Natural Language Processing

Berkeley
N L P

Classification III

Dan Klein – UC Berkeley

Classification

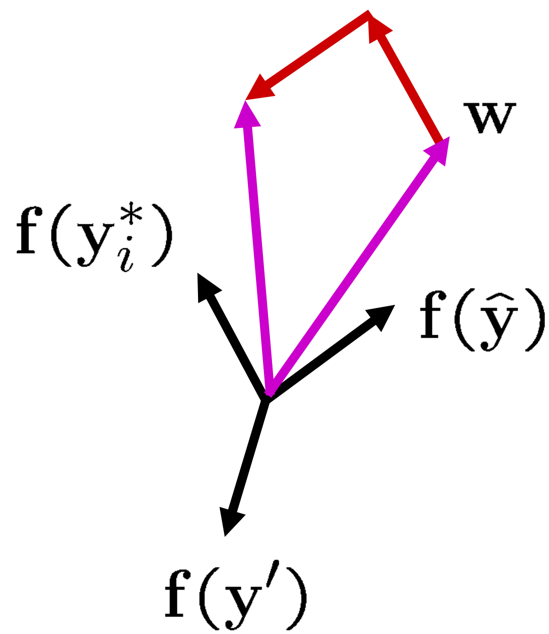
Linear Models: Perceptron

- 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

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y} \in \mathcal{Y}(\mathbf{x})} \mathbf{w}^\top \mathbf{f}(\mathbf{y})$$

    - If correct, no change!
    - If wrong: adjust weights

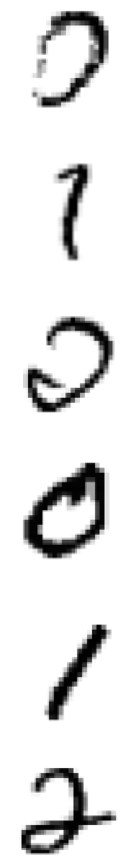
$$\mathbf{w} \leftarrow \mathbf{w} + \mathbf{f}(\mathbf{y}_i^*)$$
$$\mathbf{w} \leftarrow \mathbf{w} - \mathbf{f}(\hat{\mathbf{y}})$$



Duals and Kernels

Nearest-Neighbor Classification

- Nearest neighbor, e.g. for digits:
  - Take new example
  - Compare to all training examples
  - Assign based on closest example



- Encoding: image is vector of intensities:

$$= \langle 0.0 \;\; 0.0 \;\; 0.3 \;\; 0.8 \;\; 0.7 \;\; 0.1 \ldots 0.0 \rangle$$

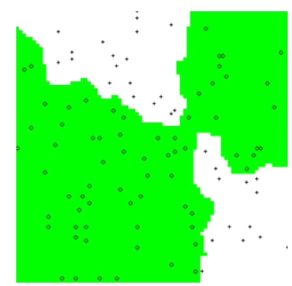
- Similarity function:
  - E.g. dot product of two images' vectors

$$\text{sim}(x, y) = x^\top y = \sum_i x_i y_i$$

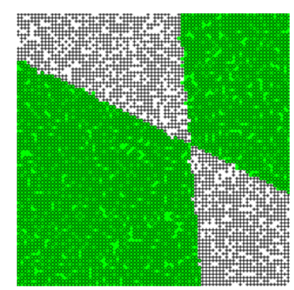
Non-Parametric Classification

- Non-parametric: more examples means (potentially) more complex classifiers

- How about K-Nearest Neighbor?
  - We can be a little more sophisticated, averaging several neighbors
  - But, it's still not really error-driven learning
  - The magic is in the distance function



- Overall: we can exploit rich similarity functions, but not objective-driven learning



A Tale of Two Approaches…

- Nearest neighbor-like approaches
  - Work with data through similarity functions
  - No explicit “learning”
- Linear approaches
  - Explicit training to reduce empirical error
  - Represent data through features
- Kernelized linear models
  - Explicit training, but driven by similarity!
  - Flexible, powerful, very very slow

The Perceptron, Again

- Start with zero weights
- Visit training instances one by one
  - Try to classify

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y} \in \mathcal{Y}(\mathbf{x})} \mathbf{w}^\top \mathbf{f}_i(\mathbf{y})$$

  - If correct, no change!
  - If wrong: adjust weights

$$\mathbf{w} \leftarrow \mathbf{w} + \mathbf{f}_i(\mathbf{y}_i^*)$$
$$\mathbf{w} \leftarrow \mathbf{w} - \mathbf{f}_i(\hat{\mathbf{y}})$$

$$\mathbf{w} \leftarrow \mathbf{w} + (\mathbf{f}_i(\mathbf{y}_i^*) - \mathbf{f}_i(\hat{\mathbf{y}}))$$

$$\mathbf{w} \leftarrow \mathbf{w} + \boxed{\Delta_i(\hat{\mathbf{y}})}$$

mistake vectors

Perceptron Weights

$$\mathbf{w} \leftarrow \mathbf{w} + \Delta_i(\mathbf{y})$$

- What is the final value of w?
  - Can it be an arbitrary real vector?
  - No! It's built by adding up feature vectors (mistake vectors).

