Algorithms for NLP

Parsing / Classification I

Taylor Berg-Kirkpatrick – CMU

Slides: Dan Klein – UC Berkeley
Latent Variable Grammars

Parse Tree $T$

Sentence $w$

Derivations $t : T$

Parameters $\theta$
EM algorithm:

- Brackets are known
- Base categories are known
- Only induce subcategories

Just like Forward-Backward for HMMs.
Number of Phrasal Subcategories
Learned Splits

- **Proper Nouns (NNP):**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NNP-12</td>
<td>John</td>
<td>Robert</td>
<td>James</td>
</tr>
<tr>
<td>NNP-2</td>
<td>J.</td>
<td>E.</td>
<td>L.</td>
</tr>
<tr>
<td>NNP-1</td>
<td>Bush</td>
<td>Noriega</td>
<td>Peters</td>
</tr>
<tr>
<td>NNP-15</td>
<td>New</td>
<td>San</td>
<td>Wall</td>
</tr>
<tr>
<td>NNP-3</td>
<td>York</td>
<td>Francisco</td>
<td>Street</td>
</tr>
</tbody>
</table>

- **Personal pronouns (PRP):**

<table>
<thead>
<tr>
<th>PRP-0</th>
<th>it</th>
<th>He</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRP-1</td>
<td>it</td>
<td>he</td>
<td>they</td>
</tr>
<tr>
<td>PRP-2</td>
<td>it</td>
<td>them</td>
<td>him</td>
</tr>
</tbody>
</table>
Learned Splits

- **Relative adverbs (RBR):**

<table>
<thead>
<tr>
<th>RBR</th>
<th>further</th>
<th>lower</th>
<th>higher</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBR-0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RBR-1</td>
<td>more</td>
<td>less</td>
<td>More</td>
</tr>
<tr>
<td>RBR-2</td>
<td>earlier</td>
<td>Earlier</td>
<td>later</td>
</tr>
</tbody>
</table>

- **Cardinal Numbers (CD):**

<table>
<thead>
<tr>
<th>CD</th>
<th>one</th>
<th>two</th>
<th>Three</th>
</tr>
</thead>
<tbody>
<tr>
<td>CD-7</td>
<td>1989</td>
<td>1990</td>
<td>1988</td>
</tr>
<tr>
<td>CD-4</td>
<td>million</td>
<td>billion</td>
<td>trillion</td>
</tr>
<tr>
<td>CD-11</td>
<td>1</td>
<td>50</td>
<td>100</td>
</tr>
<tr>
<td>CD-0</td>
<td>1</td>
<td>30</td>
<td>31</td>
</tr>
<tr>
<td>CD-3</td>
<td>78</td>
<td>58</td>
<td>34</td>
</tr>
</tbody>
</table>
## Final Results (Accuracy)

<table>
<thead>
<tr>
<th>Language</th>
<th>Method</th>
<th>≤ 40 words</th>
<th>all</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENG</td>
<td>Charniak&amp;Johnson ‘05 (generative)</td>
<td>90.1</td>
<td>89.6</td>
</tr>
<tr>
<td></td>
<td>Split / Merge</td>
<td>90.6</td>
<td>90.1</td>
</tr>
<tr>
<td>GER</td>
<td>Dubey ‘05</td>
<td>76.3</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Split / Merge</td>
<td>80.8</td>
<td>80.1</td>
</tr>
<tr>
<td>CHN</td>
<td>Chiang et al. ‘02</td>
<td>80.0</td>
<td>76.6</td>
</tr>
<tr>
<td></td>
<td>Split / Merge</td>
<td>86.3</td>
<td>83.4</td>
</tr>
</tbody>
</table>

Still higher numbers from reranking / self-training methods
Efficient Parsing for Hierarchical Grammars
Example: PP attachment
Hierarchical Pruning

split in eight: ...

split in four: ...

split in two: ...

coarse:
1621 min
111 min
35 min
15 min
(no search error)
Other Syntactactic Models
Lexicalized parsers can be seen as producing *dependency trees*

