Machine Learning for NLP
Lecture 4: Structured prediction

UNIVERSITY OF GOTHENBURG

Richard Johansson

September 21, 2015

overview

multiclass linear classifiers

structured problems: overview

step-by-step structured prediction

task-specific learning

information about assignment 3

UNIVERSITY OF GOTHENBURG

two-class (binary) linear classifiers

- a **linear classifier** is a classifier that is defined in terms of a scoring function like this

$$\text{score} = \boldsymbol{w} \cdot \boldsymbol{x}$$

- this is a **binary** (2-class) classifier:
  - return the first class if the score > 0
  - ... otherwise the second class
- how can we deal with non-binary (**multi-class**) problems when using linear classifiers?

UNIVERSITY OF GOTHENBURG

decomposing multi-class classification problems

- **idea 1**: break down the complex problem into simpler problems, train a classifier for each
- **one-versus-rest** ("long jump"):
  - for each class c, make a binary classifier to distinguish c from all other classes
  - so if there are n classes, there are n classifiers
  - at test time, we select the class giving the highest score
- **one-versus-one** ("football league"):
  - for each pair of classes c_1 and c_2, make a classifier to distinguish c_1 from c_2
  - if there are n classes, there are $\frac{n \cdot (n-1)}{2}$ classifiers
  - at test time, we select the class that has most "wins"

example

- assume we're training a function tagger and we have the classes `SBJ`, `OBJ`, `ADV`
- in one-vs-rest, we train the following three classifiers:
  - `SBJ` vs `OBJ`+`ADV`
  - `OBJ` vs `SBJ`+`ADV`
  - `ADV` vs `SBJ`+`OBJ`
- in one-vs-one, we train the following three:
  - `SBJ` vs `OBJ`
  - `SBJ` vs `ADV`
  - `OBJ` vs `ADV`

in scikit-learn

- scikit-learn includes implementations of both of the methods we have discussed:
  - `OneVsRestClassifier`
  - `OneVsOneClassifier`
- however, the built-in algorithms (e.g. `Perceptron`, `LinearSVC`) will do this automatically for you
  - they use one-versus-rest

multiclass learning algorithms

- is it good to separate the multiclass task into smaller tasks that are trained independently?
- maybe training should be similar to testing?
- idea 2: make a perceptron where one-vs-rest is used while training

multiclass perceptron

- in the **multiclass perceptron**, we make error-driven updates just like in the binary perceptron
- we keep one weight vector w_y for each class y
- assume that we have misclassified an instance x: the true class was y but we guessed g
  - then we add the features of x to w_y, and subtract from w_g

$\mathbf{w}_y = (0, \ldots, 0)$ for each class y
repeat N times
  for $(\mathbf{x}_i, y_i)$ in the training set $\mathcal{T}$
    $g = \arg\max_y \mathbf{w}_y \cdot \mathbf{x}_i$
    if g is not equal to y_i
      $\mathbf{w}_{y_i} = \mathbf{w}_{y_i} + \mathbf{x}_i$
      $\mathbf{w}_g = \mathbf{w}_g - \mathbf{x}_i$
return $w_1, \ldots$

UNIVERSITY OF GOTHENBURG

in NumPy/scikit-learn (dense vectors)

```
class MulticlassPerceptron():
    # some initialization...

    def fit(self, X, Y):
        # some initialization...

        self.ws = numpy.zeros( (n_classes, n_features) )

        for i in range(self.n_iter):
            for x, y in XY:
                scores = numpy.dot(self.ws, x)
                guess = scores.argmax()
                if guess != y:
                    self.ws[y] += x
                    self.ws[guess] -= x
```

multiclass LR and SVM

- this idea can also be used when training SVM and LR
- in the Pegasos paper, the last two rows in the table on page 15 correspond to multiclass SVM and LR, respectively
- notice the similarity of the multiclass SVM update to the multiclass perceptron!
- in scikit-learn:
  - `LinearSVC(multi_class='crammer_singer')`
  - `LogisticRegression(multi_class='multinomial')`

overview

multiclass linear classifiers

structured problems: overview

step-by-step structured prediction

task-specific learning

information about assignment 3

this lecture

- in many NLP tasks, the output is not just a category: it's a complex object
  - sequences, trees, translations, ...
- **structured prediction**: for an input x, the set of possible outputs
  - depends on the input x
  - is very large – typically exponential in the size of x
  - consists of many small but interdependent parts
- this type of problem is central in NLP but not so often taught in machine learning courses!
  - ... many of the most important ideas were invented by NLP researchers

