Background

Sequence labeling

- MEMMs - ?
- HMMs – you know, right?
- Structured perceptron – also this?
- linear-chain CRFs - ?

Sequence labeling

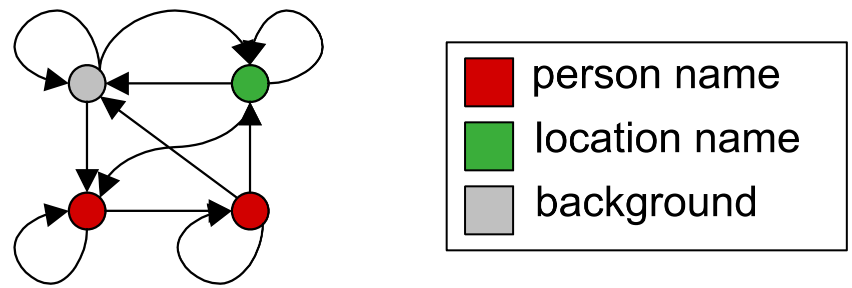
- Imagine labeling a sequence of symbols in order to ….
  - do NER (finding named entities in text)
  - labels are: entity types
  - symbols are: words

IE with Hidden Markov Models

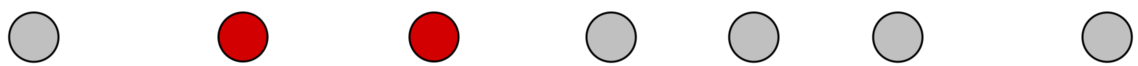
Given a sequence of observations:

Yesterday Pedro Domingos spoke this example sentence.

and a trained HMM:



Find the most likely state sequence: (Viterbi) $\arg\max_{s} P(s, o)$



Yesterday **Pedro Domingos** spoke this example sentence.

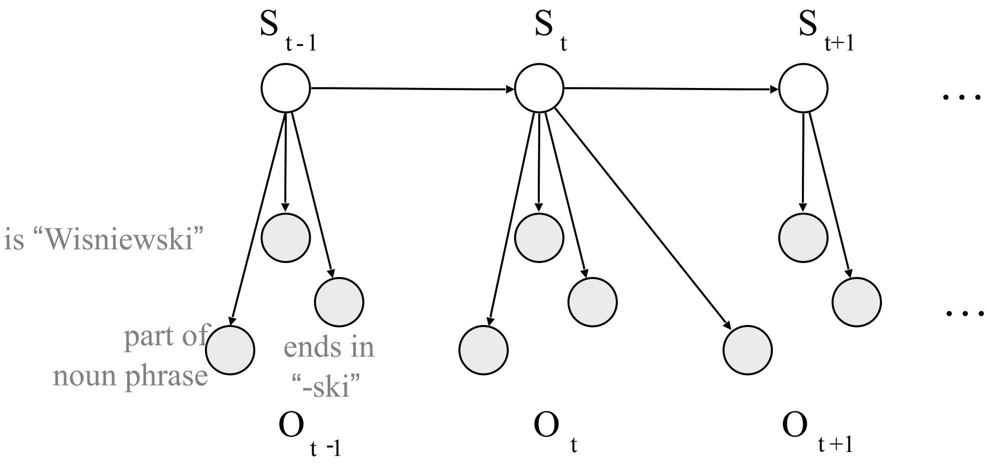
Any words said to be generated by the designated "person name" state extract as a person name:

Person name: Pedro Domingos

What is a symbol?

Ideally we would like to use many, arbitrary, overlapping features of words.

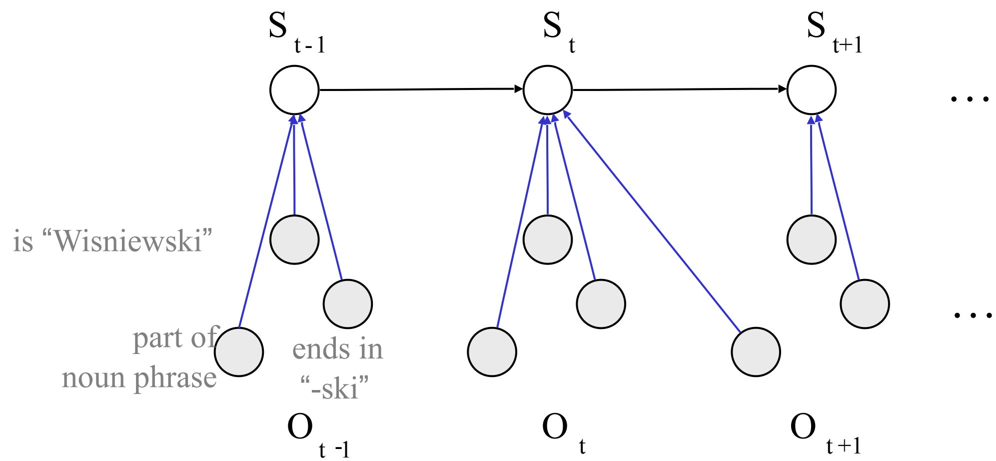
identity of word
ends in "-ski"
is capitalized
is part of a noun phrase
is in a list of city names
is under node X in WordNet
is in bold font
…



Lots of learning systems are not confounded by multiple, non-independent features: decision trees, neural nets, SVMs, …

What is a symbol?

identity of word
ends in “-ski”
is capitalized
is part of a noun phrase
is in a list of city names
is under node X in WordNet
is in bold font
…

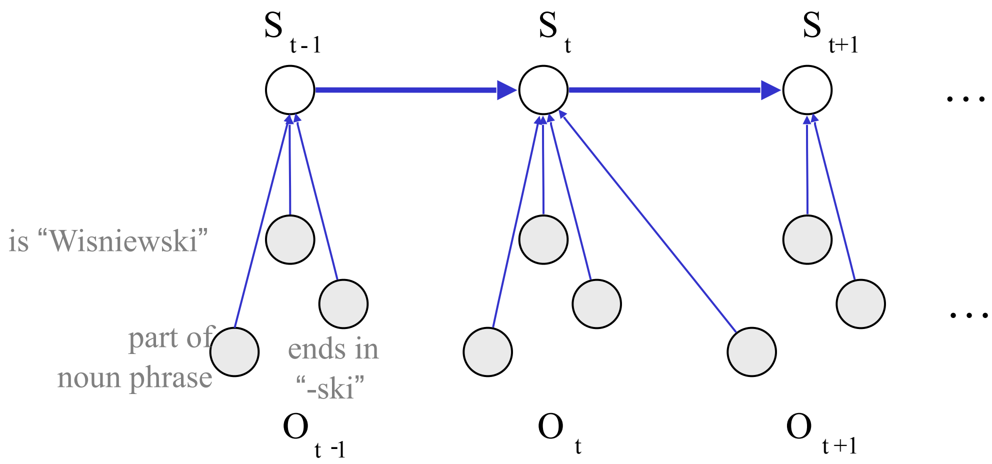


Idea: replace generative model in HMM with a maxent model, where state depends on observations

$$\Pr(s_t \mid x_t) = \ldots$$

What is a symbol?

identity of word
ends in "-ski"
is capitalized
is part of a noun phrase
is in a list of city names
is under node X in WordNet
is in bold font
…

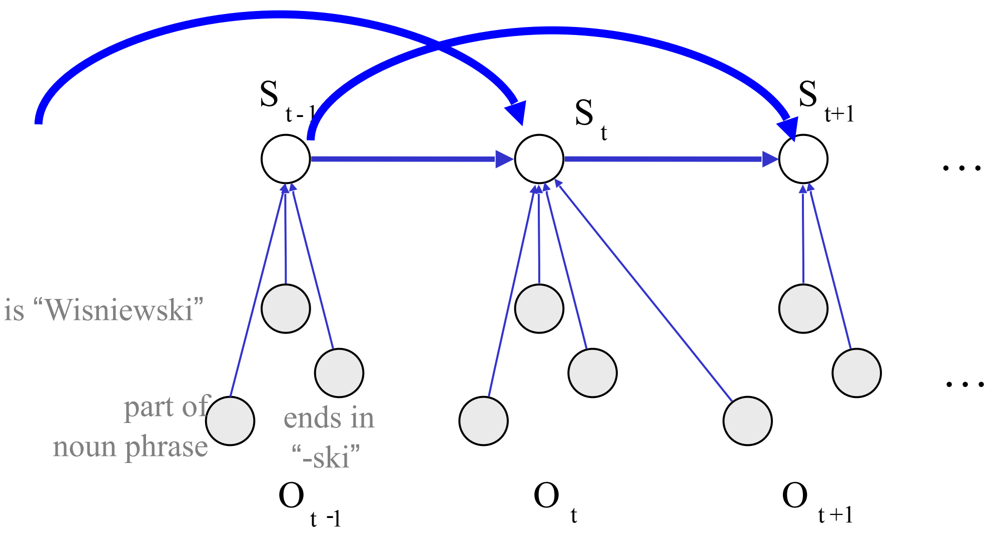


Idea: replace generative model in HMM with a maxent model, where state depends on observations and previous state

$$\Pr(s_t \mid x_t, s_{t-1,}) = \ldots$$

What is a symbol?

identity of word
ends in "-ski"
is capitalized
is part of a noun phrase
is in a list of city names
is under node X in WordNet
is in bold font
is indented
is in hyperlink anchor
…



Idea: replace generative model in HMM with a maxent model, where state depends on observations and previous state history

$$\Pr(s_t \mid x_t, s_{t-1}, s_{t-2}, \ldots) = \ldots$$

Ratnaparkhi's MXPOST

POS tagger from late 90's

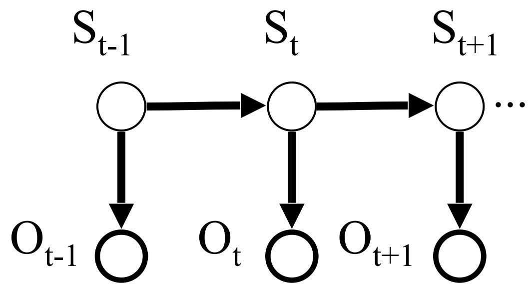
- Sequential learning problem: predict POS tags of words.
- Uses MaxEnt model described above.
- Rich feature set.
- To smooth, discard features occurring < 10 times.

Condition	Features	
w_i is not rare	$w_i = X$	& $t_i = T$
w_i is rare	X is prefix of w_i, $\|X\| \leq 4$	& $t_i = T$
	X is suffix of w_i, $\|X\| \leq 4$	& $t_i = T$
	w_i contains number	& $t_i = T$
	w_i contains uppercase character	& $t_i = T$
	w_i contains hyphen	& $t_i = T$
$\forall\ w_i$	$t_{i-1} = X$	& $t_i = T$
	$t_{i-2}t_{i-1} = XY$	& $t_i = T$
	$w_{i-1} = X$	& $t_i = T$
	$w_{i-2} = X$	& $t_i = T$
	$w_{i+1} = X$	& $t_i = T$
	$w_{i+2} = X$	& $t_i = T$

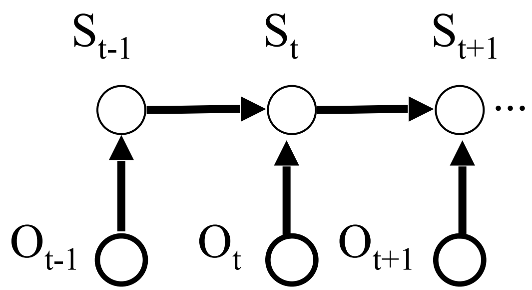
Table 1: Features on the current history h_i

Conditional Markov Models (CMMs) aka MEMMs aka Maxent Taggers *vs* HMMS

$$\Pr(s,o) = \prod_i \Pr(s_i \mid s_{i-1}) \Pr(o_i \mid s_{i-1})$$



$$\Pr(s \mid o) = \prod_i \Pr(s_i \mid s_{i-1}, o_{i-1})$$



Graphical comparison among HMMs, MEMMs and CRFs

HMM MEMM CRF

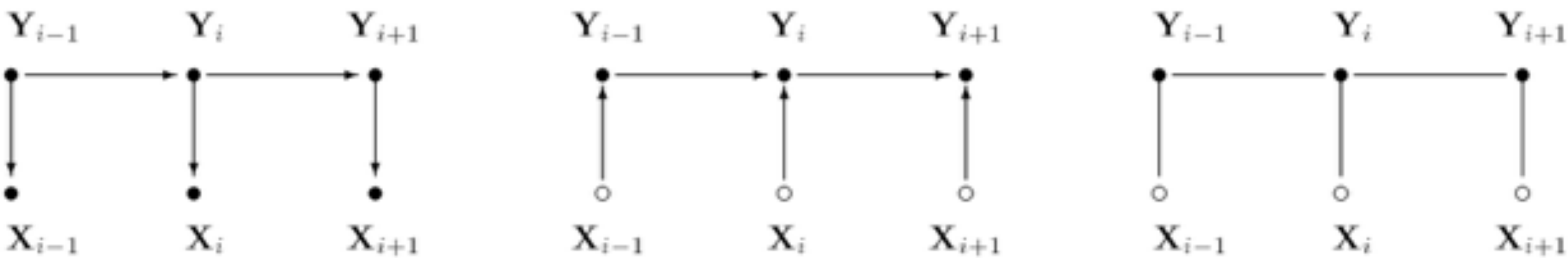


Figure 2. Graphical structures of simple HMMs (left), MEMMs (center), and the chain-structured case of CRFs (right) for sequences. An open circle indicates that the variable is not generated by the model.