$$\mathbf{w} = \Delta_i(\mathbf{y}) + \Delta_{i'}(\mathbf{y}') + \cdots$$

$$\mathbf{w} = \sum_{i,\mathbf{y}} \alpha_i(\mathbf{y}) \Delta_i(\mathbf{y})$$ *mistake counts*

- Can reconstruct weight vectors (the primal representation) from update counts (the dual representation) for each i

$$\alpha_i = \langle \alpha_i(\mathbf{y}_1)\ \ \alpha_i(\mathbf{y}_2)\ \ \ldots\ \ \alpha_i(\mathbf{y}_n) \rangle$$

Dual Perceptron

$$\mathbf{w} = \sum_{i,\mathbf{y}} \alpha_i(\mathbf{y})\Delta_i(\mathbf{y})$$

- Track mistake counts rather than weights
- Start with zero counts (α)
- For each instance x
  - Try to classify

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y} \in \mathcal{Y}(\mathbf{x})} \mathbf{w}^\top \mathbf{f}(\mathbf{y})$$

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y} \in \mathcal{Y}(\mathbf{x}_i)} \sum_{i',\mathbf{y}'} \alpha_{i'}(\mathbf{y}')\Delta_{i'}(\mathbf{y}')^\top \mathbf{f}_i(\mathbf{y})$$

  - If correct, no change!
  - If wrong: raise the mistake count for this example and prediction

$$\alpha_i(\hat{\mathbf{y}}) \leftarrow \alpha_i(\hat{\mathbf{y}}) + 1 \qquad \mathbf{w} \leftarrow \mathbf{w} + \Delta_i(\hat{\mathbf{y}})$$

Dual / Kernelized Perceptron

- How to classify an example x?

$$score(\mathbf{y}) = \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}) = \left(\sum_{i',\mathbf{y}'} \alpha_{i'}(\mathbf{y}') \Delta_{i'}(\mathbf{y}') \right)^\top \mathbf{f}_i(\mathbf{y})$$

$$= \sum_{i',\mathbf{y}'} \alpha_{i'}(\mathbf{y}') \left(\Delta_{i'}(\mathbf{y}')^\top \mathbf{f}_i(\mathbf{y}) \right)$$

$$= \sum_{i',\mathbf{y}'} \alpha_{i'}(\mathbf{y}') \left(\mathbf{f}_{i'}(\mathbf{y}^*_{i'})^\top \mathbf{f}_i(\mathbf{y}) - \mathbf{f}_{i'}(\mathbf{y}')^\top \mathbf{f}_i(\mathbf{y}) \right)$$

$$= \sum_{i',\mathbf{y}'} \alpha_{i'}(\mathbf{y}') \left(K(\mathbf{y}^*_{i'}, \mathbf{y}) - K(\mathbf{y}', \mathbf{y}) \right)$$

- If someone tells us the value of K for each pair of candidates, never need to build the weight vectors

Issues with Dual Perceptron

- Problem: to score each candidate, we may have to compare to *all* training candidates

$$score(\mathbf{y}) = \sum_{i',\mathbf{y}'} \alpha_{i'}(\mathbf{y}') \left(K(\mathbf{y}^*_{i'}, \mathbf{y}) - K(\mathbf{y}', \mathbf{y}) \right)$$

  - Very, very slow compared to primal dot product!
  - One bright spot: for perceptron, only need to consider candidates we made mistakes on during training
  - Slightly better for SVMs where the alphas are (in theory) sparse

- This problem is serious: fully dual methods (including kernel methods) tend to be extraordinarily slow
- Of course, we can (so far) also accumulate our weights as we go...

Kernels: Who Cares?

- So far: a very strange way of doing a very simple calculation
- "Kernel trick": we can substitute any* similarity function in place of the dot product
- Lets us learn new kinds of hypotheses

* Fine print: if your kernel doesn't satisfy certain technical requirements, lots of proofs break. E.g. convergence, mistake bounds. In practice, illegal kernels *sometimes* work (but not always).

Some Kernels

- Kernels implicitly map original vectors to higher dimensional spaces, take the dot product there, and hand the result back

- Linear kernel:

$$K(x, x') = x' \cdot x' = \sum_i x_i \, x'_i$$

- Quadratic kernel:

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

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

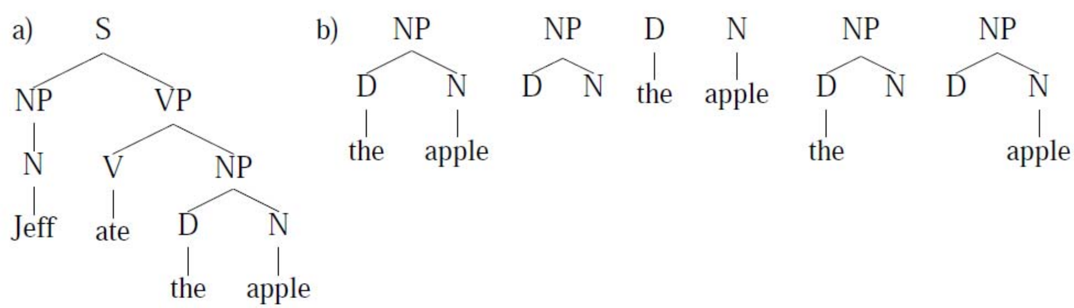
- RBF: infinite dimensional representation

$$K(x, x') = \text{exp}(-||x - x'||^2)$$

- Discrete kernels: e.g. string kernels, tree kernels

Tree Kernels

[Collins and Duffy 01]