UNIVERSITY OF GOTHENBURG

example: sequences

- input: a sequence (for instance, words)
- output: another sequence (for instance, PoS tags)

The	**rain**	**falls**	**hard**	**.**
DT	**NN**	**VBZ**	**JJ**	**.**

example: dependency parse trees

s = <D> Lisa walks home

$t^1 =$

$t^2 =$

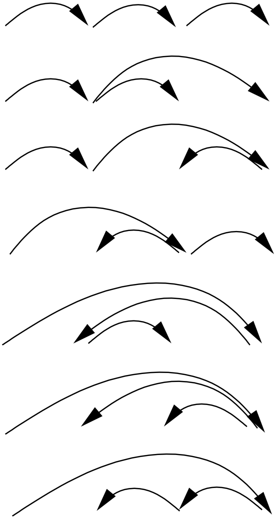
$t^3 =$

$t^4 =$

$t^5 =$

$t^6 =$

$t^7 =$



example: noun phrase coreference

- input: a document, and a list of the noun phrases in the document
- output: ?

example: temporal structure

- input: a document, and a list of the events in the document
- output: ?

two high-level approaches

- **step-by-step prediction**:
  - break down the complex prediction problem into a sequence of simple decisions
  - train a classifier to select the right decision at each step
  - example: transition-based parsing (Nivre)
- **task-specific learning**: modify the learning algorithm so that it handles the complex problem directly
  - for instance, we can make a "parsing perceptron"
  - example: graph-based parsing (McDonald)

overview

multiclass linear classifiers

structured problems: overview

step-by-step structured prediction

task-specific learning

information about assignment 3

step-by-step prediction: general ideas

- break down the problem into a sequence of smaller 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**
- use standard classifiers to guess the next step
  - the classifiers use features from the input and from the state
  - we might need to constrain the classifiers to keep the output consistent

example: part-of-speech tagging

- input: words
- output: tags
- state: position in sentence; previous outputs

	The	rain	falls	hard	.
(START)					

example: part-of-speech tagging

- input: words
- output: tags
- state: position in sentence; previous outputs

	The	**rain**	**falls**	**hard**	**.**
(START)	**DT**				

example: part-of-speech tagging

- input: words
- output: tags
- state: position in sentence; previous outputs

	The	**rain**	**falls**	**hard**	**.**
(START)	**DT**	**NN**			

example: part-of-speech tagging

- input: words
- output: tags
- state: position in sentence; previous outputs

	The	rain	falls	hard	.
(START)	DT	NN	VBZ		

example: part-of-speech tagging

- input: words
- output: tags
- state: position in sentence; previous outputs

	The	**rain**	**falls**	**hard**	**.**
(START)	**DT**	**NN**	**VBZ**	**JJ**	

example: part-of-speech tagging

- input: words
- output: tags
- state: position in sentence; previous outputs

	The	rain	falls	hard	.
(START)	DT	NN	VBZ	JJ	.

case study: SVMTool

- we can extract features from the history (the previous decisions), but not from the future
- see the paper *Fast and Accurate Part-of-Speech Tagging: The SVM Approach Revisited* by Giménez and Màrquez (2004)
- note how they use features for the tags to the right
  - "the next tag has the ambiguity class `VBZ|NNS`"
  - "the next tag might be `VBZ`"

example: named entity tagging

- input: a sentence
- output: names in the sentence bracketed and labeled

United Nations official Ekeus heads for Baghdad.
[ORG] [PER] [LOC]

example: named entity tagging

- input: a sentence
- output: names in the sentence bracketed and labeled

United Nations official Ekeus heads for Baghdad.
[ORG] [PER] [LOC]

- typical solution: Beginning/Inside/Outside coding

United Nations official Ekeus heads for Baghdad .

example: named entity tagging

- input: a sentence
- output: names in the sentence bracketed and labeled

United Nations official Ekeus heads for Baghdad.