Each local binary tree corresponds to an attachment in the dependency graph
Pure dependency parsing is only cubic [Eisner 99]

Some work on non-projective dependencies
- Common in, e.g. Czech parsing
- Can do with MST algorithms [McDonald and Pereira 05]
Shift-Reduce Parsers

- Another way to derive a tree:

 Parsing
  - No useful dynamic programming search
  - Can still use beam search [Ratnaparkhi 97]
- Assume the number of parses is very small
- We can represent each parse $T$ as a feature vector $\varphi(T)$
  - Typically, all local rules are features
  - Also non-local features, like how right-branching the overall tree is
  - [Charniak and Johnson 05] gives a rich set of features
Classification
Classification

- Automatically make a decision about inputs
  - Example: document → category
  - Example: image of digit → digit
  - Example: image of object → object type
  - Example: query + webpages → best match
  - Example: symptoms → diagnosis
  - ...

- Three main ideas
  - Representation as feature vectors
  - Scoring by linear functions (or not, actually)
  - Learning by optimization
Some Definitions

INPUTS

\[ X_i \]

CANDIDATE SET

\[ \mathcal{Y}(x) \]

CANDIDATES

\[ y \]

TRUE OUTPUTS

\[ y^* \]

OUTPUTS

\[ \{ \text{door, table, ...} \} \]

FEATURE VECTORS

\[ f(x, y) = [0 0 1 0 0 0 1 0 0 0 0 0] \]

- \( x_1 = \text{“the”} \land y = \text{“door”} \)
- \( x_1 = \text{“the”} \land y = \text{“table”} \)
- \( \text{“close” in } x \land y = \text{“door”} \)
- \( y \text{ occurs in } x \)
Features
Feature Vectors

- Example: web page ranking (not actually classification)

\[ x_i = \text{“Apple Computers”} \]

\[ f_i(\text{Apple}) = [0.3 \ 5 \ 0 \ 0 \ 0 \ldots] \]

\[ f_i(\text{Apple Inc.}) = [0.8 \ 4 \ 2 \ 1 \ldots] \]
Sometimes, we think of the input as having features, which are multiplied by outputs to form the candidates

\[ x \quad \ldots \text{win the election} \ldots \]

\[ \text{“f(x)”} \quad [1 
0 
1 
0 
] \quad \text{“win”} \quad \text{“election”} \]

\[ \text{“win the election”} \]

\[ f(\text{SPORTS}) = [1 
0 
1 
0 
0 
0 
0 
0 
0 
0 
0 
0 
0 
0 
0 
0 
] \]

\[ f(\text{POLITICS}) = [0 
0 
0 
0 
0 
1 
0 
1 
0 
0 
0 
0 
0 
0 
0 
0 
] \]

\[ f(\text{OTHER}) = [0 
0 
0 
0 
0 
0 
0 
0 
0 
0 
1 
0 
1 
0 
1 
0 
] \]
Non-Block Feature Vectors

- Sometimes the features of candidates cannot be decomposed in this regular way
- Example: a parse tree’s features may be the productions present in the tree

\[
f( NP \quad NP \quad VP ) = [1 \ 0 \ 1 \ 0 \ 1]
\]

\[
f( NP \quad VP \quad N ) = [1 \ 1 \ 0 \ 1 \ 0 \ 0]
\]

- Different candidates will thus often share features
- We’ll return to the non-block case later
Linear Models
Linear Models: Scoring

- In a linear model, each feature gets a weight \( w \)

\[
\begin{align*}
\text{... win the election ...} \\
\mathbf{f}(\text{POLITICS}) &= [0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0] \\
\text{... win the election ...} \\
\mathbf{f}(\text{SPORTS}) &= [1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0] \\
w &= [1 \ 1 \ -1 \ -2 \ 1 \ -1 \ 1 \ -2 \ -2 \ -1 \ -1 \ 1]
\end{align*}
\]