Stacking and Searn

William W. Cohen

Stacked Sequential Learning

William W. Cohen

Center for Automated Learning and Discovery
Carnegie Mellon University

Vitor Carvalho

Language Technology Institute
Carnegie Mellon University

Outline

- Motivation:
  - MEMMs don’t work on segmentation tasks
- New method:
  - Stacked sequential MaxEnt
  - Stacked sequential YFL
- Results
- More results...
- Conclusions

However, in celebration of the locale, I will present this results in the style of Sir Walter Scott (1771-1832), author of “Ivanhoe” and other classics.

In that pleasant district of merry Pennsylvania which is watered by the river Mon, there extended since ancient times a large computer science department. Such being our chief scene, the date of our story refers to a period towards the middle of the year 2003 ....

Chapter 1, in which a graduate student (Vitor) discovers a bug in his advisor's code that he cannot fix

The *problem*: identifying *reply* and *signature* sections of email messages.
The *method*: classify each line as *reply, signature,* or *other.*

```
<other>  From: wcohen@cs.cmu.edu
<other>  To: Vitor Carvalho <vitor@cs.cmu.edu>
<other>  Subject: Re: Did you try to compile javadoc recently?
<other>  Date: 25 Mar 2004 12:05:51 -0500
<other>
<other>  Try cvs update -dP, this removes files & directories that have been
deleted from cvs.
<other>  - W
<other>
<reply>  On Wed, 2004-03-24 at 19:58, Vitor Carvalho wrote:
<reply>  > I've just checked-out the baseline m3 code and
<reply>  > "Ant dist" is working fine, but "ant javadoc" is not.
<reply>  > Thanks
<reply>  > Vitor
<other>
<sig>    -------------------------------------------------------------------
<sig>    William W. Cohen                           "Would you drive a mime
<sig>    wcohen@cs.cmu.edu                             nuts if you played a
<sig>    http://www.wcohen.com                           blank audio tape
<sig>    Associate Research Professor                      full blast?" --
<sig>    CALD, Carnegie-Mellon University                        S. Wright
```

Chapter 1, in which a graduate student discovers a bug in his advisor's code that he cannot fix

The *problem*: identifying *reply* and *signature* sections of email messages.
The *method*: classify each line as *reply, signature,* or *other.*

The *warmup:* classify each line is *signature* or *nonsignature,* using learning methods from Minorthird, and dataset of 600+ messages

The *results:* from [CEAS-2004, Carvalho & Cohen]....

Chapter 1, in which a graduate student discovers a bug in his advisor's code that he cannot fix

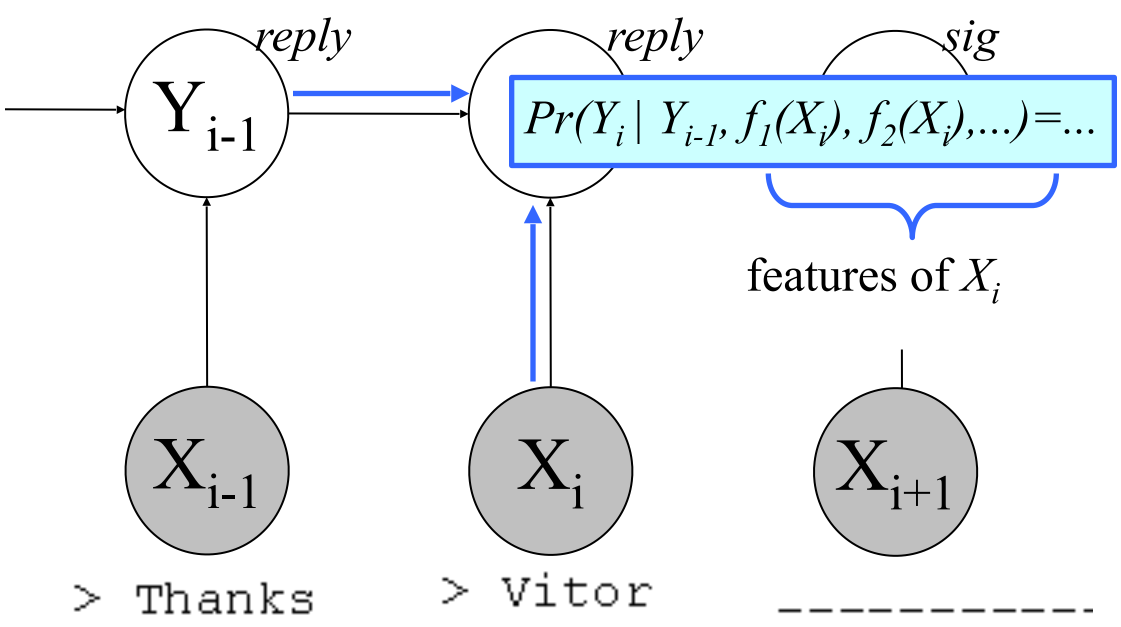
But... Minorthird's version of MEMMs has an accuracy of less than 70% (guessing majority class gives accuracy around 10%!)

Learning Algorithm		Without Features from Previous and Next Lines			
		Accuracy (%)	F1	Precision	Recall
Non-Sequential					
Naïve Bayes		94.13	73.88	66.80	82.65
Maximum Entropy		96.26	80.16	86.07	75.00
SVM		96.41	80.39	89.41	73.02
VotedPerceptron		96.10	80.23	81.88	78.65
AdaBoost		**96.53**	**82.12**	85.44	79.04
Sequential					
CPerceptron(5, 25)		97.01	83.62	93.02	75.94
CMM(SVM, 5)		91.28	66.82	54.11	87.35
CRF		**98.13**	**90.97**	88.05	94.09

Flashback: In which we recall the invention and re-invention of sequential classification with recurrent sliding windows, ..., MaxEnt Markov Models (MEMM)

- From data, **learn** $Pr(y_i|y_{i-1},x_i)$
  - MaxEnt model
- To classify a sequence $x_1,x_2,...$ **search** for the best $y_1,y_2,...$
  - *Viterbi*
  - *beam search*

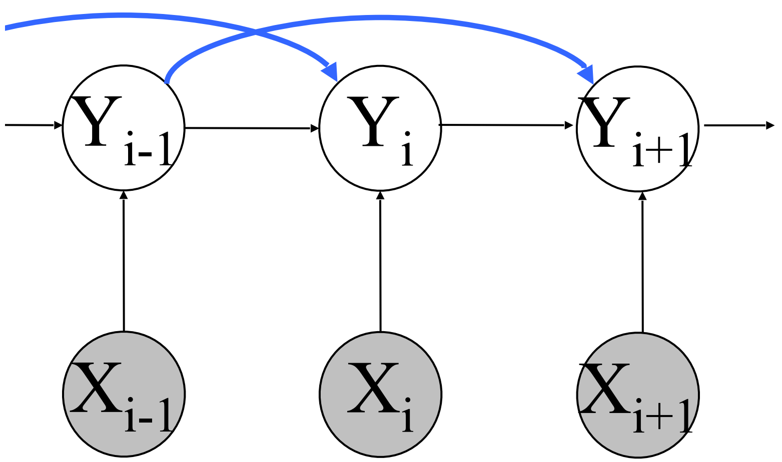
probabilistic classifier using previous label Y_{i-1} as a feature (or conditioned on *Yi-1)*



Flashback: In which we recall the invention and re-invention of sequential classification with recurrent sliding windows, ..., MaxEnt Markov Models (MEMM) ... and also praise their *many virtues* relative to CRFs

- MEMMs are easy to implement
- MEMMs train quickly
  - no probabilistic inference in the inner loop of learning
- You can use any old classifier (even if it's not probabilistic)
- MEMMs scale well with number of classes and length of history

$Pr(Y_i \mid Y_{i-1}, Y_{i-2}, \ldots, f_1(X_i), f_2(X_i), \ldots) = \ldots$



The flashback ends and we return again to our document analysis task , on which the elegant MEMM method fails miserably *for reasons unknown*

MEMMs have an accuracy of less than 70% on this problem – but *why* ?

Learning Algorithm		Without Features from Previous and Next Lines			
		Accuracy (%)	F1	Precision	Recall
Non-Sequential					
Naïve Bayes		94.13	73.88	66.80	82.65
Maximum Entropy		96.26	80.16	86.07	75.00
SVM		96.41	80.39	89.41	73.02
VotedPerceptron		96.10	80.23	81.88	78.65
AdaBoost		**96.53**	**82.12**	85.44	79.04
Sequential					
CPerceptron(5, 25)		97.01	83.62	93.02	75.94
CMM(SVM, 5)		91.28	66.82	54.11	87.35
CRF		**98.13**	**90.97**	88.05	94.09

Chapter 2, in which, in the fullness of time, the mystery is investigated...

...and it transpires that often the classifier predicts a signature block that is much **longer** than is correct

false positive predictions

...as if the MEMM "gets stuck" predicting the sig label.

predicted	*true*	
<other>	<other>	From: wcohen@cs.
<other>	<other>	To: Vitor Carval
<other>	<other>	Subject: Re: Did
<other>	<other>	Date: 25 Mar 200
<other>	<other>	
<other>	<other>	Try cvs update -
deleted	deleted	from cvs.
<sig>	<other>	- W
<sig>	<other>	
<sig>	<reply>	On Wed, 2004-03-
<sig>	<reply>	> I've just chec
<sig>	<reply>	> "Ant dist" is
<sig>	<reply>	> Thanks
<sig>	<reply>	> Vitor
<sig>	<other>	
<sig>	<sig>	----------------
<sig>	<sig>	William W. Cohen
<sig>	<sig>	wcohen@cs.cmu.ed
<sig>	<sig>	http://www.wcohe
<sig>	<sig>	Associate Resear
<sig>	<sig>	CALD, Carnegie-M

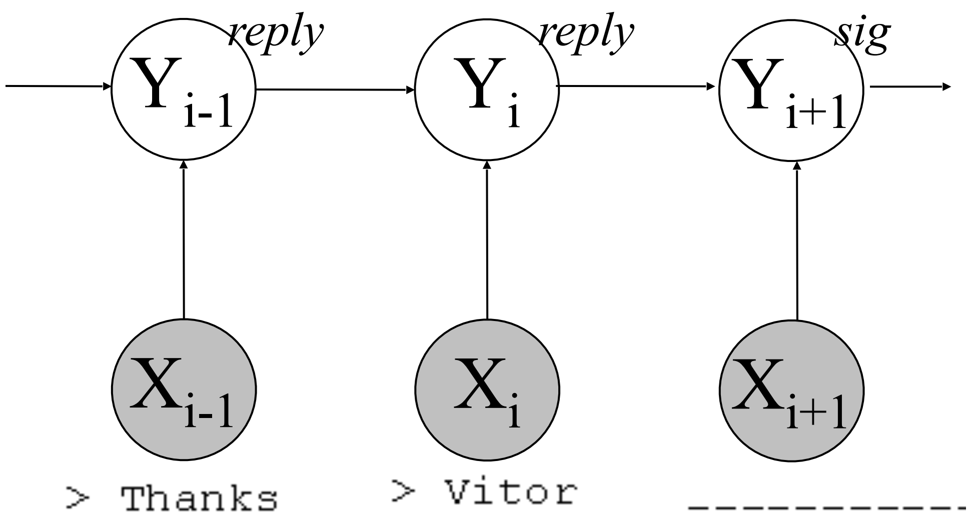
Chapter 2, in which, in the fullness of time, the mystery is investigated...