- Want to compute number of common subtrees between T, T'
- Add up counts of all pairs of nodes n, n'
  - Base: if n, n' have different root productions, or are depth 0:

$$C(n_1, n_2) = 0$$

  - Base: if n, n' are share the same root production:

$$C(n_1, n_2) = \lambda \prod_{j=1}^{nc(n_1)} (1 + C(ch(n_1, j), ch(n_2, j)))$$

Dual Formulation for SVMs

- We want to optimize: (separable case for now)

$$\min_{\mathbf{w}} \quad \frac{1}{2}||\mathbf{w}||^2$$
$$\forall i, \mathbf{y} \quad \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}_i^*) \geq \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}) + \ell_i(\mathbf{y})$$

- This is hard because of the constraints
- Solution: method of Lagrange multipliers
- The *Lagrangian* representation of this problem is:

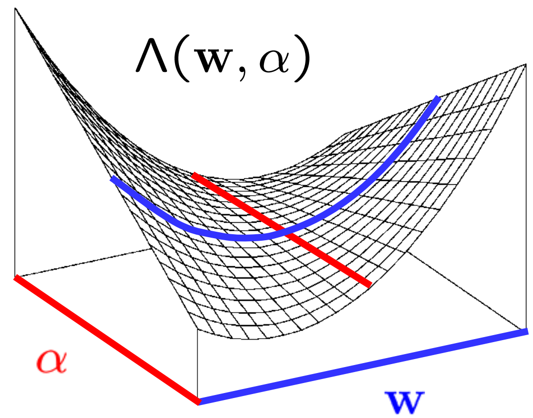
$$\min_{\mathbf{w}} \max_{\alpha \geq 0} \quad \Lambda(\mathbf{w}, \alpha) = \frac{1}{2}||\mathbf{w}||^2 - \sum_{i,\mathbf{y}} \alpha_i(\mathbf{y}) \left(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}_i^*) - \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}) - \ell_i(\mathbf{y})\right)$$

- All we've done is express the constraints as an adversary which leaves our objective alone if we obey the constraints but ruins our objective if we violate any of them

Lagrange Duality

- We start out with a constrained optimization problem:

$$f(\mathbf{w}^*) = \min_{\mathbf{w}} f(\mathbf{w})$$
$$g(\mathbf{w}) \geq 0$$



- We form the Lagrangian:

$$\Lambda(\mathbf{w}, \alpha) = f(\mathbf{w}) - \alpha\, g(\mathbf{w})$$

- This is useful because the constrained solution is a saddle point of Λ (this is a general property):

$$f(\mathbf{w}^*) = \underbrace{\min_{\mathbf{w}} \max_{\alpha \geq 0} \Lambda(\mathbf{w}, \alpha)}_{\text{Primal problem in w}} = \underbrace{\max_{\alpha \geq 0} \min_{\mathbf{w}} \Lambda(\mathbf{w}, \alpha)}_{\text{Dual problem in } \alpha}$$

Primal problem in w   *Dual problem in* α

Dual Formulation II

- Duality tells us that

$$\min_{\mathbf{w}} \max_{\alpha \geq 0} \quad \frac{1}{2}||\mathbf{w}||^2 - \sum_{i,\mathbf{y}} \alpha_i(\mathbf{y}) \left(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}_i^*) - \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}) - \ell_i(\mathbf{y}) \right)$$

has the same value as

$$\max_{\alpha \geq 0} \overbrace{\min_{\mathbf{w}} \quad \frac{1}{2}||\mathbf{w}||^2 - \sum_{i,\mathbf{y}} \alpha_i(\mathbf{y}) \left(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}_i^*) - \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}) - \ell_i(\mathbf{y}) \right)}^{Z(\alpha)}$$

- This is useful because if we think of the α's as constants, we have an unconstrained min in w that we can solve analytically.
- Then we end up with an optimization over α instead of $\mathbf{w}$ (easier).

Dual Formulation III

- Minimize the Lagrangian for fixed α's:

$$\Lambda(\mathbf{w}, \alpha) = \frac{1}{2}||\mathbf{w}||^2 - \sum_{i,\mathbf{y}} \alpha_i(\mathbf{y}) \left(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}_i^*) - \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}) - \ell_i(\mathbf{y})\right)$$

$$\frac{\partial \Lambda(\mathbf{w}, \alpha)}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i,\mathbf{y}} \alpha_i(\mathbf{y}) \left(\mathbf{f}_i(\mathbf{y}_i^*) - \mathbf{f}_i(\mathbf{y})\right)$$

$$\frac{\partial \Lambda(\mathbf{w}, \alpha)}{\partial \mathbf{w}} = 0 \quad \Rightarrow \quad \mathbf{w} = \sum_{i,\mathbf{y}} \alpha_i(\mathbf{y}) \left(\mathbf{f}_i(\mathbf{y}_i^*) - \mathbf{f}_i(\mathbf{y})\right)$$

