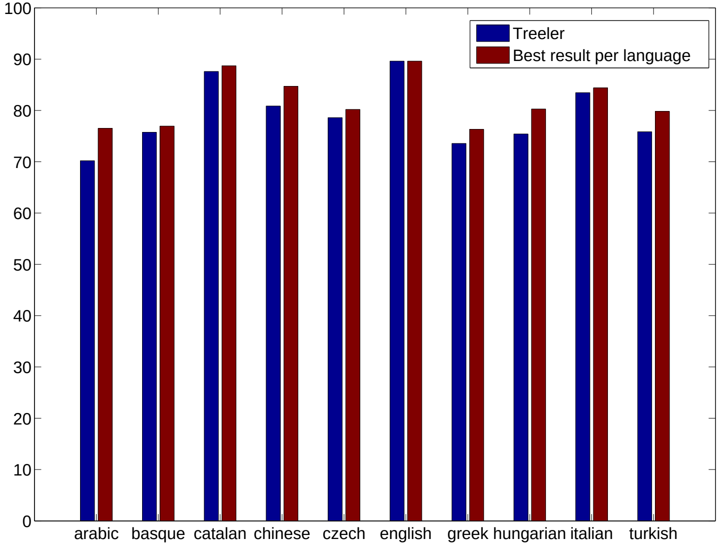
Dataset: English "WSJ" Penn Treebank



X	Y	R	I::max	I::sum
sent.	proj.	h,m	Eisner<1>	IO-Eisner<1>
sent.	proj.	h,m,c	Eisner<2>	IO-Eisner<2>

- Learner: Averaged Perceptron

CoNLL-2007: Multilingual Dependency Parsing



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 accross different structured tasks
- C++ templates, effective and efficient polymorphism