[ORG] [PER] [LOC]

- typical solution: Beginning/Inside/Outside coding

United	**Nations**	**official**	**Ekeus**	**heads**	**for**	**Baghdad**	**.**
B-ORG							

example: named entity tagging

- input: a sentence
- output: names in the sentence bracketed and labeled

United Nations official Ekeus heads for Baghdad.
[ORG] [PER] [LOC]

- typical solution: Beginning/Inside/Outside coding

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

example: named entity tagging

- input: a sentence
- output: names in the sentence bracketed and labeled

United Nations official Ekeus heads for Baghdad.
[ORG] [PER] [LOC]

- typical solution: Beginning/Inside/Outside coding

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

example: named entity tagging

- input: a sentence
- output: names in the sentence bracketed and labeled

United Nations official Ekeus heads for Baghdad.
[ORG] [PER] [LOC]

- typical solution: Beginning/Inside/Outside coding

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

example: named entity tagging

- input: a sentence
- output: names in the sentence bracketed and labeled

United Nations official Ekeus heads for Baghdad.
[ORG] [PER] [LOC]

- typical solution: Beginning/Inside/Outside coding

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

example: named entity tagging

- input: a sentence
- output: names in the sentence bracketed and labeled

United Nations	official	Ekeus	heads for	Baghdad	.
[ORG]		[PER]		[LOC]	

- typical solution: Beginning/Inside/Outside coding

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

example: named entity tagging

- input: a sentence
- output: names in the sentence bracketed and labeled

United Nations official Ekeus heads for Baghdad.
[ORG] [PER] [LOC]

- typical solution: Beginning/Inside/Outside coding

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

example: named entity tagging

- input: a sentence
- output: names in the sentence bracketed and labeled

United Nations official Ekeus heads for Baghdad.
[ORG] [PER] [LOC]

- typical solution: Beginning/Inside/Outside coding

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

- see Ratinov and Roth: *Design challenges and misconceptions in named entity recognition*. CoNLL 2009.

recap: transition-based parsing

- input: words
- output: dependency edges
- state: stack and queue, and the edges we've output so far

transition-based parsing example

S *Q*

		<D>	Then	we	met	the	cat	.

UNIVERSITY OF GOTHENBURG

transition-based parsing example

S

<D>

Q

Then	we	met	the	cat	.

transition-based parsing example

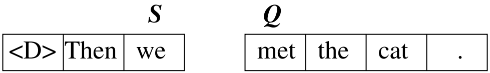
S

<D>	Then

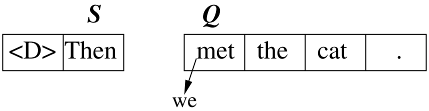
Q

we	met	the	cat	.

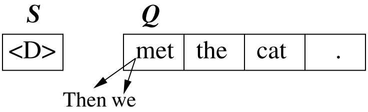
transition-based parsing example



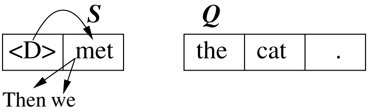
transition-based parsing example



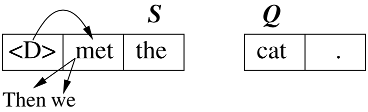
transition-based parsing example



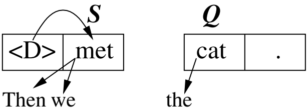
transition-based parsing example



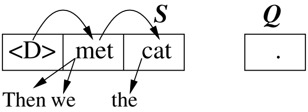
transition-based parsing example



transition-based parsing example

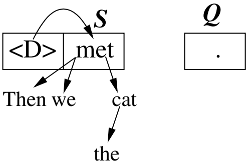


transition-based parsing example

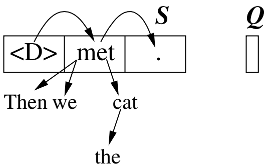


UNIVERSITY OF GOTHENBURG

transition-based parsing example



transition-based parsing example



discussion: noun phrase coreference

- the problem:
  - input: a document and its noun phrases
  - output: (what we discussed before)
- how can we solve the problem in a step-by-step fashion?

discussion: temporal structure

- the problem:
  - input: a document and its events
  - output: (what we discussed before)
- how can we solve the problem in a step-by-step fashion?

training step-by-step systems: the basic approach

- how can we train the decision classifier in a step-by-step system?

training step-by-step systems: the basic approach

- how can we train the decision classifier in a step-by-step system?
- the simple approach: learn to imitate an expert
  - force the system to generate the correct output
  - observe the states we pass along the way
  - use them as examples and train the classifier

what happens when we just try to imitate the expert?