- So we have the Lagrangian as a function of only α's:

$$\min_{\alpha \geq 0} Z(\alpha) = \frac{1}{2} \left|\left| \sum_{i,\mathbf{y}} \alpha_i(\mathbf{y}) \left(\mathbf{f}_i(\mathbf{y}_i^*) - \mathbf{f}_i(\mathbf{y})\right) \right|\right|^2 - \sum_{i,\mathbf{y}} \alpha_i(\mathbf{y}) \ell_i(\mathbf{y})$$

Back to Learning SVMs

- We want to find α which minimize

$$\min_{\alpha \geq 0} \Lambda(\alpha) = \frac{1}{2} \left\| \sum_{i,\mathbf{y}} \alpha_i(\mathbf{y}) \left(\mathbf{f}_i(\mathbf{y}^i) - \mathbf{f}_i(\mathbf{y}) \right) \right\|^2 - \sum_{i,\mathbf{y}} \alpha_i(\mathbf{y}) \ell_i(\mathbf{y})$$

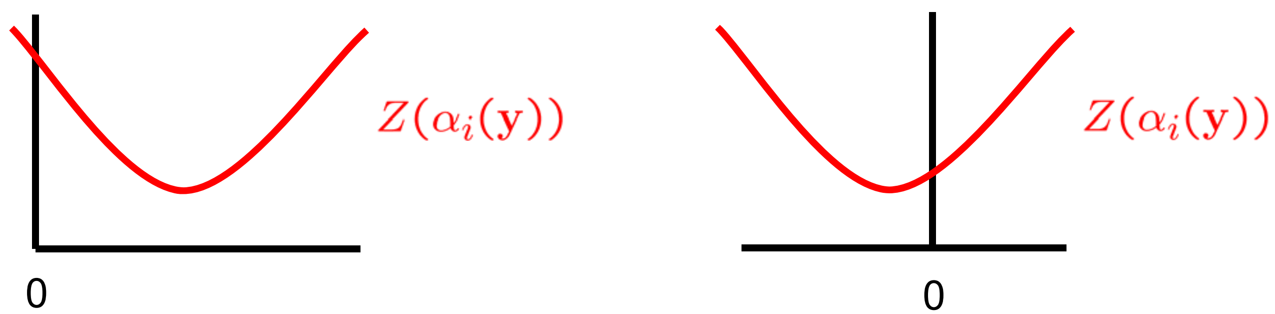
$$\forall i, \quad \sum_{\mathbf{y}} \alpha_i(\mathbf{y}) = C$$

- This is a quadratic program:
  - Can be solved with general QP or convex optimizers
  - But they don't scale well to large problems
  - Cf. maxent models work fine with general optimizers (e.g. CG, L-BFGS)
- How would a special purpose optimizer work?

Coordinate Descent I

$$\min_{\alpha \geq 0} Z(\alpha) = \min_{\alpha \geq 0} \quad \frac{1}{2} \left\| \sum_{i,\mathbf{y}} \alpha_i(\mathbf{y}) \left(\mathbf{f}_i(\mathbf{y}_i^*) - \mathbf{f}_i(\mathbf{y}) \right) \right\|^2 - \sum_{i,\mathbf{y}} \alpha_i(\mathbf{y}) \ell_i(\mathbf{y})$$

- Despite all the mess, Z is just a quadratic in each $\alpha_i(y)$
- Coordinate descent: optimize one variable at a time



- If the unconstrained argmin on a coordinate is negative, just clip to zero…

Coordinate Descent II

- Ordinarily, treating coordinates independently is a bad idea, but here the update is very fast and simple

$$\alpha_i(\mathbf{y}) \leftarrow \max\left(0, \alpha_i(\mathbf{y}) + \frac{\ell_i(\mathbf{y}) - \mathbf{w}^\top\left(\mathbf{f}_i(\mathbf{y}_i^*) - \mathbf{f}_i(\mathbf{y})\right)}{\left|\left|\left(\mathbf{f}_i(\mathbf{y}_i^*) - \mathbf{f}_i(\mathbf{y})\right)\right|\right|^2}\right)$$

- So we visit each axis many times, but each visit is quick

- This approach works fine for the separable case
- For the non-separable case, we just gain a simplex constraint and so we need slightly more complex methods (SMO, exponentiated gradient)

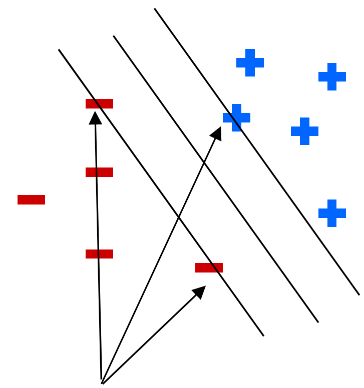
$$\forall i, \quad \sum_{\mathbf{y}} \alpha_i(\mathbf{y}) = C$$

What are the Alphas?

