Algorithms for NLP

Classification II

Sachin Kumar - CMU

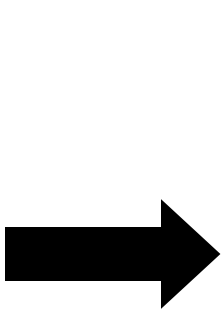
Slides: Dan Klein – UC Berkeley, Taylor Berg-Kirkpatrick – CMU

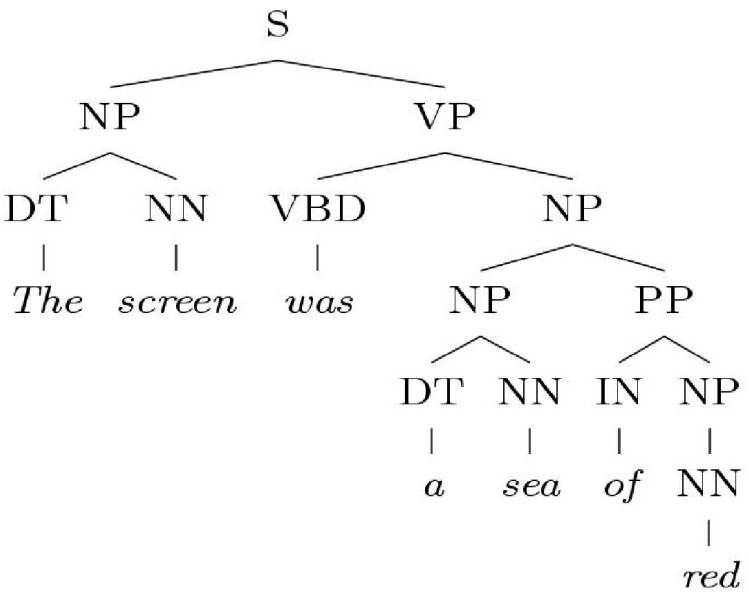
Parsing as Classification

- Input: Sentence X
- Output: Parse Y
- Potentially millions of candidates

x **y**

The screen was a sea of red





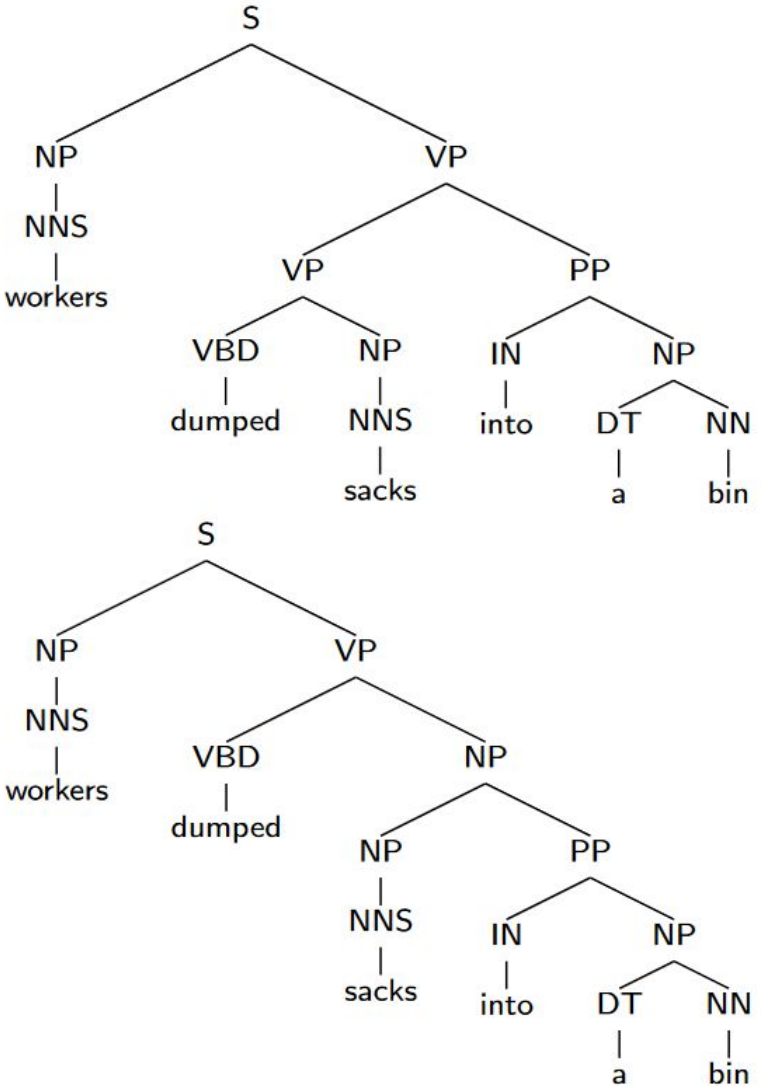
Generative Model for Parsing

- PCFG: Model joint probability P(S, Y)
- Many advantages
  - Learning is often clean and analytical: count and divide