- We score hypotheses by multiplying features and weights:

\[
\text{score}(\mathbf{y}, \mathbf{w}) = \mathbf{w}^\top \mathbf{f}(\mathbf{y})
\]

\[
\begin{align*}
\text{... win the election ...} \\
\mathbf{f}(\text{POLITICS}) &= [0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0] \\
w &= [1 \ 1 \ -1 \ -2 \ 1 \ -1 \ 1 \ -2 \ -2 \ -1 \ -1 \ 1]
\end{align*}
\]

\[
\text{score}(\text{POLITICS}, \mathbf{w}) = 1 \times 1 + 1 \times 1 = 2
\]
Linear Models: Decision Rule

- **The linear decision rule:**

  \[
  \text{prediction}(\ldots \text{win the election} \ldots, \mathbf{w}) = \arg \max_{y \in \mathcal{Y}(x)} \mathbf{w}^\top f(y)
  \]

  \[
  \text{score}(\text{SPORTS}, \mathbf{w}) = 1 \times 1 + (-1) \times 1 = 0
  \]

  \[
  \text{score}(\text{POLITICS}, \mathbf{w}) = 1 \times 1 + 1 \times 1 = 2
  \]

  \[
  \text{score}(\text{OTHER}, \mathbf{w}) = (-2) \times 1 + (-1) \times 1 = -3
  \]

  \[
  \text{prediction}(\ldots \text{win the election} \ldots, \mathbf{w}) = \text{POLITICS}
  \]

- We’ve said nothing about where weights come from
**Binary Classification**

- Important special case: binary classification
  - Classes are $y=+1/-1$
    
    $$f(x, -1) = -f(x, +1)$$
    
    $$f(x) = 2f(x, +1)$$
  
  - Decision boundary is a hyperplane
    
    $$w^\top f(x) = 0$$

  ![Graph](image.png)
Multiclass Decision Rule

- If more than two classes:
  - Highest score wins
  - Boundaries are more complex
  - Harder to visualize

\[
prediction(x_i, w) = \arg \max_{y \in Y} w^T f_i(y)
\]
Learning
Learning Classifier Weights

- Two broad approaches to learning weights
  - Generative: work with a probabilistic model of the data, weights are (log) local conditional probabilities
    - Advantages: learning weights is easy, smoothing is well-understood, backed by understanding of modeling
  - Discriminative: set weights based on some error-related criterion
    - Advantages: error-driven, often weights which are good for classification aren’t the ones which best describe the data

- We’ll mainly talk about the latter for now
How to pick weights?

- **Goal:** choose “best” vector \( w \) given training data
  - For now, we mean “best for classification”

- The ideal: the weights which have greatest test set accuracy / F1 / whatever
  - But, don’t have the test set
  - Must compute weights from training set

- Maybe we want weights which give best training set accuracy?
  - Hard discontinuous optimization problem
  - May not (does not) generalize to test set
  - Easy to overfit

Though, min-error training for MT does exactly this.
Minimize Training Error?

- A loss function declares how costly each mistake is

\[ \ell_i(y) = \ell(y, y_i^*) \]

- E.g. 0 loss for correct label, 1 loss for wrong label
- Can weight mistakes differently (e.g. false positives worse than false negatives or Hamming distance over structured labels)

- We could, in principle, minimize training loss:

\[ \min_w \sum_i \ell_i \left( \arg\max_y w^T f_i(y) \right) \]

- This is a hard, discontinuous optimization problem
The perceptron algorithm
- Iteratively processes the training set, reacting to training errors
- Can be thought of as trying to drive down training error