- Each candidate corresponds to a primal constraint

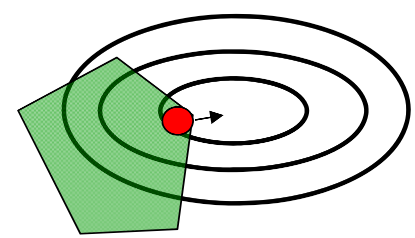
$$\min_{\mathbf{w},\xi} \quad \frac{1}{2}||\mathbf{w}||^2 + C\sum_i \xi_i$$

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



Support vectors

- In the solution, an $\alpha_i(y)$ will be:
  - Zero if that constraint is inactive
  - Positive if that constrain is active
  - i.e. positive on the support vectors



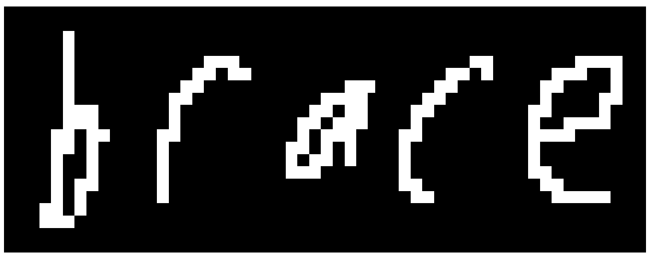
- Support vectors contribute to weights:

$$\mathbf{w} = \sum_{i,\mathbf{y}} \alpha_i(\mathbf{y}) \left(\mathbf{f}_i(\mathbf{y}_i^*) - \mathbf{f}_i(\mathbf{y})\right)$$

Structure

Handwriting recognition

x **y**



→ **brace**

Sequential structure

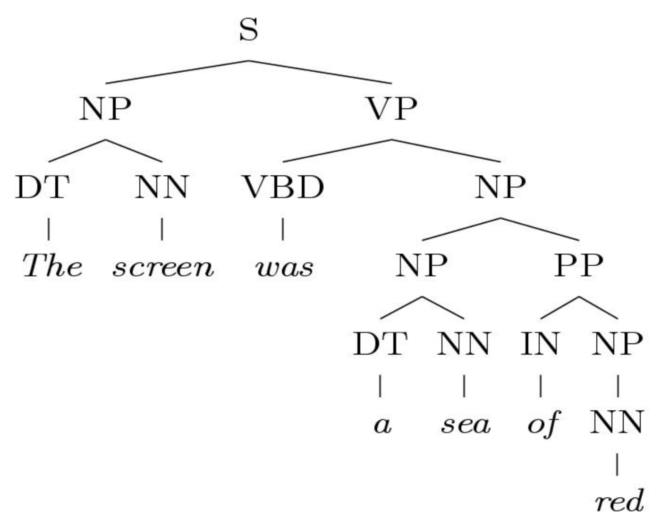
[Slides: Taskar and Klein 05]

CFG Parsing

x

The screen was a sea of red

y



Recursive structure

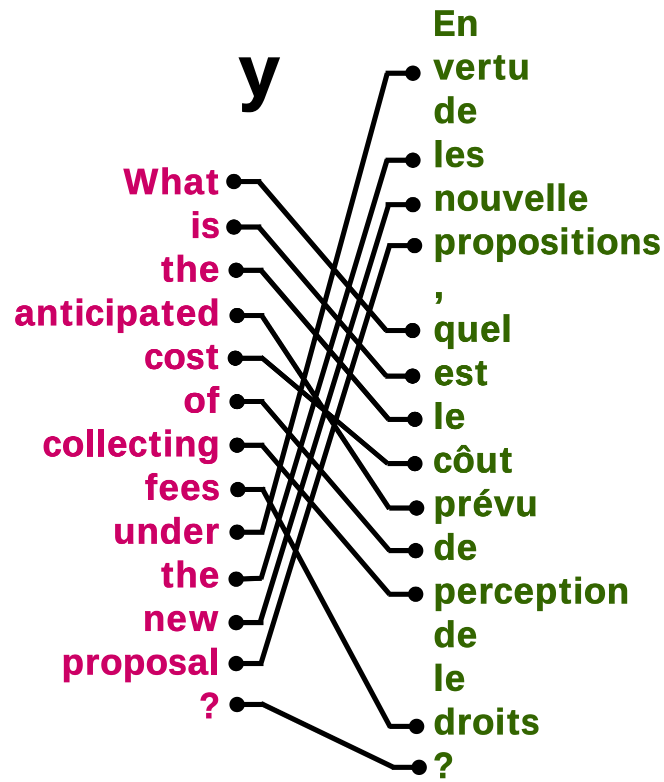
Bilingual Word Alignment

x

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

En vertu de nouvelle propositions, quel est le côut prévu de perception de les droits?

y



Combinatorial structure

Structured Models

$$prediction(\mathbf{x}, \mathbf{w}) = \arg\max_{\mathbf{y} \in \mathcal{Y}(\mathbf{x})} score(\mathbf{y}, \mathbf{w})$$

space of feasible outputs

Assumption:

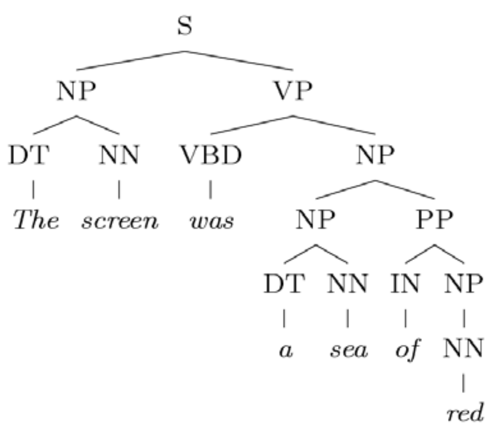
$$score(\mathbf{y}, \mathbf{w}) = \mathbf{w}^\top \mathbf{f}(\mathbf{y}) = \sum_p \mathbf{w}^\top \mathbf{f}(\mathbf{y}_p)$$

Score is a sum of local "part" scores

Parts = nodes, edges, productions

CFG Parsing

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



$$\mathbf{f} : \mathcal{X} \times \mathcal{Y} \rightarrow \Re^d$$

#(NP → DT NN)

...