...and it transpires that

$$\Pr(Y_i = sig \mid Y_{i-1} = sig) = 1 - \varepsilon$$

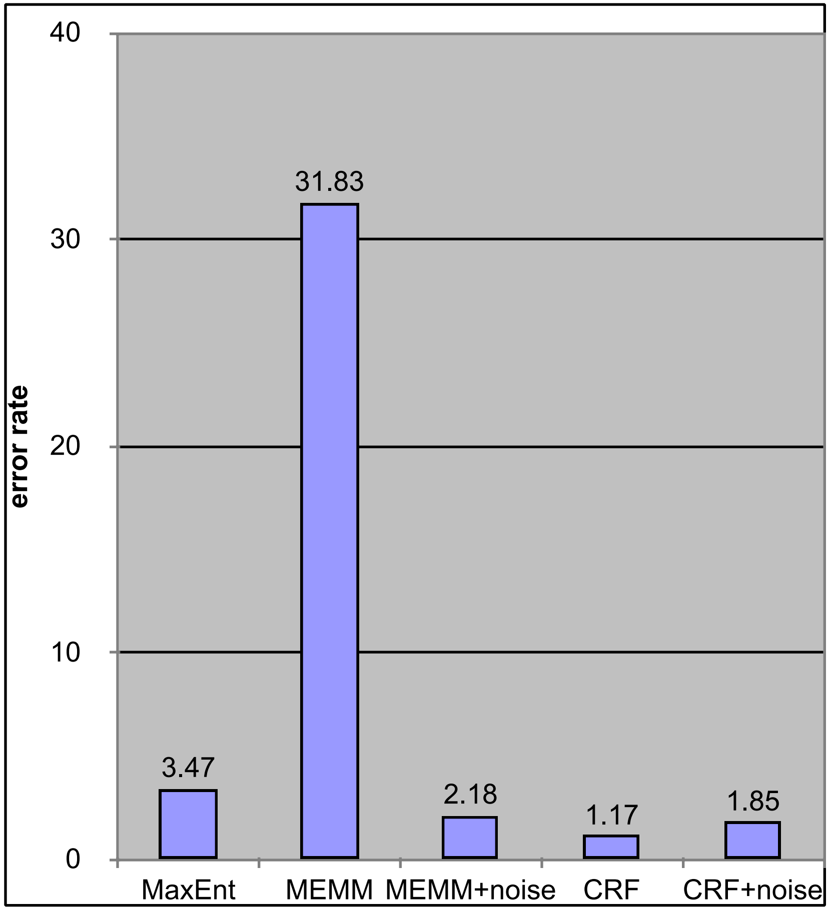
as estimated from the data, giving the previous label a very high weight.

```
<other>   From: wcohen@cs.
<other>   To: Vitor Carval
<other>   Subject: Re: Did
<other>   Date: 25 Mar 200
<other>
<other>   Try cvs update -
deleted from cvs.
<other>   - W
<other>
<reply>   On Wed, 2004-03-
<reply>   > I've just chec
<reply>   > "Ant dist" is
<reply>   > Thanks
<reply>   > Vitor
<other>
<sig>     ----------------
<sig>     William W. Cohen
<sig>     wcohen@cs.cmu.ed
<sig>     http://www.wcohe
<sig>     Associate Resear
<sig>     CALD, Carnegie-M
```



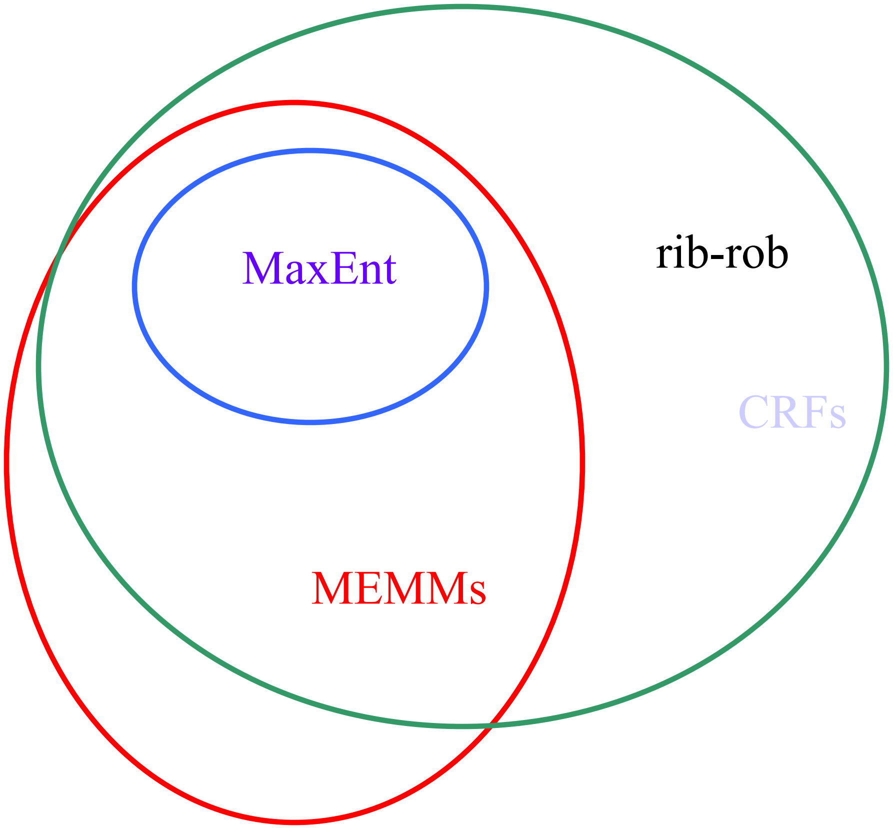
Chapter 2, in which, in the fullness of time, the mystery is investigated...

- We added "sequence noise" by randomly switching around 10% of the lines: this
  - lowers the weight for the previous-label feature
  - *improves* performance for MEMMs
  - *degrades* performance for CRFs
- **Adding noise in this case however is a loathsome bit of hackery.**



Chapter 2, in which, in the fullness of time, the mystery is investigated...

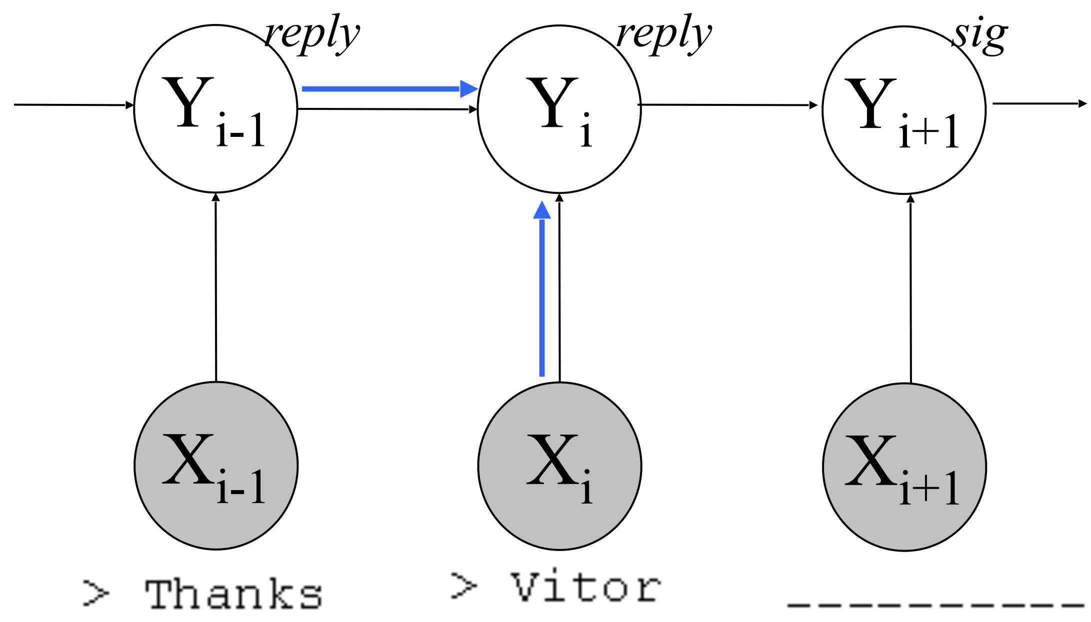
- ~~*Label bias problem* CRFs can represent some distributions that MEMMs cannot [Lafferty et al 2000]:~~
  - ~~e.g., the "rib-rob" problem~~
  - ~~this doesn't explain why MaxEnt >> MEMMs~~
- ~~*Observation bias problem:* MEMMs can overweight "observation" features [Klein and Manning 2002] :~~
  - ~~here we observe the *opposite:* the history features are overweighted~~



Chapter 2, in which, in the fullness of time, the mystery is investigated...and an explanation is proposed.

- From data, **learn** $Pr(y_i|y_{i-1},x_i)$
  - MaxEnt model
- To classify a sequence $x_1,x_2,\ldots$ **search** for the best $y_1,y_2,\ldots$
  - Viterbi
  - beam search

probabilistic classifier using previous label Y_{i-1} as a feature (or conditioned on $Yi-1$)



Chapter 2, in which, in the fullness of time, the mystery is investigated...and an explanation is proposed.

- From data, **learn** $Pr(y_i|y_{i-1},x_i)$
  - MaxEnt model
- To classify a sequence $x_1,x_2,\ldots$ **search** for the best $y_1,y_2,\ldots$
  - Viterbi
  - beam search

Learning data is noise-free, including values for Y_{i-1}

Classification data values for Y_{i-1} are noisy since they come from predictions

i.e., the history values used at learning time are a poor approximation of the values seen in classification

Chapter 3, in which a novel extension to MEMMs is proposed that will correct the performance problem

- From data, **learn** $Pr(y_i|y_{i-1},x_i)$
  - MaxEnt model
- To classify a sequence $x_1,x_2,\ldots$

find approximate Y's with a MaxEnt-learned hypothesis, and then apply the sequential model to that

  - ~~beam search~~

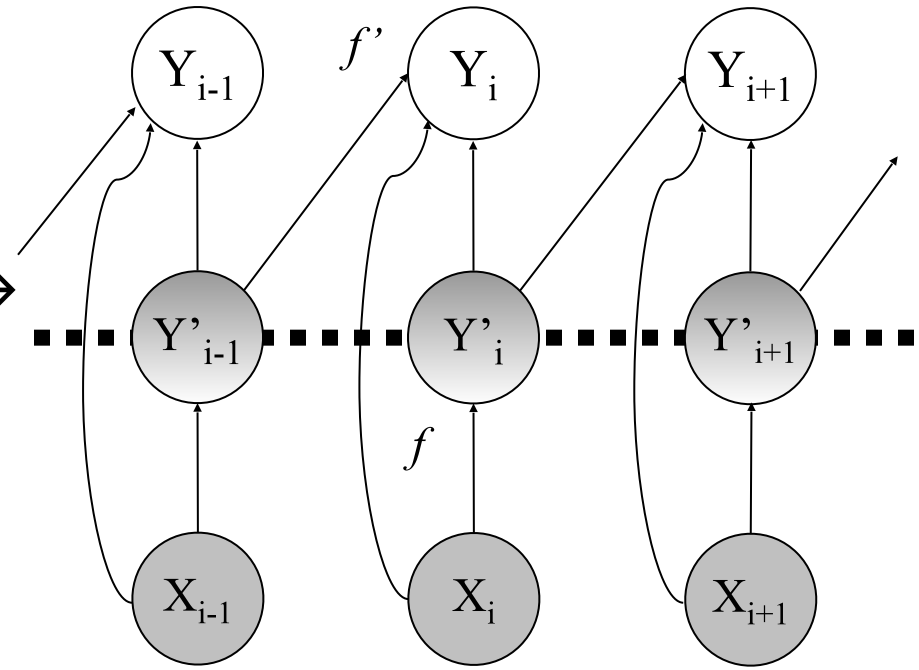
While learning, replace the true value for Y_{i-1} with an approximation of the predicted value of Y_{i-1}

To approximate the value predicted by MEMMs, use the value predicted by non-sequential MaxEnt in a cross-validation experiment.