image: Ross, Gordon, and Bagnell (2011)

training step-by-step systems: more carefully

- a problem: if we just generate states perfectly, our training set contains no examples of bad states
  - ...so we don't learn to recover from errors!
- one solution: mix expert-generated states with automatically generated states in the training set
- further reading:
  - `http://hunch.net/~l2s/`
  - SEARN: Daumé III et al: *Search-based structured prediction*, Machine Learning Journal, 2009.
  - DAGGER: Ross et al: *A reduction of imitation learning and structured prediction to no-regret online learning*, AISTATS 2011.

going beyond greedy decisions

- step-by-step systems tend to be **greedy**: they select what looks like the best decision at the moment, and can't regret that decision
- **beam search** can reduce this problem: keep track of the N top-scoring results, not just one
  - for coding details: see instructions for VG part of assignment
  - example in dependency parsing: Johansson and Nugues *Investigating Multilingual Dependency Parsing*, CoNLL 2006.
- often improves the the accuracy, but obviously makes the system slower

overview

multiclass linear classifiers

structured problems: overview

step-by-step structured prediction

task-specific learning

information about assignment 3

task-specific learning

- the second high-level approach is that we integrate the task (e.g. parsing, tagging) into the learning process
- we adapt our learning methods:
  - perceptron $\rightarrow$ structured perceptron
  - SVM $\rightarrow$ structured SVM
  - LR $\rightarrow$ conditional random field

recap: multiclass perceptron

$\boldsymbol{w}$ = zero vector
repeat N times
  for $(\boldsymbol{x}_i, y_i)$ in the training set $\mathcal{T}$
    $g = \text{guess}(\boldsymbol{w}, \boldsymbol{x}_i)$
    if g is not equal to y_i
      change $\boldsymbol{w}$ to increase the score of y_i
      change $\boldsymbol{w}$ to decrease the score of g
return $\boldsymbol{w}$

reranking

- assume that we have a simple system that is easy to train and fast to run, but uses a linguistic model that is too simple
  - for instance, a PCFG parser or an IBM model in translation
  - may have made linguistically problematic assumptions in order to make the system computationally efficient
- how can we build a smarter system on top of the simple one?

reranking

- assume that we have a simple system that is easy to train and fast to run, but uses a linguistic model that is too simple
  - for instance, a PCFG parser or an IBM model in translation
  - may have made linguistically problematic assumptions in order to make the system computationally efficient
- how can we build a smarter system on top of the simple one?
- **reranking**:
  - let the simplistic system generate k hypotheses
  - then use another system to select one of them
  - the reranker doesn't have to care about efficiency, so it can use any information

example: translation reranking

Kas sul kõht on tühi?	Is the stomach empty on you? Do you have an empty stomach? Are you starved? ...

a linear model for a reranker

- let's assume that we have a list of hypotheses H
  - for instance, the top 50 parse trees generated by a PCFG
- for each hypothesis h, we can extract some features using the function $\boldsymbol{f}(h)$
- we'll implement the reranker by using a scoring function with a weight vector $\boldsymbol{w}$

$$\text{score}(\boldsymbol{w}, h) = \boldsymbol{w} \cdot \boldsymbol{f}(h)$$

- then, for a hypothesis list H, we select the h that maximizes the scoring function

$$\text{SELECT}(\boldsymbol{w}, H) = \arg\max_{h \in H} \boldsymbol{w} \cdot \boldsymbol{f}(h)$$

UNIVERSITY OF GOTHENBURG

adapting the perceptron for reranking

- when training a reranker, our training set $\mathcal{T}$ consists of hypothesis lists for each input
  - and for each H_i, we have selected the best output y_i
- how can we find the weight vector $\boldsymbol{w}$?

adapting the perceptron for reranking

- when training a reranker, our training set $\mathcal{T}$ consists of hypothesis lists for each input
  - and for each H_i, we have selected the best output y_i
- how can we find the weight vector $\boldsymbol{w}$?
- let's do something similar to the multiclass perceptron:

$\boldsymbol{w}$ = zero vector
repeat N times
  for (H_i, y_i) in $\mathcal{T}$
    $g = \textbf{SELECT}(\boldsymbol{w}, H_i)$
    $\boldsymbol{w} = \boldsymbol{w} + \boldsymbol{f}(y_i) - \boldsymbol{f}(g)$
return w

examples

- parsing example: see Charniak and Johnson: *Coarse-to-fine n-best parsing and MaxEnt discriminative reranking*, ACL 2005.
- MT example: see `http://www.statmt.org/survey/Topic/Reranking`

going further: task-specific learning for dependency parsing

- the multiclass perceptron can be adapted to general prediction tasks – not just reranking
- let's see how to do dependency parsing in this framework

going further: task-specific learning for dependency parsing

- the multiclass perceptron can be adapted to general prediction tasks – not just reranking
- let's see how to do dependency parsing in this framework
- let's assume we can extract a feature vector $\boldsymbol{f}(x, y)$ for a sentence x and a parse tree y
  - so we can use a weight vector $\boldsymbol{w}$ to score parse trees

$$\text{score}(\boldsymbol{w}, x, y) = \boldsymbol{w} \cdot \boldsymbol{f}(x, y)$$

- we also have some procedure $\text{PARSE}(\boldsymbol{w}, x)$ that finds the top-scoring parse tree y

$$\text{PARSE}(\boldsymbol{w}, x) = \arg\max_{y} \boldsymbol{w} \cdot \boldsymbol{f}(x, y)$$

pseudocode: "parsing perceptron"

- we'd like to train the dependency parser: that is, find a weight vector $\boldsymbol{w}$ that scores good parses higher than bad parses
- in this case, our training set $\mathcal{T}$ consists of sentences x coupled with their corresponding gold-standard parse tree y
- how can we find the weight vector $\boldsymbol{w}$?

pseudocode: "parsing perceptron"

- we'd like to train the dependency parser: that is, find a weight vector $\boldsymbol{w}$ that scores good parses higher than bad parses
- in this case, our training set $\mathcal{T}$ consists of sentences x coupled with their corresponding gold-standard parse tree y
- how can we find the weight vector $\boldsymbol{w}$?
- almost the same idea as in the reranking perceptron:

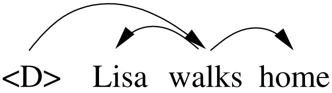
$\boldsymbol{w}$ = zero vector
repeat N times
  for (x_i, y_i) in $\mathcal{T}$
    $g = \textbf{PARSE}(\boldsymbol{w}, x_i)$
    $\boldsymbol{w} = \boldsymbol{w} + \boldsymbol{f}(x_i, y_i) - \boldsymbol{f}(x_i, g)$
return w

graph-based parsing

- so what about the missing pieces?
  - what is the feature function $\boldsymbol{f}(x, y)$?
  - how do we find the highest-scoring tree? $\text{PARSE}(\boldsymbol{w}, x_i)$
- there are many ways they could be implemented; we'll briefly go through the famous **maximum spanning tree** parser (MSTParser)
  - see McDonald, Crammer and Pereira, *Online Large-Margin Training of Dependency Parsers*, ACL 2005.
- MSTParser is a special case of a **graph-based parser** – a parser where we search for the best tree in a graph

UNIVERSITY OF GOTHENBURG

edge-factored feature representation



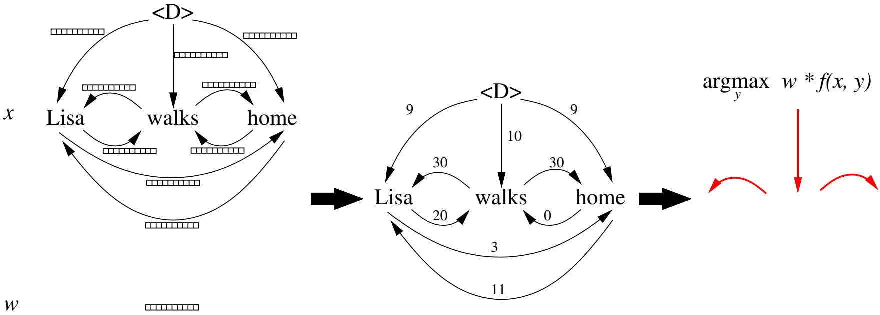
- MSTParser uses an **edge-factored** model, where features are extracted from each edge in a parse tree:

$$
\begin{aligned}
\boldsymbol{f}(x,y) \quad = \quad & \boldsymbol{f}_{edge}(x, \text{<D>}{\rightarrow}\text{walks})+ \\
& +\boldsymbol{f}_{edge}(x, \text{walks}{\rightarrow}\text{Lisa})+ \\
& +\boldsymbol{f}_{edge}(x, \text{walks}{\rightarrow}\text{home})
\end{aligned}
$$