#(PP → IN NP)

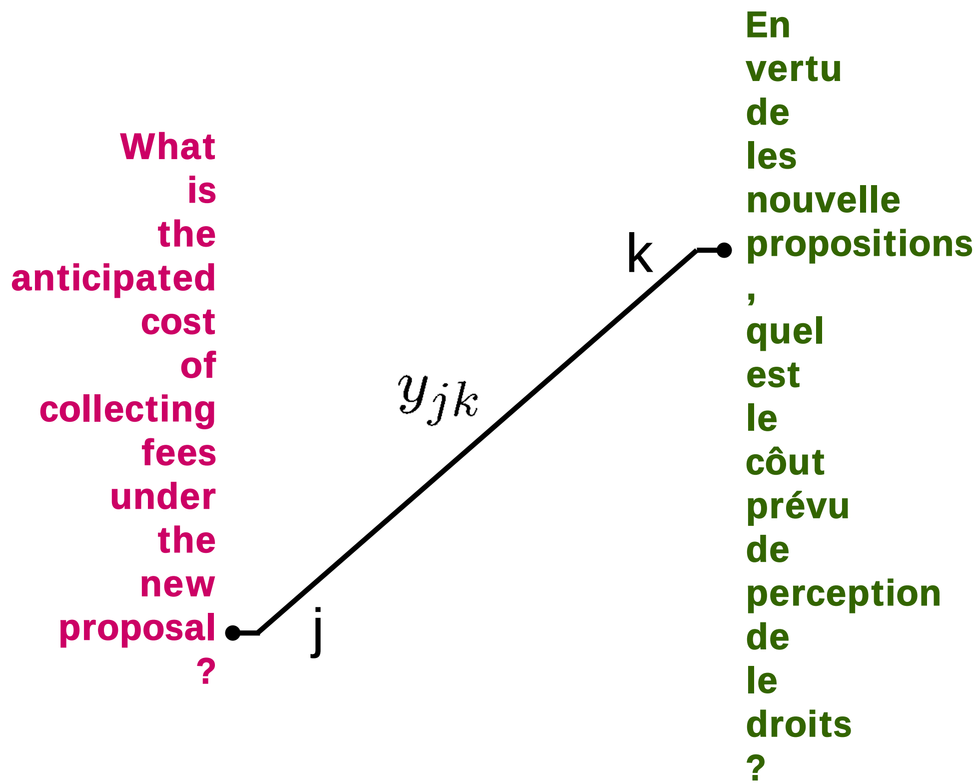
...

#(NN → 'sea')

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

Bilingual word alignment

$$\sum_{y_{jk} \in \mathbf{y}} \mathbf{w}^\top \mathbf{f}(\mathbf{x}_{jk}) = \mathbf{w}^\top \mathbf{f}(\mathbf{x}, \mathbf{y})$$