The (online) perceptron algorithm:
- Start with zero weights $w$
- Visit training instances one by one
  - Try to classify
    $$\hat{y} = \arg \max_{y \in \mathcal{Y}(x)} w^\top f(y)$$
  - If correct, no change!
  - If wrong: adjust weights
    $$w \leftarrow w + f(y_i^*)$$
    $$w \leftarrow w - f(\hat{y})$$
Example: “Best” Web Page

\[
w = [1 \ 2 \ 0 \ 0 \ 0 \ \ldots]\]

\[x_i = “Apple Computers”\]

\[f_i(\quad) = [0.3 \ 5 \ 0 \ 0 \ \ldots]\]

\[\mathbf{w}^\top \mathbf{f} = 10.3 \quad \hat{y}\]

\[f_i(\quad) = [0.8 \ 4 \ 2 \ 1 \ \ldots]\]

\[\mathbf{w}^\top \mathbf{f} = 8.8 \quad y_i^*\]

\[\mathbf{w} \leftarrow \mathbf{w} + f(y_i^*) - f(\hat{y})\]

\[\mathbf{w} = [1.5 \ 1 \ 2 \ 1 \ \ldots]\]
Examples: Perceptron

- Separable Case
Examples: Perceptron

- Non-Separable Case
Margin
Objective Functions

- **What do we want from our weights?**
  - Depends!
  - So far: minimize (training) errors:

\[
\sum_i \text{step} \left( w^\top f_i(y_i^*) - \max_{y \neq y_i^*} w^\top f_i(y) \right)
\]

- This is the “zero-one loss”
  - Discontinuous, minimizing is NP-complete
  - Maximum entropy and SVMs have other objectives related to zero-one loss
Linear Separators

- Which of these linear separators is optimal?
- Distance of \( x_i \) to separator is its margin, \( m_i \)
- Examples closest to the hyperplane are support vectors
- Margin \( \gamma \) of the separator is the minimum \( m \)
Classification Margin

- For each example $x_i$ and possible mistaken candidate $y$, we avoid that mistake by a margin $m_i(y)$ (with zero-one loss)

$$m_i(y) = w^\top f_i(y_i^*) - w^\top f_i(y)$$

- Margin $\gamma$ of the entire separator is the minimum $m$

$$\gamma = \min_i \left( w^\top f_i(y_i^*) - \max_{y \neq y_i^*} w^\top f_i(y) \right)$$

- It is also the largest $\gamma$ for which the following constraints hold

$$\forall i, \forall y \quad w^\top f_i(y_i^*) \geq w^\top f_i(y) + \gamma \ell_i(y)$$
Maximum Margin

- **Separable SVMs**: find the max-margin $w$

  $$\max_{||w||=1} \gamma$$

  $$\ell_i(y) = \begin{cases} 
  0 & \text{if } y = y_i^* \\
  1 & \text{if } y \neq y_i^* 
  \end{cases}$$

  $$\forall i, \forall y \quad w^T f_i(y_i^*) \geq w^T f_i(y) + \gamma \ell_i(y)$$

- Can stick this into Matlab and (slowly) get an SVM
- Won’t work (well) if non-separable
Why Max Margin?

- Why do this? Various arguments:
  - Solution depends only on the boundary cases, or support vectors (but remember how this diagram is broken!)
  - Solution robust to movement of support vectors
  - Sparse solutions (features not in support vectors get zero weight)
  - Generalization bound arguments
  - Works well in practice for many problems
Max Margin / Small Norm

- Reformulation: find the smallest $w$ which separates data

Remember this condition?

$$
\max_{||w||=1} \gamma
\forall i, y \quad w^\top f_i(y_i^*) \geq w^\top f_i(y) + \gamma \ell_i(y)
$$

- $\gamma$ scales linearly in $w$, so if $||w||$ isn’t constrained, we can take any separating $w$ and scale up our margin

$$
\gamma = \min_{i, y \neq y_i^*} \left[ w^\top f_i(y_i^*) - w^\top f_i(y) \right] / \ell_i(y)
$$