After Wolpert [1992] we call this *stacked MaxEnt.*

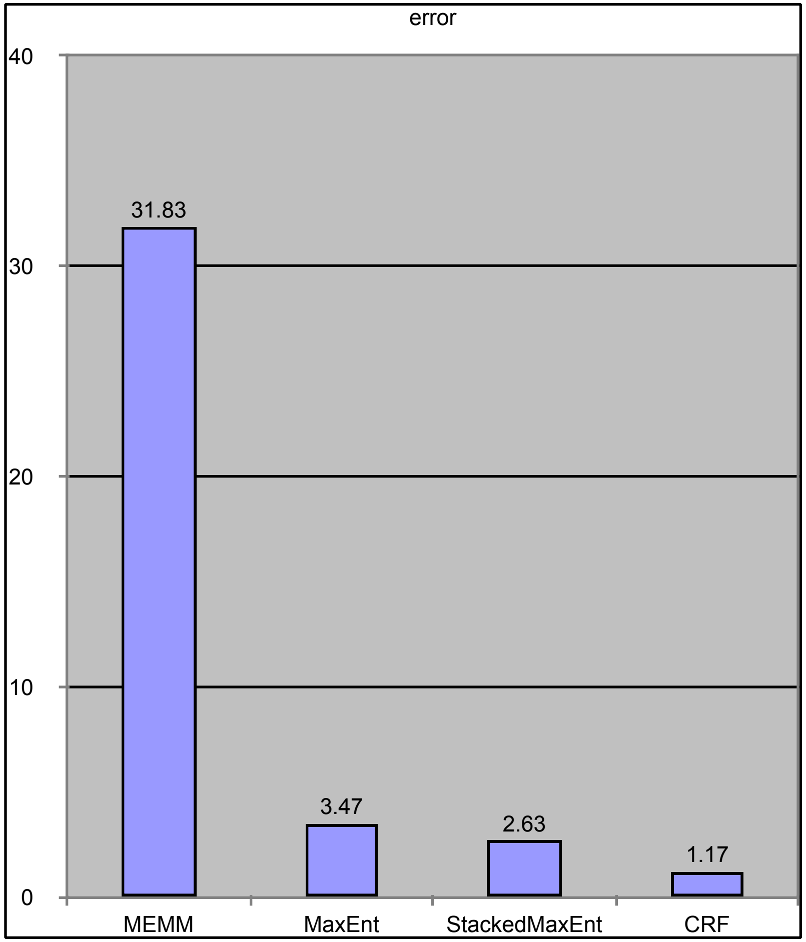
Chapter 3, in which a novel extension to MEMMs is proposed that will correct the performance problem

- Learn $Pr(y_i|x_i)$ with MaxEnt and save the model as $f(x)$
- Do k-fold cross-validation with MaxEnt, saving the cross-validated predictions the cross-validated predictions $y'_i=f_k(x_i)$
- Augment the original examples with the y''s and compute history features: $g(x,y')$ → x'
- Learn $Pr(y_i|x'_i)$ with MaxEnt and save the model as $f'(x')$
- To classify: augment x with $y'=f(x)$, and apply f to the resulting x': i.e., return $f'(g(x,f(x)))$



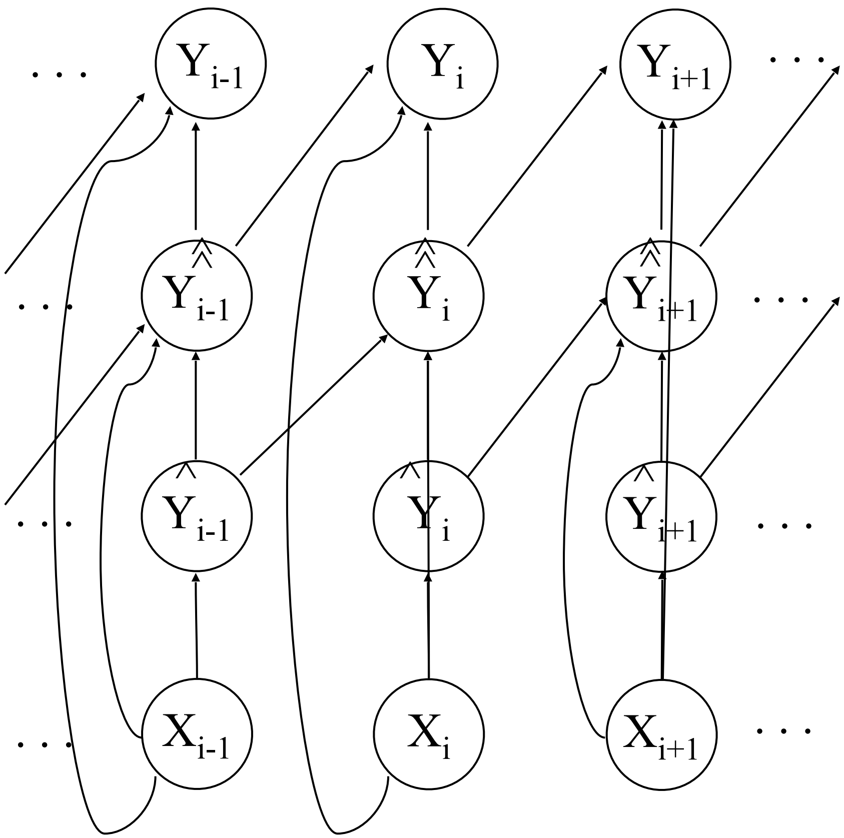
Chapter 3, in which a novel extension to MEMMs is proposed that will correct the performance problem

- StackedMaxEnt (k=5) outperforms MEMMs and non-sequential MaxEnt, but not CRFs
- StackedMaxEnt can also be easily extended....
  - It's easy (but expensive) to increase the depth of stacking
  - It's easy to increase the history size
  - It's easy to build features for "future" estimated Yi's as well as "past" Yi's.
  - stacking can be applied to any other sequential learner



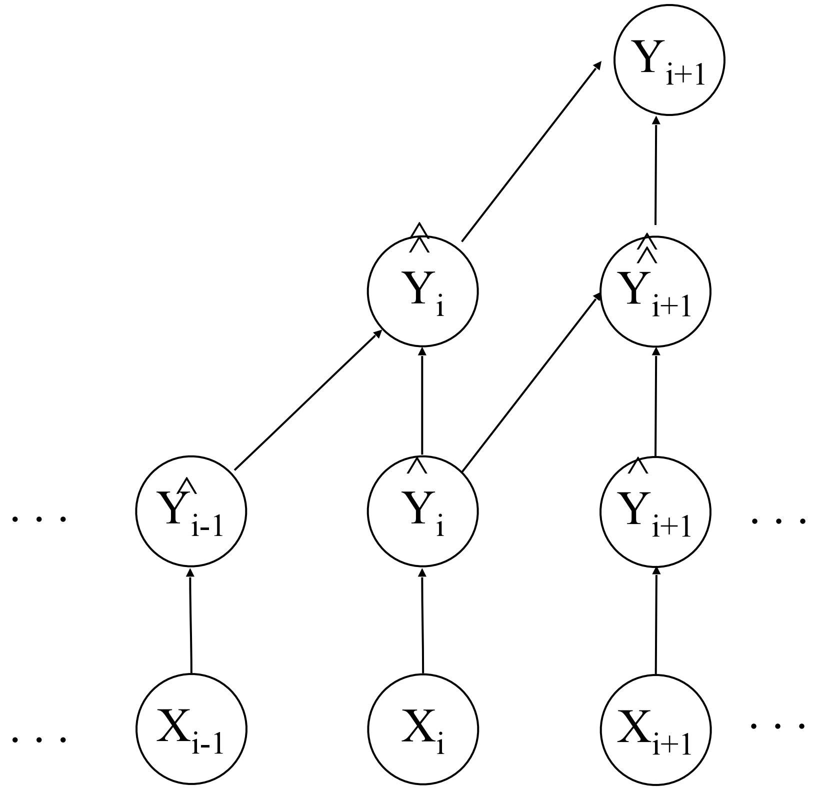
Chapter 3, in which a novel extension to MEMMs is proposed that will correct the performance problem

- StackedMaxEnt can also be easily extended....
  - **It's easy (but expensive) to increase the depth of stacking**
  - It's cheap to increase the history size
  - It's easy to build features for "future" estimated Yi's as well as "past" Yi's.
  - stacking can be applied to any other sequential learner



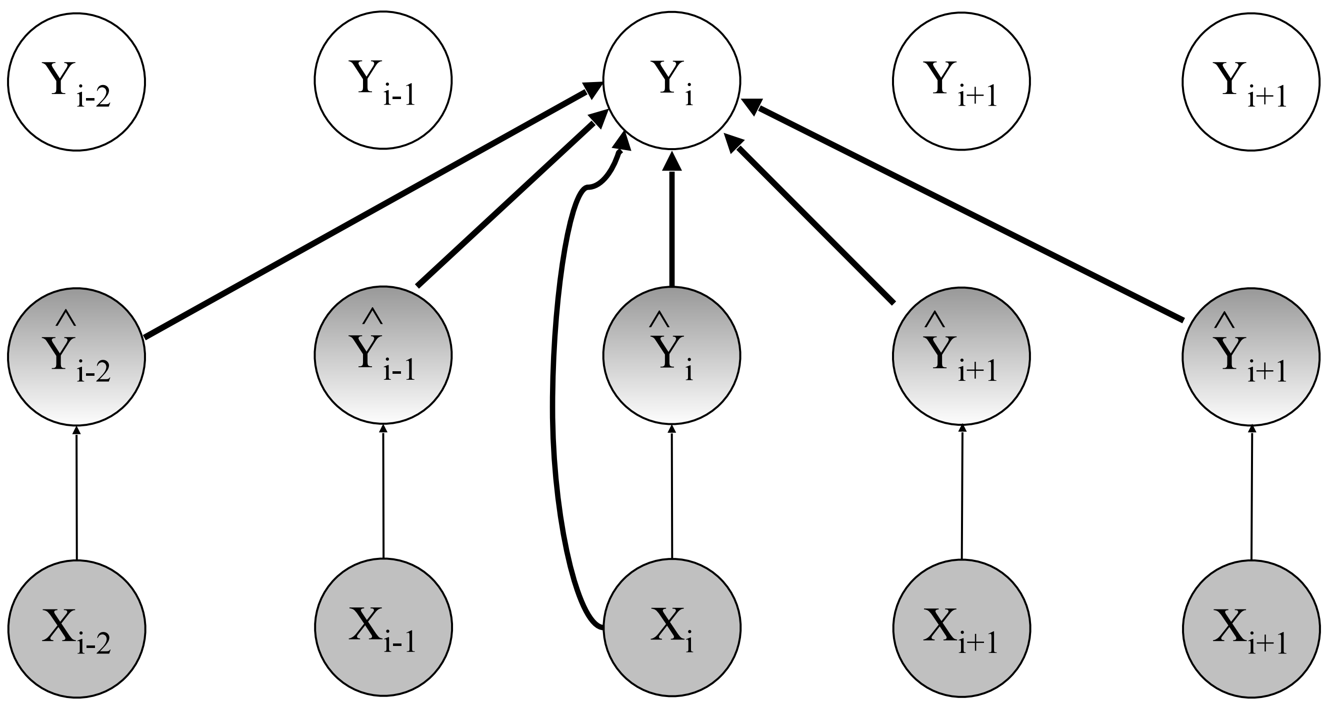
Chapter 3, in which a novel extension to MEMMs is proposed that will correct the performance problem

- StackedMaxEnt can also be easily extended....
  - **It's easy (but expensive) to increase the depth of stacking**
  - It's cheap to increase the history size
  - It's easy to build features for "future" estimated Yi's as well as "past" Yi's.
  - stacking can be applied to any other sequential learner



Chapter 3, in which a novel extension to MEMMs is proposed that will correct the performance problem

- StackedMaxEnt can also be easily extended....
  - **It’s cheap to increase the history size, and build features for “future” estimated Yi’s as well as “past” Yi’s.**