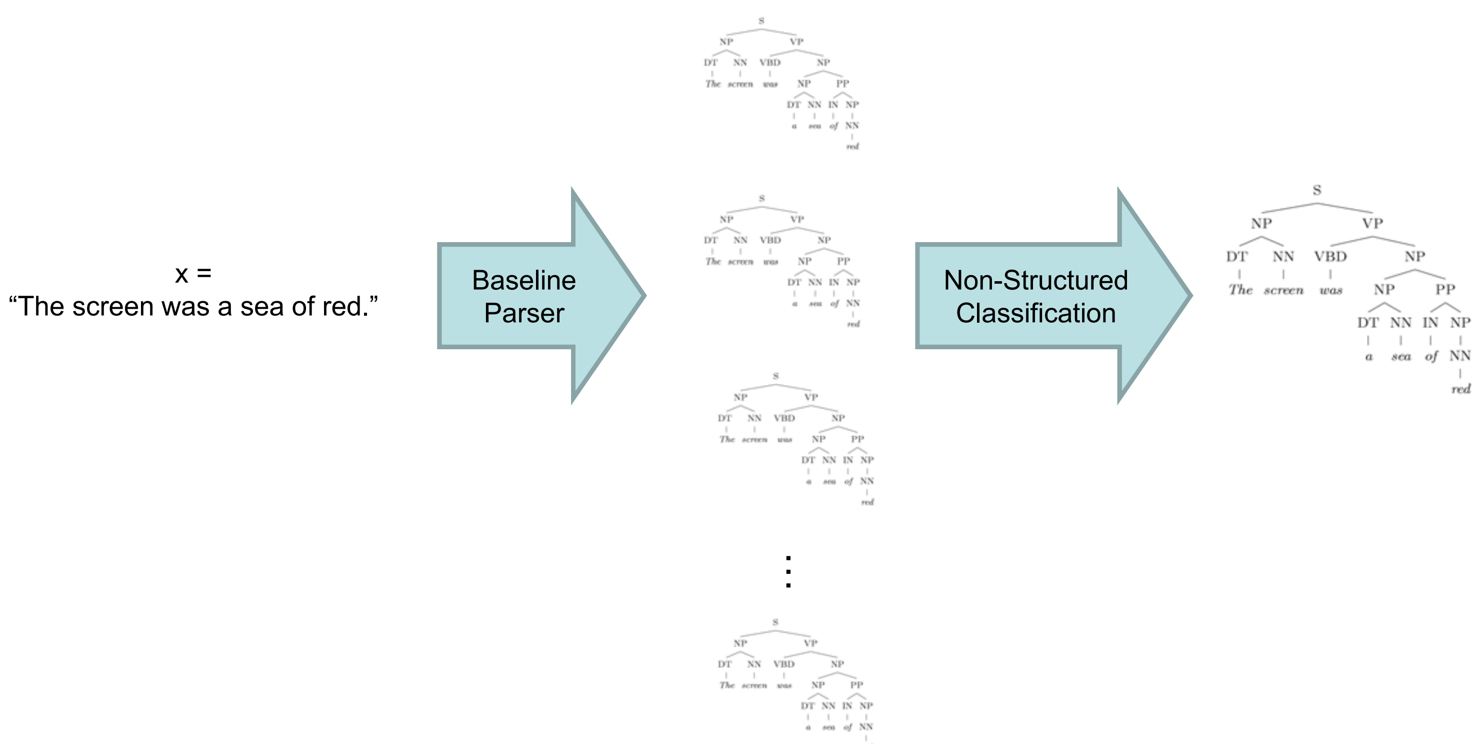
$\mathbf{f}(\mathbf{x}_{jk})$

- association
- position
- orthography

Option 0: Reranking

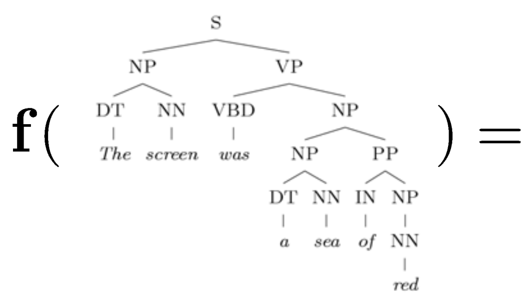
[e.g. Charniak and Johnson 05]

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



Reranking

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



- Disadvantages:
  - Stuck with errors of baseline parser
  - Baseline system must produce n-best lists
  - But, feedback is possible [McCloskey, Charniak, Johnson 2006]

Efficient Primal Decoding

- Common case: you have a black box which computes

$$\text{prediction}(\mathbf{x}) = \arg\max_{\mathbf{y}\in\mathcal{Y}(\mathbf{x})} \mathbf{w}^\top \mathbf{f}(\mathbf{y})$$

  at least approximately, and you want to learn w

- Many learning methods require more (expectations, dual representations, k-best lists), but the most commonly used options do not

- 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

- Remember the margin objective:

$$\min_{\mathbf{w}} \quad \frac{1}{2}||\mathbf{w}||^2$$
$$\forall i, \mathbf{y} \quad \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}_i^*) \geq \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}) + \ell_i(\mathbf{y})$$

- This is still defined, but lots of constraints

Full Margin: OCR

- We want:

$$\arg\max_{\mathbf{y}} \; \mathbf{w}^\top \mathbf{f}(\boxed{\textit{brace}}, \mathbf{y}) = \text{“brace”}$$

- Equivalently:

$$\mathbf{w}^\top \mathbf{f}(\boxed{\textit{brace}}, \text{“brace”}) > \mathbf{w}^\top \mathbf{f}(\boxed{\textit{brace}}, \text{“aaaaa”})$$

$$\mathbf{w}^\top \mathbf{f}(\boxed{\textit{brace}}, \text{“brace”}) > \mathbf{w}^\top \mathbf{f}(\boxed{\textit{brace}}, \text{“aaaab”})$$

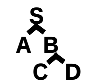
$$\cdots$$

$$\mathbf{w}^\top \mathbf{f}(\boxed{\textit{brace}}, \text{“brace”}) > \mathbf{w}^\top \mathbf{f}(\boxed{\textit{brace}}, \text{“zzzzz”})$$

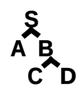
a lot!