- Instead of fixing the scale of $w$, we can fix $\gamma = 1$

$$
\min_w \frac{1}{2} ||w||^2
\forall i, y \quad w^\top f_i(y_i^*) \geq w^\top f_i(y) + 1 \ell_i(y)
$$
\[ \max_{\|w\|=1} \gamma \]
\[ \forall i, y \quad w^T f_i(y_i^*) \geq w^T f_i(y) + \gamma \ell_i(y) \]
\[ w = \gamma u \]
\[ \gamma = 1/\|u\| \]

\[ \min_{\|\gamma u\|=1} \|u\|^2 \]
\[ \forall i, y \quad u^T f_i(y_i^*) \geq u^T f_i(y) + \ell_i(y) \]

\[ \max_{\|u\|=1} 1/\|u\|^2 \]
\[ \forall i, y \quad u^T f_i(y_i^*) \geq u^T f_i(y) + \gamma \ell_i(y) \]

\[ \min_{\|u\|=1} \frac{1}{2}\|u\|^2 \]
\[ \forall i, y \quad u^T f_i(y_i^*) \geq u^T f_i(y) + \ell_i(y) \]

\[ \max_{\|u\|=1} 1/\|u\|^2 \]
\[ \forall i, y \quad u^T f_i(y_i^*) \geq u^T f_i(y) + \ell_i(y) \]

\[ \min_{\|w\|=1} \frac{1}{2}\|w\|^2 \]
\[ \forall i, y \quad w^T f_i(y_i^*) \geq w^T f_i(y) + \ell_i(y) \]
Soft Margin Classification

- What if the training set is not linearly separable?
- *Slack variables* $\xi_i$ can be added to allow misclassification of difficult or noisy examples, resulting in a *soft margin* classifier.
Maximum Margin

- **Non-separable SVMs**
  - Add slack to the constraints
  - Make objective pay (linearly) for slack:
    \[
    \min_{w, \xi} \frac{1}{2} \|w\|^2 + C \sum_i \xi_i \\
    \forall i, y, \quad w^\top f_i(y_i^*) + \xi_i \geq w^\top f_i(y) + \ell_i(y)
    \]
  - C is called the *capacity* of the SVM – the smoothing knob

- **Learning:**
  - Can still stick this into Matlab if you want
  - Constrained optimization is hard; better methods!
  - We’ll come back to this later

*Note: exist other choices of how to penalize slacks!*
Maximum Margin
Likelihood
Linear Models: Maximum Entropy

- **Maximum entropy (logistic regression)**
  - Use the scores as probabilities:
    \[
    P(y|x, w) = \frac{\exp(w^T f(y))}{\sum_{y'} \exp(w^T f(y'))}
    \]
  - Maximize the (log) conditional likelihood of training data
    \[
    L(w) = \log \prod_i P(y_i^*|x_i, w) = \sum_i \log \left( \frac{\exp(w^T f_i(y_i^*))}{\sum_y \exp(w^T f_i(y))} \right)
    \]
    \[
    = \sum_i \left( w^T f_i(y_i^*) - \log \sum_y \exp(w^T f_i(y)) \right)
    \]
Motivation for maximum entropy:

- Connection to maximum entropy principle (sort of)
- Might want to do a good job of being uncertain on noisy cases...
- ... in practice, though, posteriors are pretty peaked

Regularization (smoothing)

$$\max_w \sum_i \left( w^T f_i(y_i^*) - \log \sum_y \exp(w^T f_i(y)) \right) - k ||w||^2$$

$$\min_w \quad k ||w||^2 - \sum_i \left( w^T f_i(y_i^*) - \log \sum_y \exp(w^T f_i(y)) \right)$$
Loss Comparison
Log-Loss

- If we view maxent as a minimization problem:

\[
\min_w k||w||^2 + \sum_i \left( w^\top f_i(y_i^*) - \log \sum_y \exp(w^\top f_i(y)) \right)
\]

- This minimizes the “log loss” on each example

\[
-\left( w^\top f_i(y_i^*) - \log \sum_y \exp(w^\top f_i(y)) \right) = -\log P(y_i^*|x_i, w)
\]

- One view: log loss is an upper bound on zero-one loss
Remember SVMs...