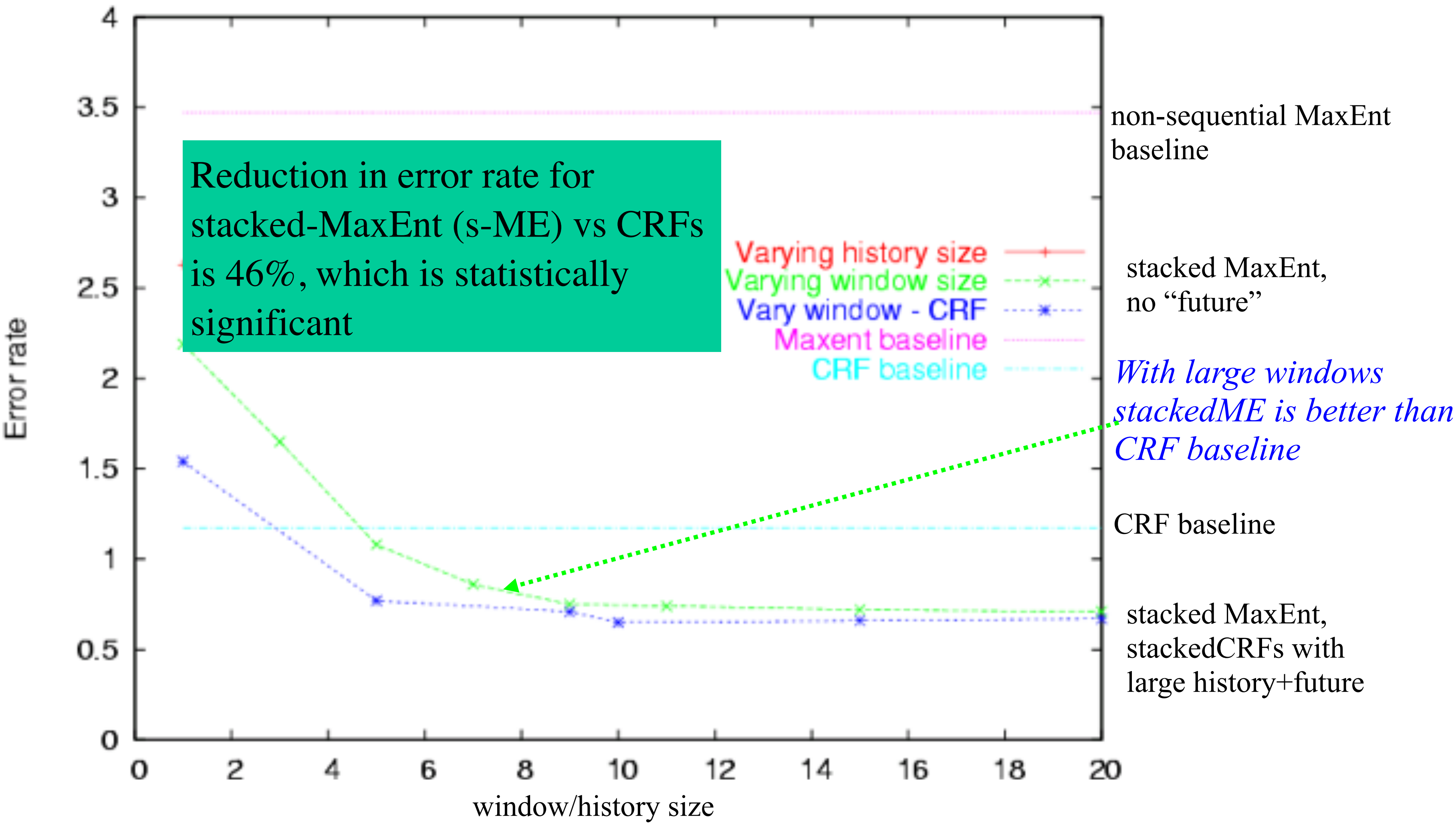


Chapter 3, in which a novel extension to MEMMs is proposed that will correct the performance problem

- StackedMaxEnt can also be easily extended....
  - It's easy (but expensive) to increase the depth of stacking
  - It's cheap to increase the history size
  - It's easy to build features for "future" estimated Yi's as well as "past" Yi's.
  - **stacking can be applied to any other sequential learner**

- Learn $Pr(y_i|x_i)$ with CRF ~~MaxEnt~~ and save the model as $f(x)$
- Do k-fold cross-validation with CRF ~~MaxEnt~~, saving the cross-validated predictions the cross-validated predictions $y'_i=f_k(x_i)$
- Augment the original examples with the y''s and compute history features: $g(x,y') \rightarrow x'$
- Learn $Pr(y_i|x'_i)$ with CRF ~~MaxEnt~~ and save the model as $f'(x')$
- To classify: augment x with $y'=f(x)$, and apply f to the resulting x': i.e., return $f'(g(x,f(x)))$

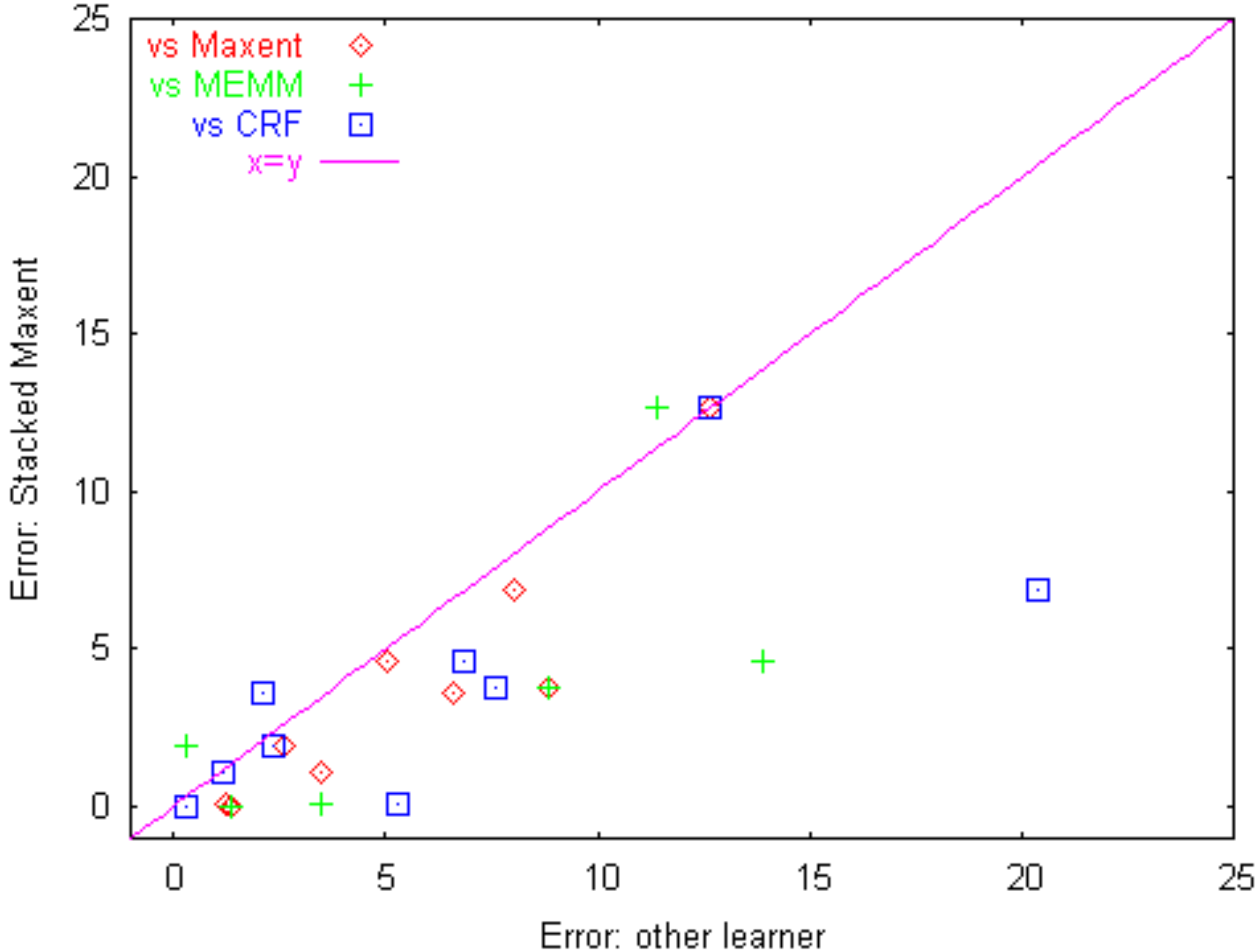
Chapter 3, in which a novel extension to MEMMs is proposed and several diverse variants of the extension are evaluated on signature-block finding....



Chapter 4, in which the experiment above is repeated on a new domain, and then repeated again on yet another new domain.

			-stacking		+stacking (w=k=5)	
	Task	MEMM	ME	CRF	s-ME	s-CRF
newsgroup FAQ segmentation (2 labels x three newsgroups)	A/aigen	53.61	8.02	20.35	6.91	**5.78**
	A/ainn	70.09	6.61	2.14	3.65	**1.67**
	A/aix	13.86	5.02	6.83	**4.59**	11.79
	T/aigen	0.30	2.60	2.39	1.92	**0.00**
	T/ainn	1.36	1.39	0.28	**0.00**	0.28
	T/aix	3.51	1.25	5.26	**0.05**	4.44
video segmentation	1/video	**11.39**	12.66	12.66	12.66	13.92
	2/video	8.86	8.86	7.59	**3.80**	7.59
	mailsig	31.83	3.47	1.17	1.08	**0.77**

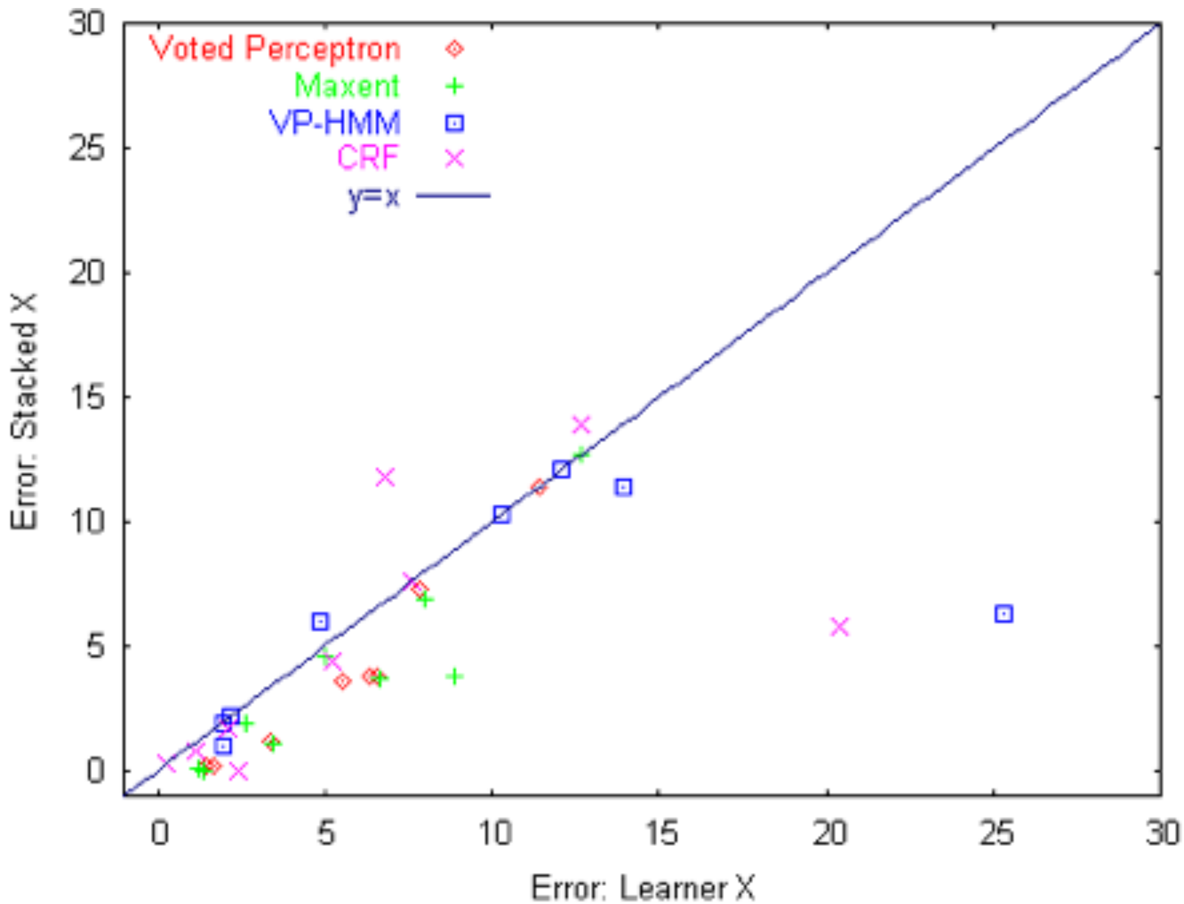
Chapter 4, in which the experiment above is repeated on a new domain, and then repeated again on yet another new domain.



Chapter 5, in which all the experiments above were repeated for a second set of learners: the *voted perceptron* (VP), the *voted-perceptron-trained HMM* (VP-HMM), and their stacked versions.

Task	VP	VPHMM	s-VP	s-VPHMM
A/aigen	7.87	12.09	**7.33**	12.09
A/ainn	6.59	10.26	**3.76**	10.26
A/aix	5.50	4.86	**3.61**	5.95
J/aigen	1.68	2.16	**0.18**	2.16
J/ainn	1.44	1.93	**0.19**	1.93
J/aix	3.40	1.95	1.16	**1.01**
1/video	**11.39**	13.92	**11.39**	**11.39**
2/video	6.33	25.32	**3.80**	6.33
mailsig	3.40	1.95	1.16	**1.01**

Chapter 5, in which all the experiments above were repeated for a second set of learners: the *voted perceptron* (VP), the *voted-perceptron-trained HMM* (VP-HMM), and their stacked versions.