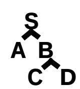
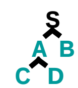
Parsing example

- We want:

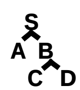
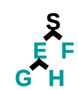
$$\arg\max_{\mathrm{y}}\ \mathrm{w}^\top \mathrm{f}(\text{'It was red'}, \mathrm{y}) =$$ 

- Equivalently:

$\mathrm{w}^\top \mathrm{f}(\text{'It was red'},$  $) > \mathrm{w}^\top \mathrm{f}(\text{'It was red'},$  $)$

$\mathrm{w}^\top \mathrm{f}(\text{'It was red'},$  $) > \mathrm{w}^\top \mathrm{f}(\text{'It was red'},$  $)$

$\ldots$

$\mathrm{w}^\top \mathrm{f}(\text{'It was red'},$  $) > \mathrm{w}^\top \mathrm{f}(\text{'It was red'},$  $)$

} **a lot!**

Alignment example

- We want:

$$\arg\max_{\mathbf{y}} \ \mathbf{w}^\top \mathbf{f}\left(\begin{smallmatrix}\text{'What is the'}\\ \text{'Quel est le'}\end{smallmatrix}, \mathbf{y}\right) = \begin{smallmatrix}1 \leftrightarrow 1\\ 2 \leftrightarrow 2\\ 3 \leftrightarrow 3\end{smallmatrix}$$

- Equivalently:

$$\mathbf{w}^\top \mathbf{f}\left(\begin{smallmatrix}\text{'What is the'}\\ \text{'Quel est le'}\end{smallmatrix}, \begin{smallmatrix}1 \leftrightarrow 1\\ 2 \leftrightarrow 2\\ 3 \leftrightarrow 3\end{smallmatrix}\right) > \mathbf{w}^\top \mathbf{f}\left(\begin{smallmatrix}\text{'What is the'}\\ \text{'Quel est le'}\end{smallmatrix}, \begin{smallmatrix}1 \leftrightarrow 1\\ 2 \leftrightarrow 3\\ 3 \leftrightarrow 2\end{smallmatrix}\right)$$

$$\mathbf{w}^\top \mathbf{f}\left(\begin{smallmatrix}\text{'What is the'}\\ \text{'Quel est le'}\end{smallmatrix}, \begin{smallmatrix}1 \leftrightarrow 1\\ 2 \leftrightarrow 2\\ 3 \leftrightarrow 3\end{smallmatrix}\right) > \mathbf{w}^\top \mathbf{f}\left(\begin{smallmatrix}\text{'What is the'}\\ \text{'Quel est le'}\end{smallmatrix}, \begin{smallmatrix}1 \leftrightarrow 2\\ 2 \leftrightarrow 1\\ 3 \leftrightarrow 3\end{smallmatrix}\right)$$

$$\cdots$$

$$\mathbf{w}^\top \mathbf{f}\left(\begin{smallmatrix}\text{'What is the'}\\ \text{'Quel est le'}\end{smallmatrix}, \begin{smallmatrix}1 \leftrightarrow 1\\ 2 \leftrightarrow 2\\ 3 \leftrightarrow 3\end{smallmatrix}\right) > \mathbf{w}^\top \mathbf{f}\left(\begin{smallmatrix}\text{'What is the'}\\ \text{'Quel est le'}\end{smallmatrix}, \begin{smallmatrix}1 \leftrightarrow 3\\ 2 \leftrightarrow 2\\ 3 \leftrightarrow 1\end{smallmatrix}\right)$$

a lot!

Cutting Plane

- 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:

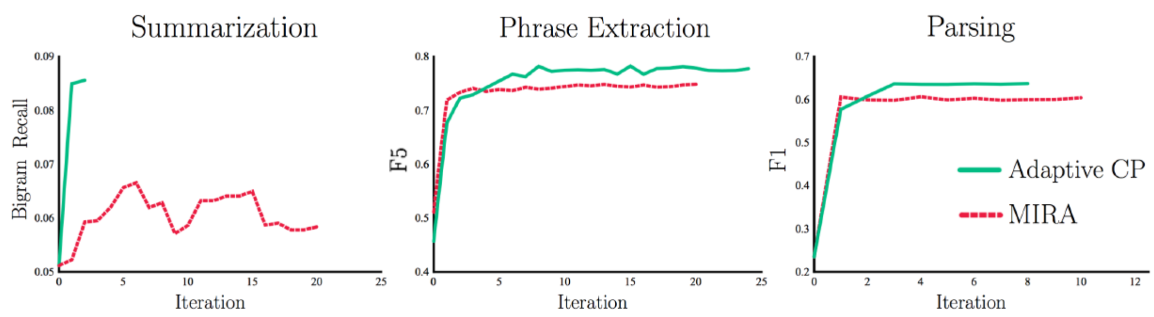
$$\forall \mathbf{y} \quad \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}_i^*) \geq \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}) + \ell_i(\mathbf{y})$$

$$\arg\max_{\mathbf{y}} \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}) + \ell_i(\mathbf{y})$$

  - Add this constraint and resolve the (non-structured) QP (e.g. with SMO or other QP solver)

Cutting Plane

- 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



M3Ns

- 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

Likelihood, Structured

$$L(\mathbf{w}) = -k||\mathbf{w}||^2 + \sum_i \left(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}_i^*) - \log \sum_{\mathbf{y}} \exp(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y})) \right)$$

$$\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}} = -2k\mathbf{w} + \sum_i \left(\mathbf{f}_i(\mathbf{y}_i^*) - \sum_{\mathbf{y}} P(\mathbf{y}|\mathbf{x}_i)\mathbf{f}_i(\mathbf{y}) \right)$$

- 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|sentence) for each position and sum
- Also works with latent variables (more later)