- We had a constrained minimization

\[
\min_{\mathbf{w}, \xi} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_i \xi_i
\]

\[
\forall i, y, \quad \mathbf{w}^\top \mathbf{f}_i(y_i^*) + \xi_i \geq \mathbf{w}^\top \mathbf{f}_i(y) + \ell_i(y)
\]

- ...but we can solve for \(\xi_i\)

\[
\forall i, y, \quad \xi_i \geq \mathbf{w}^\top \mathbf{f}_i(y) + \ell_i(y) - \mathbf{w}^\top \mathbf{f}_i(y_i^*)
\]

\[
\forall i, \quad \xi_i = \max_y \left( \mathbf{w}^\top \mathbf{f}_i(y) + \ell_i(y) \right) - \mathbf{w}^\top \mathbf{f}_i(y_i^*)
\]

- Giving

\[
\min_{\mathbf{w}} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_i \left( \max_y \left( \mathbf{w}^\top \mathbf{f}_i(y) + \ell_i(y) \right) - \mathbf{w}^\top \mathbf{f}_i(y_i^*) \right)
\]
Hinge Loss

- Consider the per-instance objective:

\[
\min_w k||w||^2 + \sum_i \left( \max_y \left( w^T f_i(y) + \ell_i(y) \right) - w^T f_i(y_i^*) \right)
\]

- This is called the “hinge loss”
  - Unlike maxent / log loss, you stop gaining objective once the true label wins by enough
  - You can start from here and derive the SVM objective
  - Can solve directly with sub-gradient decent (e.g. Pegasos: Shalev-Shwartz et al 07)

Plot really only right in binary case

\[
w^T f_i(y_i^*) - \max_{y \neq y_i^*} \left( w^T f_i(y) \right)
\]
Max vs “Soft-Max” Margin

- **SVMs:**
  \[
  \min_w k||w||^2 - \sum_i \left( w^T f_i(y_i^*) - \max_y (w^T f_i(y) + \ell_i(y)) \right)
  \]
  You can make this zero

- **Maxent:**
  \[
  \min_w k||w||^2 - \sum_i \left( w^T f_i(y_i^*) - \log \sum_y \exp (w^T f_i(y)) \right)
  \]
  … but not this one

- **Very similar! Both try to make the true score better than a function of the other scores**
  - The SVM tries to beat the augmented runner-up
  - The Maxent classifier tries to beat the “soft-max”
Loss Functions: Comparison

- **Zero-One Loss**
  \[
  \sum_i \text{step} \left( w^T f_i(y_i^*) - \max_{y \neq y_i^*} w^T f_i(y) \right)
  \]

- **Hinge**
  \[
  \sum_i \left( w^T f_i(y_i^*) - \max_y \left( w^T f_i(y) + \ell_i(y) \right) \right)
  \]

- **Log**
  \[
  \sum_i \left( w^T f_i(y_i^*) - \log \sum_y \exp \left( w^T f_i(y) \right) \right)
  \]

\[
 w^T f_i(y_i^*) - \max_{y \neq y_i^*} \left( w^T f_i(y) \right)
\]
Separators: Comparison
Structure
Handwriting recognition

Sequential structure

[Slides: Taskar and Klein 05]
The screen was a sea of red

Recursive structure
What is the anticipated cost of collecting fees under the new proposal?

En vertu de nouvelle propositions, quel est le coût prévu de perception des droits?

Combinatorial structure
Structured Models

\[ \text{prediction}(x, w) = \arg \max_{y \in \mathcal{Y}(x)} \text{score}(y, w) \]

\[ \text{score}(y, w) = w^\top f(y) = \sum_p w^\top f(y_p) \]

Assumption:

Score is a sum of local “part” scores

Parts = nodes, edges, productions
CFG Parsing

\[ P(y \mid x) \propto \prod_{A \rightarrow \alpha \in (x, y)} \phi(A \rightarrow \alpha) \]

\[ \prod_{A \rightarrow \alpha \in (x, y)} \exp \left\{ w^\top f(A \rightarrow \alpha) \right\} = \exp \left\{ w^\top f(x, y) \right\} \]

\[ \text{#(NP \rightarrow DT NN)} \]

\[ \text{#(NP \rightarrow IN NP)} \]

\[ \text{#(NN \rightarrow 'sea') \} } \]
What is the anticipated cost of collecting fees under the new proposal?