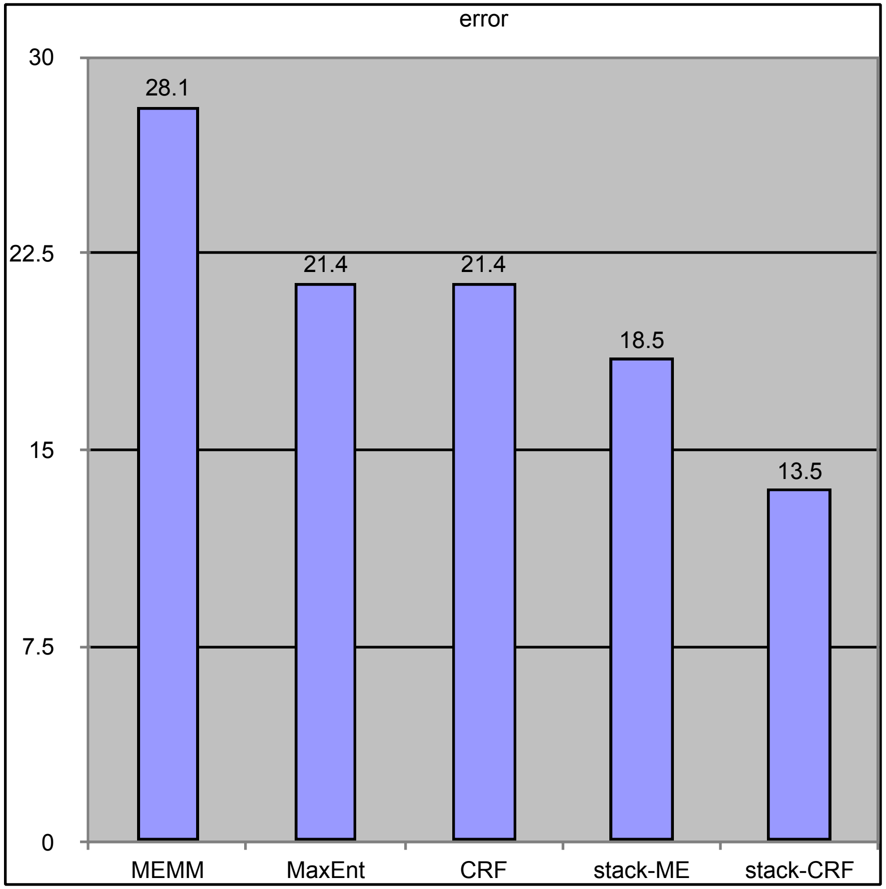
Stacking usually* improves or leaves unchanged

- MaxEnt (p>0.98)
- VotedPerc (p>0.98)
- VPHMM (p>0.98)
- CRFs (p>0.92)

*on a randomly chosen problem using a 1-tailed sign test

Chapter 4b, in which the experiment above is repeated again for yet *one more* new domain....

- Classify pop songs as "happy" or "sad"
- 1-second long song "*frames*" inherit the mood of their containing song
- Song frames are classified with a sequential classifier
- Song mood is majority class of all its frames
- 52,188 frames from 201 songs, 130 features per frame, used *k=5, w=25*



Epilog: in which the speaker discusses certain issues of possible interest to the listener, who is now fully informed of the technical issues (or it may be, only better rested) and thus receptive to such commentary

- Scope:
  - we considered only *segmentation tasks*—sequences with long runs of *identical* labels—and 2-class problems.
  - MEMM fails here.
- Issue:
  - learner is brittle w.r.t. assumptions
  - training data for local model is assumed to be error-free, which is systematically wrong
- Solution: *sequential stacking*
  - model-free way to improve robustness
  - stacked MaxEnt outperforms or ties CRFs on 8/10 tasks; stacked VP outperforms CRFs on 8/9 tasks.
  - a meta-learning method applies to any base learner, and can also reduce error of CRF substantially
  - experiments with non-segmentation problems (NER) had no large gains

Epilog to the Epilog

- Further experiments: (Mostly due to Zhenzhen Kou)
  - Stacking works fine on arbitrary graphs (vs just sequences)
  - Practical advantages
    - Feature construction: Allows arbitrary *aggregations* of nearby-label features, which CRFs don't allow
    - Test-time efficiency: cascade of classifiers vs Gibbs, etc.

Search-based Structured Prediction

Hal Daumé III HDAUME@ISI.EDU
Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, CA 90292 USA

John Langford JL@HUNCH.NET
Toyota Technological Institute at Chicago, 1427 East 60th Street, Chicago, IL, 60637 USA

Daniel Marcu MARCU@ISI.EDU
Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, CA 90292 USA

Stacked Sequential Learning

- MEMM's can perform badly
  - e.g. when there are long runs of identical labels
  - Diagnosis: a mismatch between the training and test data
    - Clean previous-state features at training
    - Noisy previous-state features at test time
- Cure: eliminate the mismatch
  - Train a first classifier *f* to predict previous-state values
    - A plain 'ol classifier, nothing fancy here
  - ~~Train a second classifier *f'* to that uses the predictions of *f*~~
    - At test time: use predictions of *f*
    - ~~As training time: use predictions of *f'* in cross-validating the training data~~
  - Still have *one* classifier that uses its *own* predictions
  - ***How to we train a classifier using its own predictions as input?***

Stacked Sequential Learning → SEARN

1. Use clean previous-state features to train a next-state classifier f_1
   - i.e., $f1$ is an MEMM
2. Use a mixture of clean previous-state features and predictions from f_1 to train a new next-state classifier f_2
3. Use a mixture of previous-state features predicted by f_1 and f_2 to train a new next-state classifier f_3
4. Use a mixture of previous-state features predicted by f_2 and f_3 to train a new next-state classifier f_4

- ….

Stacked Sequential Learning → SEARN

1. Use previous-state features predicted by f_0 (where f_0=the training labels) to train a new next-state classifier f_1
2. Use a mixture* of previous-state features predicted by f_0 and f_1 to train a new next-state classifier f_2
3. Use a mixture of previous-state features predicted by f_1 and f_2 to train a new next-state classifier f_3
4. Use a mixture of previous-state features predicted by f_2 and f_3 to train a new next-state classifier f_4
5. ….

*Mixture of f_i and f_j: flip a coin with bias β. If heads predict using f_i and otherwise use f_j.

For $i,j > 1$ you can pick β to minimize error on a hold-out set.

Stacked Sequential Learning → SEARN

- Let f_0=the training labels (the "optimal policy")
- For i=1,2,….
  - Generate previous-state features with f_{i-1}
  - Train a next-state classifier g_i
  - Let f_i be a mixture of g_i and f_{i-1}

- *If* this converges (i.e., for some i, f_{i-2} and f_{i-1} and f_i are very similar) then f_i was trained on (approximately) its own output.
- *If* g_1 (the first learned classifier) is close to f_0 (the labels) we're on our way to convergence…..

Stacked Sequential Learning → Searn

- MEMM's can perform badly
  - e.g. when there are long runs of identical labels
  - Diagnosis: a mismatch between the training and test data
    - Clean previous-state features at training
    - Noisy previous-state features at test time
- Cure: eliminate the mismatch
  - Train a first classifier *f* to predict previous-state values
    - A plain 'ol classifier, nothing fancy here
  - Train a second classifier *f'* to that uses the predictions of *f*
    - At test time: use predictions of *f*
    - As training time: use predictions of *f* in cross-validating the training data
  - Still have *one* classifier that uses its *own* predictions
  - ***How to we train a classifier using its own predictions as input?***

Solved!

Stacked Sequential Learning → SEARN

- Let f_0=the training labels (the "optimal policy")
- For i=1,2,….
  - Generate previous-state features with f_{i-1}
  - Train a next-state classifier g_i
  - Let f_i be a mixture of g_i and f_{i-1}

- This is a special case of SEARN: in general
  - We can apply this idea to (almost) any task involving a sequential set of *decisions* to be made: NER, parsing, summarization, …
  - We can apply this to many different *loss* functions: F1 for NER, SenseEval scores, ….
  - We only need to get *feedback* on each decision…

Search-based Structured Prediction

Hal Daumé III HDAUME@ISI.EDU
Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, CA 90292 USA

John Langford JL@HUNCH.NET
Toyota Technological Institute at Chicago, 1427 East 60th Street, Chicago, IL, 60637 USA

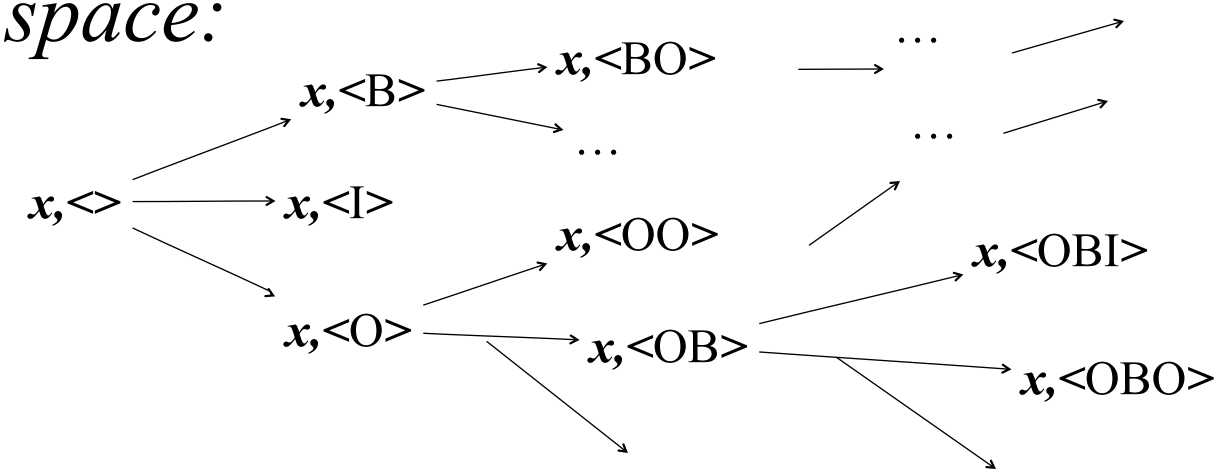
Daniel Marcu MARCU@ISI.EDU
Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, CA 90292 USA

Definitions

- ~~*Structure prediction problem*: distribution over pairs $\boldsymbol{x},\boldsymbol{y}$ ($\boldsymbol{x}$=vector of inputs, $\boldsymbol{y}$=outputs)~~
- *Structure prediction problem (with loss)*: distribution over pairs $\boldsymbol{x},c$
  - c is a function c: $\boldsymbol{y}' \rightarrow R$ (cost of $\boldsymbol{y}'$ relative to $\boldsymbol{y}$)
  - Think of $\boldsymbol{x},c$ as $\boldsymbol{x},\boldsymbol{y}$ and *loss function* L($\boldsymbol{y},\boldsymbol{y}'$)
- *Search space:* graph where nodes are pairs $\boldsymbol{x},<y_1,..,y_j>$ and edges connect $\boldsymbol{x},<y_1,..,y_j>$ and $\boldsymbol{x},<y_1,..,y_{j+1}>$