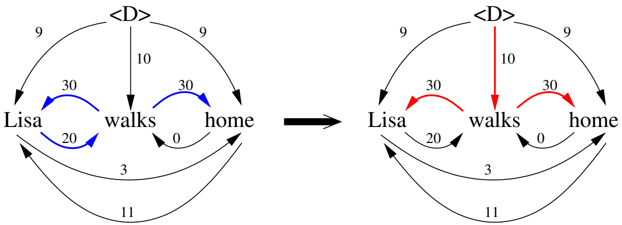
- the edges are scored independently of each other

UNIVERSITY OF GOTHENBURG

MSTParser: scoring the edges and finding the best tree



finding the maximum spanning tree



- the **Chu–Liu/Edmonds** algorithm:
  1. for each node, find the top-scoring incoming edge
  2. if there are no cycles, we are done
  3. if there is a cycle, create a single node containing the cycle
  4. find the MST in the new graph (recursion)
  5. break the cycle...
- also: **Eisner** algorithm, similar to CKY (see McDonald paper)
  - CL/E finds the highest-scoring tree
  - Eisner finds the highest-scoring **projective** tree

discussion

- what do we need to implement a PoS tagger in this way?

discussion

- what do we need to implement a PoS tagger in this way?
- see the classic paper by Collins: *Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms*, EMNLP 2002

structure-prediction variants of SVM and LR

- what we called the “parsing perceptron” is typically called the **structured perceptron**
  - as we saw, the parts specific to parsing can be replaced
  - the feature function: $\boldsymbol{f}(x, y)$?
  - finding the highest-scoring tree: $\text{PARSE}(\boldsymbol{w}, x_i)$
- SVM can be adapted in a very similar way
  - multiclass Pegasos can be applied without change
- the counterpart of LR is called **conditional random field**
  - it is probably the most popular model for sequence tagging
  - not so popular for other problems, since it's a bit more complicated to implement

software libraries

- there isn't anything comparable to scikit-learn for structured prediction
- PyStruct: `https://pystruct.github.io`
  - contains a number of learning algorithms as well as optimization tools to help implementing the arg max
  - designed to be compatible with scikit-learn
  - unfortunately, can't yet handle sparse feature vectors...
- several specialized libraries, mostly for sequence tagging with CRF:
  - Mallet – a Java library that can be called from NLTK
  - CRF++
  - CRF-SGD – very efficient, will be used in the assignment
- seqlearn: `http://larsmans.github.io/seqlearn`
  - implemented by one of the designers of scikit-learn
  - only sequence tagging

step-by-step or task-specific: pros and cons

- step-by-step systems
  - are easier to build
  - tend to be faster
  - are less restrictive about features
  - can build on existing ML packages
- task-specific models are more accurate for some problems
- for the dependency parsing, there have been some studies comparing transition-based and graph-based methods from a linguistic perspective

example: parser comparison

Sammanställning parsrar

parser	korrekthet	länkkorrekthet ▲	tid/mening	kommentar
LTH	82.43	88.58	0.193	2-ordning, pseudoprojektiv, Brown-kluster
Mate-tools	81.65	87.93	0.141	ickeprojektiv, Brown-kluster
TurboParser	79.91	87.31	0.053	
ZPar	80.78	87.26	0.190	projektiv
MSTParser	78.14	86.32	0.119	2-ordning, ickeprojektiv
MaltParser	78.42	85.17	0.005	ickeprojektiv, Brown-kluster, tränad enligt instruktioner av Johan Hall
Huang	-	84.74	0.017	projektiv, inga funktioner

overview

multiclass linear classifiers

structured problems: overview

step-by-step structured prediction

task-specific learning

information about assignment 3

assignment 3

United Nations official Ekeus heads for Baghdad.
[ORG] [PER] [LOC]

- implement a named entity tagger
  - step-by-step sequence tagging approach
  - naive training
  - compare to an off-the-shelf CRF
  - file processing and evaluation code will be provided
- for a VG, implement a beam search