En vertu de les nouvelle propositions, quel est le coût prévu de perception de le droits?
Common case: you have a black box which computes

\[
prediction(x) = \arg \max_{y \in \mathcal{Y}(x)} w^\top f(y)
\]

at least approximately, and you want to learn \( w \)

Easiest option is the structured perceptron [Collins 01]

- Structure enters here in that the search for the best \( y \) is typically a combinatorial algorithm (dynamic programming, matchings, ILPs, A*…)
- Prediction is structured, learning update is not
Structured Margin (Primal)

Remember our primal margin objective?

$$\min_w \frac{1}{2} \|w\|^2_2 + C \sum_i \left( \max_y \left( w^\top f_i(y) + \ell_i(y) \right) - w^\top f_i(y_i^*) \right)$$

Still applies with structured output space!
Structured Margin (Primal)

Just need efficient loss-augmented decode:

$$\bar{y} = \text{argmax}_y (w^\top f_i(y) + \ell_i(y))$$

$$\min_w \frac{1}{2} \|w\|_2^2 + C \sum_i (w^\top f_i(\bar{y}) + \ell_i(\bar{y}) - w^\top f_i(y_i^*))$$

$$\nabla_w = w + C \sum_i (f_i(\bar{y}) - f_i(y_i^*))$$

Still use general subgradient descent methods! (Adagrad)
Structured Margin (Dual)

- Remember the constrained version of primal:

\[
\min_{w, \xi} \quad \frac{1}{2} \|w\|^2 + C \sum_i \xi_i \\
\forall i, y \quad w^T f_i(y^*_i) \geq w^T f_i(y) + \ell_i(y) - \xi_i
\]

- Dual has a variable for every constraint here
We want:

$$\arg \max_y \ w^\top f(\text{brace}, y) = \text{"brace"}$$

Equivalently:

$$w^\top f(\text{brace}, \text{"brace"}) > w^\top f(\text{brace}, \text{"aaaaa"})$$
$$w^\top f(\text{brace}, \text{"brace"}) > w^\top f(\text{brace}, \text{"aaaab"})$$
$$\ldots$$
$$w^\top f(\text{brace}, \text{"brace"}) > w^\top f(\text{brace}, \text{"zzzzz"})$$
We want:

$$\text{arg max}_y \quad w^\top f\left(\text{‘It was red’}, y\right) = \hat{x}_{ABCD}$$

Equivalently:

$$w^\top f\left(\text{‘It was red’}, \hat{x}_{ABCD}\right) > w^\top f\left(\text{‘It was red’}, \hat{x}_{ABDF}\right)$$
$$w^\top f\left(\text{‘It was red’}, \hat{x}_{ABCD}\right) > w^\top f\left(\text{‘It was red’}, \hat{x}_{ABCD}\right)$$
$$\ldots$$
$$w^\top f\left(\text{‘It was red’}, \hat{x}_{ABCD}\right) > w^\top f\left(\text{‘It was red’}, \hat{x}_{GF}^{FG}\right)$$

a lot!
alignment example

- We want:

\[
\arg \max_y \ w^\top f('What is the', y) = \begin{array}{c}
1 \\ 2 \\ 3 \\
\end{array} \
\]