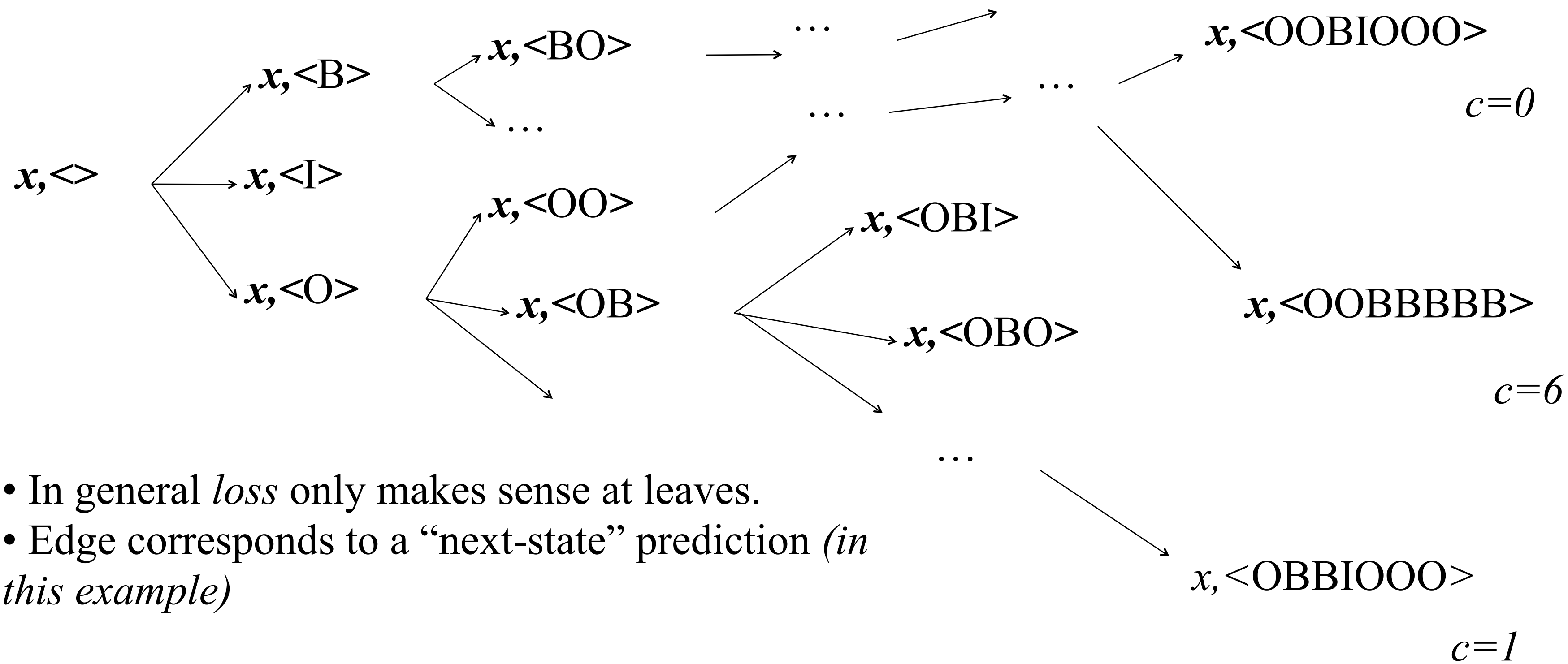
Examples

- *Structure prediction problem (with loss)*: distribution over pairs ***x***,*c*
  - ***x***=*“when will prof cohen post the notes”*,
  - ***c***={c(OOBIOOO)=0, c(OOOBOOO)=0.1, c(OOOOOOO)=1, c(OBIIOOO)=1.2, …}
- *Search space:*



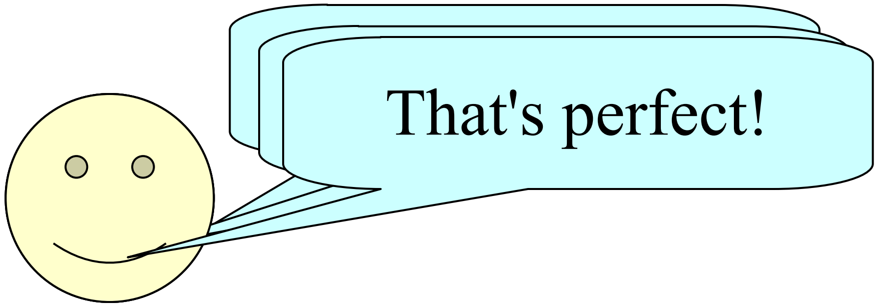
Examples

Search space for ***x*=“*when will prof cohen post the notes*”**

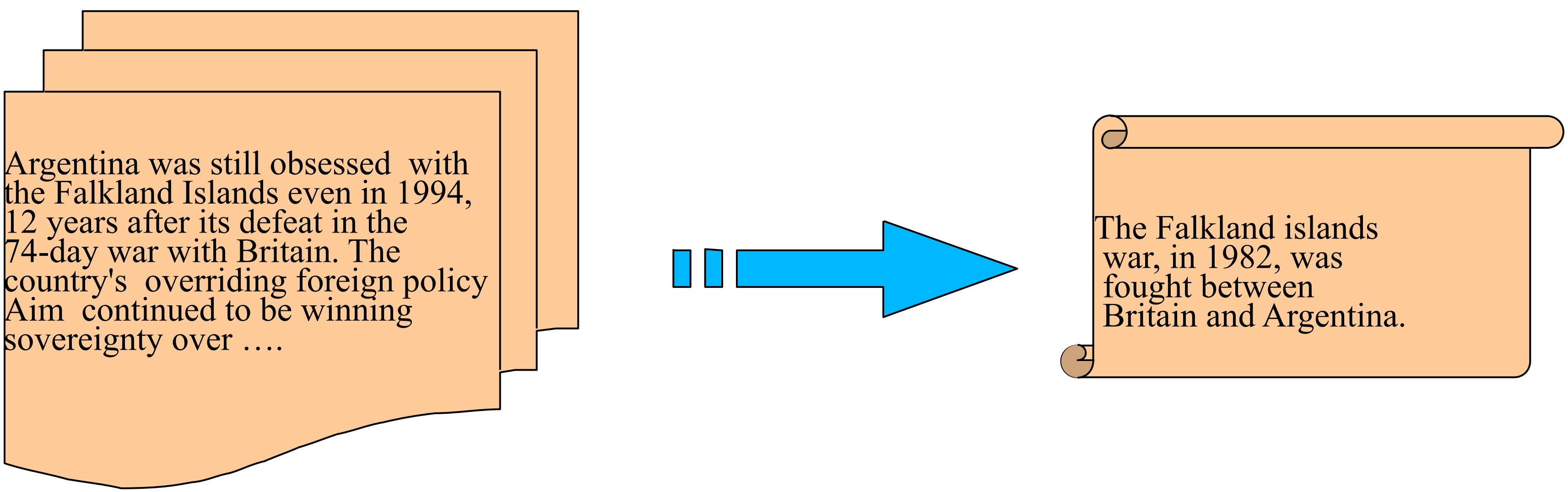


- In general *loss* only makes sense at leaves.
- Edge corresponds to a “next-state” prediction *(in this example)*

Example task: summarization



Standard approach is sentence extraction, but that is often deemed to “coarse” to produce good, very short summaries. We wish to also drop words and phrases => document compression



[D+Langford+Marcu, MLJ09]

[Hal Daume’s slide]

[Hal Daume's slide]

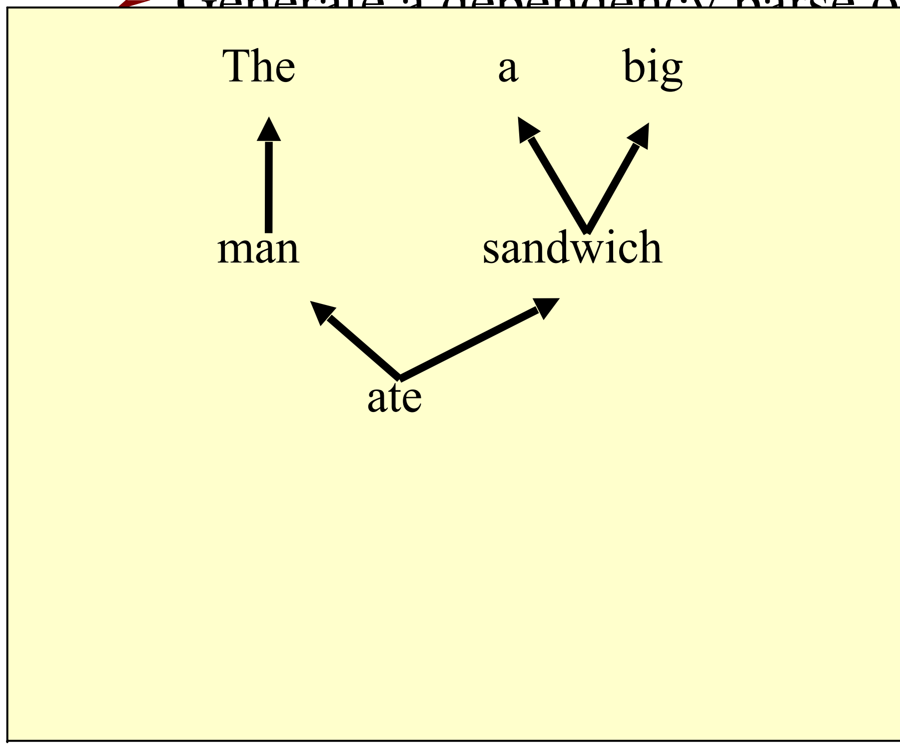
Structure of search

- Lay sentences out sequentially
- Generate a dependency parse of each

de

d to the

ds

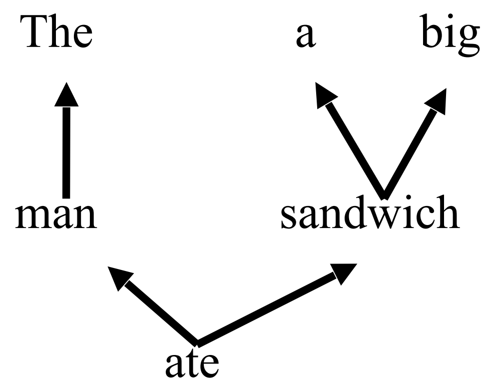


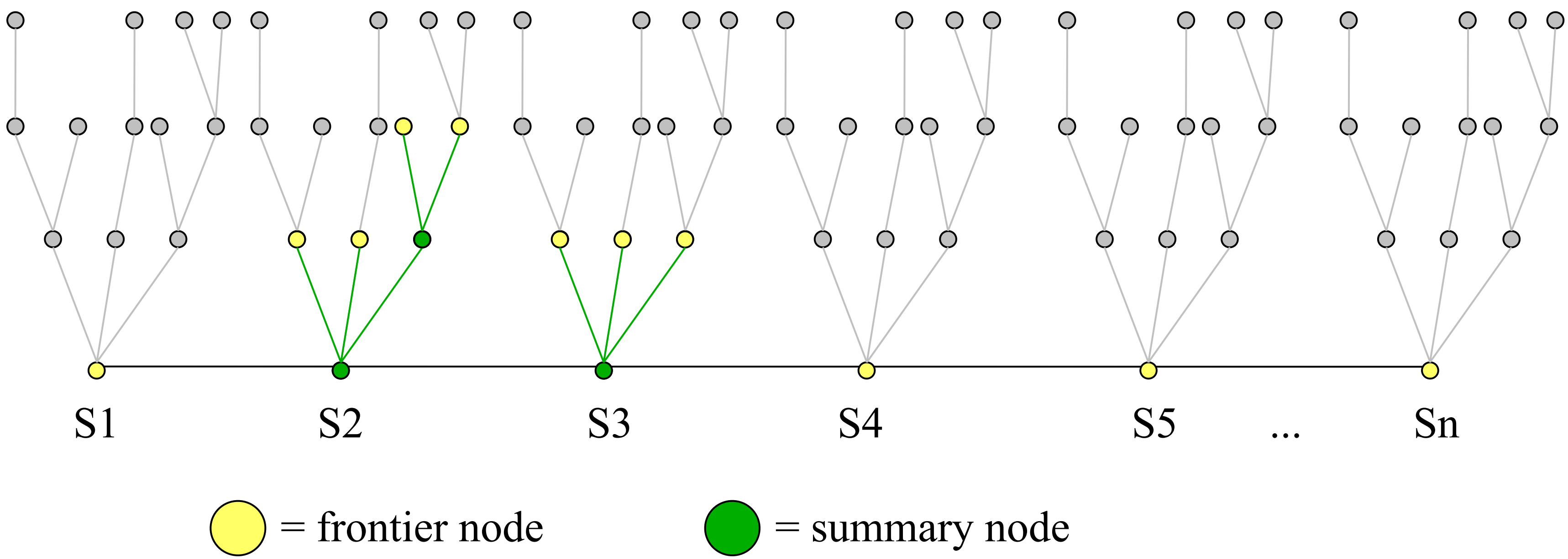
= frontier node = summary node

[D+Langford+Marcu, MLJ09]

[Hal Daume's slide]

Structure of search





[D+Langford+Marcu, MLJ09]

Definitions

- *Structure prediction problem (with loss)*: distribution over pairs $\boldsymbol{x}, c$
- *Search space:* graph where nodes are pairs $\boldsymbol{x}, <y_1,..,y_j>$ and edges …
- *Policy:* function π: $\boldsymbol{x}, y_1,..,y_j \rightarrow y'_{j+1}$
  - Think of this as making a guess at the *next* atomic decision in the sequence.
  - π^* is the *best* policy – i.e., y'_{j+1} is the next decision in the *lowest-cost* completion of $y_1,..,y_j$
  - You can usually work out π^* for the training data

Learning a policy

- A policy is a classifier: h: $(\boldsymbol{x}, y_1, \ldots, y_j) \rightarrow y_{j+1}$
- The *loss* of the classifier h is the loss of the decisions if makes relative to the optimal ones.
- The *loss* of a specific decision y_j versus y^*_j is expected value of Loss(**y'**,**y***) where
  - $\boldsymbol{y'} = < y_1, \ldots, y_{j-1}, y_j, h(\boldsymbol{x}, y_1, \ldots, y_j), h(\boldsymbol{x}, y_1, \ldots, y_j, h(\boldsymbol{x}, y_1, \ldots, y_j)), \ldots >$ i.e. complete the sequence with h
  - $\boldsymbol{y^*} = < y_1, \ldots, y_{j-1}, y^*_j, h(\boldsymbol{x}, y_1, \ldots, y^*_j), h(\boldsymbol{x}, y_1, \ldots, y^*_j, h(\boldsymbol{x}, y_1, \ldots, y^*_j)), \ldots >$ i.e. complete with h

Learning a policy with cost-sensitive learning

- A policy is a classifier: h: $(\mathbf{x}, y_1, \ldots, y_j) \rightarrow y_{j+1}$
- The *loss* of the classifier h is the loss of the decisions if makes relative to the optimal ones.
- The *loss* of a specific decision y_j versus y^*_j is *approximately* Loss(**y'**,**y***) where
  – $\mathbf{y'} = < y_1, \ldots, y_{j-1}, y_j, \pi^*(\mathbf{x}, y_1, \ldots, y_j), \pi^*(\mathbf{x}, y_1, \ldots, y_j, \pi^*(\mathbf{x}, y_1, \ldots, y_j)), \ldots >$ i.e. complete with π^* instead of h …. Just to save some time
  – $\mathbf{y^*} = < y_1, \ldots, y_{j-1}, y^*_j, \pi^*(\mathbf{x}, y_1, \ldots, y^*_j), \pi^*(\mathbf{x}, y_1, \ldots, y^*_j, \pi^*(\mathbf{x}, y_1, \ldots, y^*_j)), \ldots >$ i.e. complete with π^*

Learning a policy with YFCL

- A policy is a (multi-class) classifier: h: $(\boldsymbol{x}, y_1, \ldots, y_j) \rightarrow y_{j+1}$
- We know how to turn a multi-class classification problem to a binary one
- The *loss* of a specific decision y_j versus y^*_j is defined
- We know how to turn a multi-class problem with costs to a standard classification problem
  - By sampling

Algorithm SEARN(S^{SP}, π, A)

Initialize policy $h \leftarrow \pi$

while h has a significant dependence on π **do**

  Initialize the set of cost-sensitive examples $S \leftarrow \emptyset$

  for $(x, y) \in S^{\mathrm{SP}}$ **do**

    Compute predictions under the current policy $\hat{y} \sim x, h$

    for $t = 1 \dots T_x$ **do**

      Compute features $\Phi = \Phi(s_t)$ for state $s_t = (x, y_1, ..., y_t)$

      Initialize a cost vector $c = \langle\rangle$

      for each possible action a **do**

        Let the cost ℓ_a for example x, c at state s be $\ell_h(c, s, a)$

      end for

      Add cost-sensitive example (Φ, ℓ) to S

    end for

  end for

  Learn a classifier on S: $h' \leftarrow A(S)$

  Interpolate: $h \leftarrow \beta h' + (1 - \beta) h$

end while

return h_{last} without π

The optimal policy

While chance of picking the original optimal policy in the current mixture > small number

Cost of this decision vs best decision compared to cost of ***best*** *choice wrt this policy*

Fig. 1 Complete SEARN Algorithm

It is important in the analysis to refer explicitly to the error of the classifiers learned during SEARN process. Let SEARN($\mathcal{D}, h$) denote the distribution over classification problems generated by running SEARN with policy h on distribution $\mathcal{D}$. Also let $\ell_h^{\mathrm{CS}}(h')$ denote the loss of classifier h' on the distribution SEARN($\mathcal{D}, h$). Let the average cost sensitive loss over I iterations be:

T=length of examples ***x***

$$\ell_{\mathrm{avg}} = \frac{1}{I}\sum_{i=1}^{I} \ell_{h_i}^{\mathrm{cs}}(h_i') \tag{4}$$

where h_i is the ith policy and h_i' is the classifier learned on the ith iteration.

Theorem 2 *For all $\mathcal{D}$ with $c_{max} = \mathbb{E}_{(x,\mathbf{c})\sim\mathcal{D}} \max_{\boldsymbol{y}} c_{\boldsymbol{y}}$ (with $(x, \mathbf{c})$ as in Def 1), for all learned cost sensitive classifiers h', SEARN with $\beta = 1/T^3$ and $2T^3 \ln T$ iterations, outputs a learned policy with loss bounded by:*

$$L(\mathcal{D}, h_{last}) \leq L(\mathcal{D}, \pi) + 2T\ell_{avg} \ln T + (1 + \ln T)c_{max}/T$$

Best we can do

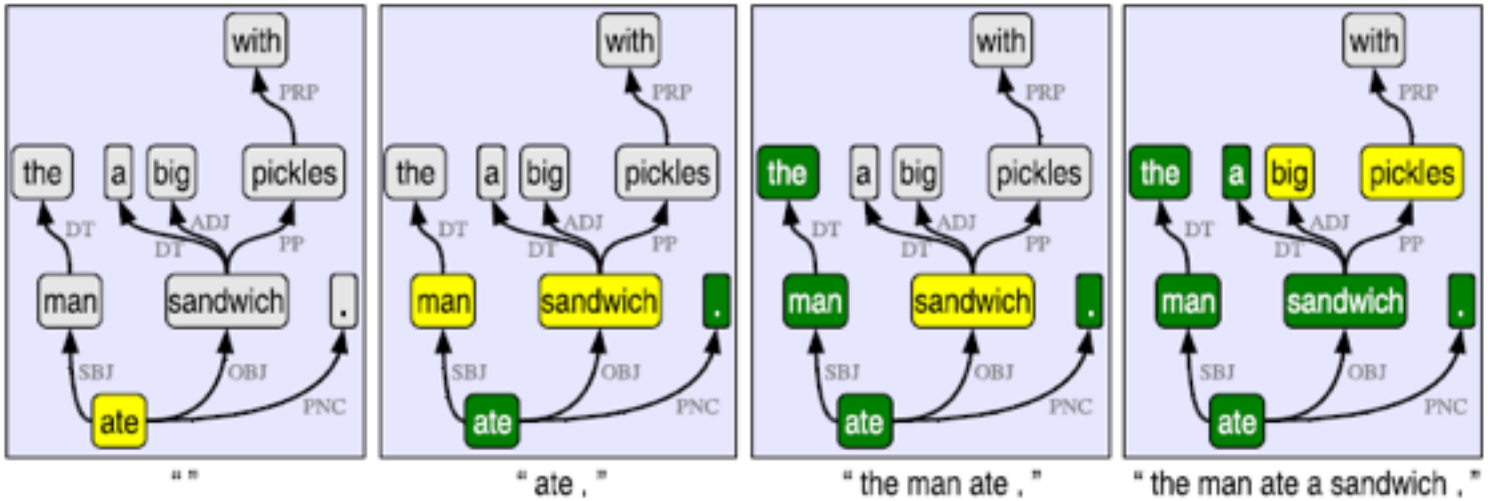
Avg non-sequential classifier errors

"Scale " of loss

ALGORITHM	Handwriting		NER		Chunk	C+T
	Small	Large	Small	Large		
CLASSIFICATION						
Perceptron	65.56	70.05	91.11	94.37	83.12	87.88
Log Reg	68.65	72.10	93.62	96.09	85.40	90.39
SVM-Lin	75.75	82.42	93.74	97.31	86.09	93.94
SVM-Quad	82.63	82.52	85.49	85.49	∼	∼
STRUCTURED						
Str. Perc.	69.74	74.12	93.18	95.32	92.44	93.12
CRF	–	–	94.94	∼	94.77	96.48
SVM$^{\text{struct}}$	–	–	94.90	∼	–	–
M^{3}N-Lin	81.00	∼	–	–	–	–
M^{3}N-Quad	87.00	∼	–	–	–	–
SEARN						
Perceptron	70.17	76.88	95.01	97.67	94.36	96.81
Log Reg	73.81	79.28	95.90	98.17	94.47	96.95
SVM-Lin	82.12	90.58	95.91	98.11	94.44	96.98
SVM-Quad	87.55	90.91	89.31	90.01	∼	∼

Table 1 Empirical comparison of performance of alternative structured prediction algorithms against SEARN on sequence labeling tasks. (Top) Comparison for whole-sequence 0/1 loss; (Bottom) Comparison for individual losses: Hamming for handwriting and Chunking+Tagging and F for NER and Chunking. SEARN is always optimized for the appropriate loss.

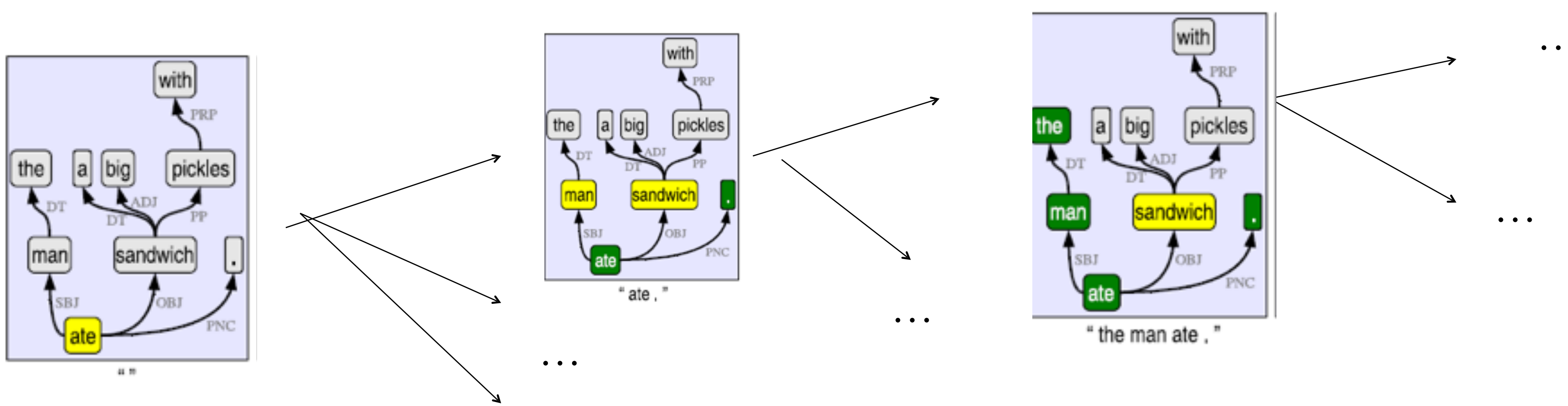
A non-sequential search example: summarization



- Parse each sentence with a dependence parser
- While the summary could be longer:
    - Add a root node *or* a child of a previously-picked node
- Loss: Rouge2 (bi-gram overlap), approximated w/ search

A non-sequential example

- *Structure prediction problem (with loss)*: distribution over pairs ***x***,*c*
  - ***x***=*"Search-based structured …. " // long story*
  - ***c***=**c**:*summary* ***y***→*R*
- *Search space:*



6.2.4 Feature Functions Features in the vine-growth model may consider any aspect of the currently generated summary, and any part of the input document set. These features include simple lexical features: word identity, stem and part of speech of the word under consideration, the syntactic relation with its parent, the position and length of the sentence it appears in, whether it appears in quotes, the length of the document it appears in, the number of pronouns and attribution verbs in the subtree rooted at the word. The features also include language model probabilities for: the word, sentence and subtree under language models derived from the query, a BAYESUM representation of the query, and the existing partial summary.

	ORACLE		**SEARN**		**BAYESUM**			
	Vine	Extr	Vine	Extr	D05	D03	Base	Best
100 w	.0729	.0362	.0415	.0345	.0340	.0316	.0181	-
250 w	.1351	.0809	.0824	.0767	.0762	.0698	.0403	.0725

Table 2 Summarization results; values are Rouge 2 scores (higher is better).

Summary of Searn

- Generalizes MEMM/Maxent tagging model
  - Structured prediction is a *sequence* of atomic *decisions*, each of which potentially depends on the previous ones
  - Applies to a number of tasks that don't allow efficient inference (e.g., joint POS/NPChunk tag assignment)
  - Allows flexibility with cost function as long as you can do "credit assignment" (i.e., associate changes in cost with atomic decisions)

Summary of Searn

- Key ideas
  - Structured prediction is a *sequence* of atomic *decisions*, each of which potentially depends on the previous ones
  - *Learn* to make decisions using cost-sensitive multiclass classification (YFCL)
  - Train a classifier on its on output (approximately)
    - Iterative scheme
    - Start with "clean" data on decisions, gradually mix in data generated from previous iterations of the training