- Equivalently:

\[
\begin{align*}
w^\top f('What is the', \begin{array}{c}
1 \\ 2 \\ 3 \\
\end{array}) & > w^\top f('What is the', \begin{array}{c}
1 \\ 2 \\ 3 \\
\end{array}) \\
w^\top f('Quel est le', \begin{array}{c}
1 \\ 2 \\ 3 \\
\end{array}) & > w^\top f('Quel est le', \begin{array}{c}
1 \\ 2 \\ 3 \\
\end{array}) \\
\vdots \\
\end{align*}
\]

\[
\begin{align*}
w^\top f('What is the', \begin{array}{c}
1 \\ 2 \\ 3 \\
\end{array}) & > w^\top f('What is the', \begin{array}{c}
1 \\ 2 \\ 3 \\
\end{array}) \\
\end{align*}
\]
a lot!
A constraint induction method [Joachims et al 09]
- Exploits that the number of constraints you actually need per instance is typically very small
- Requires (loss-augmented) primal-decode only

Repeat:
- Find the most violated constraint for an instance:
  \[
  \forall y \quad w^\top f_i(y^*_i) \geq w^\top f_i(y) + \ell_i(y)
  \]
  \[
  \arg\max_y \ w^\top f_i(y) + \ell_i(y)
  \]
- Add this constraint and resolve the (non-structured) QP (e.g. with SMO or other QP solver)
Some issues:

- Can easily spend too much time solving QPs
- Doesn’t exploit shared constraint structure
- In practice, works pretty well; fast like perceptron/MIRA, more stable, no averaging
Likelihood, Structured

\[ L(w) = -k \|w\|^2 + \sum_i \left( w^T f_i(y_i^*) - \log \sum_y \exp(w^T f_i(y)) \right) \]

\[ \frac{\partial L(w)}{\partial w} = -2kw + \sum_i \left( f_i(y_i^*) - \sum_y P(y|x_i)f_i(y) \right) \]

- **Structure needed to compute:**
  - Log-normalizer
  - Expected feature counts
    - E.g. if a feature is an indicator of DT-NN then we need to compute posterior marginals \( P(DT-NN|\text{sentence}) \) for each position and sum

- **Also works with latent variables (more later)**
Comparison

![Graphs showing performance comparison for different methods in Constituency Parsing and Constituency Parsing, Neural CRF. The methods include Cutting Plane, Online Cutting Plane, Online Primal Subgradient & $L_1$, Online Primal Subgradient & $L_2$, Averaged Perceptron, MIRA, Averaged MIRA (MST built-in), and Stochastic Gradient Descent.](image-url)
Option 0: Reranking

Input

N-Best List
(e.g. n=100)

Output

\[ x = \text{"The screen was a sea of red."} \]

Baseline Parser

Non-Structured Classification

[Charniak and Johnson 05]
Reranking

**Advantages:**
- Directly reduce to non-structured case
- No locality restriction on features

```
f(S) =
```

**Disadvantages:**
- Stuck with errors of baseline parser
- Baseline system must produce n-best lists
- But, feedback is possible [McCloskey, Charniak, Johnson 2006]
Another option: express all constraints in a packed form
- Maximum margin Markov networks [Taskar et al 03]
- Integrates solution structure deeply into the problem structure

Steps
- Express inference over constraints as an LP
- Use duality to transform minimax formulation into min-min
- Constraints factor in the dual along the same structure as the primal; alphas essentially act as a dual “distribution”
- Various optimization possibilities in the dual
Example: Kernels

- Quadratic kernels

\[
K(x, x') = (x \cdot x' + 1)^2
\]

\[
= \sum_{i,j} x_i x_j x'_i x'_j + 2 \sum_i x_i x'_i + 1
\]

\[
K(y, y') = (f(y)^\top f(y') + 1)^2
\]
Non-Linear Separators

- Another view: kernels map an original feature space to some higher-dimensional feature space where the training set is (more) separable
Why Kernels?

- Can’t you just add these features on your own (e.g. add all pairs of features instead of using the quadratic kernel)?
  - Yes, in principle, just compute them
  - No need to modify any algorithms
  - But, number of features can get large (or infinite)
  - Some kernels not as usefully thought of in their expanded representation, e.g. RBF or data-defined kernels [Henderson and Titov 05]

- Kernels let us compute with these features implicitly
  - Example: implicit dot product in quadratic kernel takes much less space and time per dot product
  - Of course, there’s the cost for using the pure dual algorithms...