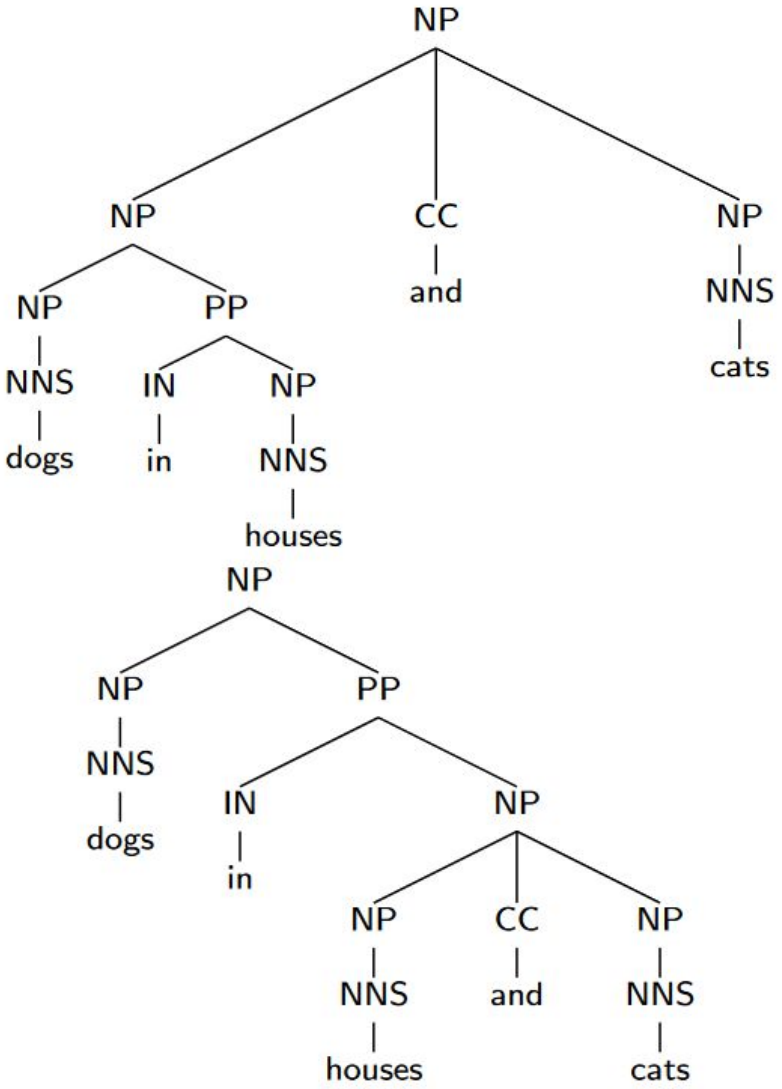
$$p(X \to \alpha) = \frac{C(X \to \alpha)}{C(X)}$$

- Disadvantages?
  - Rigid independence assumption
  - Lack of sensitivity to lexical information
  - Lack of sensitivity to structural frequencies

Lack of sensitivity to lexical information

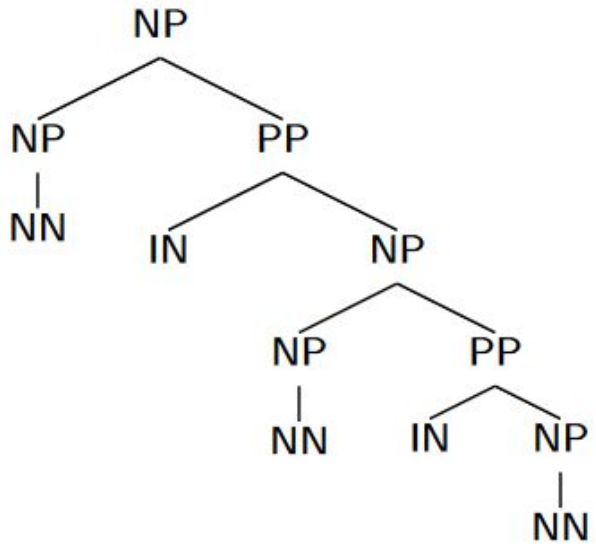


Lack of sensitivity to structural frequencies: Coordination Ambiguity

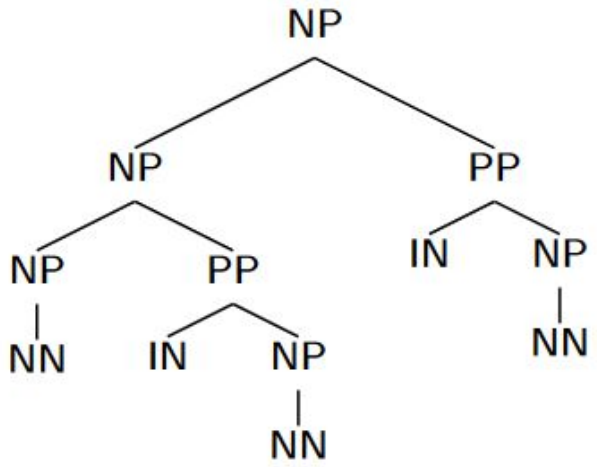


Lack of sensitivity to structural frequencies: Close attachment

(a)



(b)



president of a company in Africa

Discriminative Model for Parsing

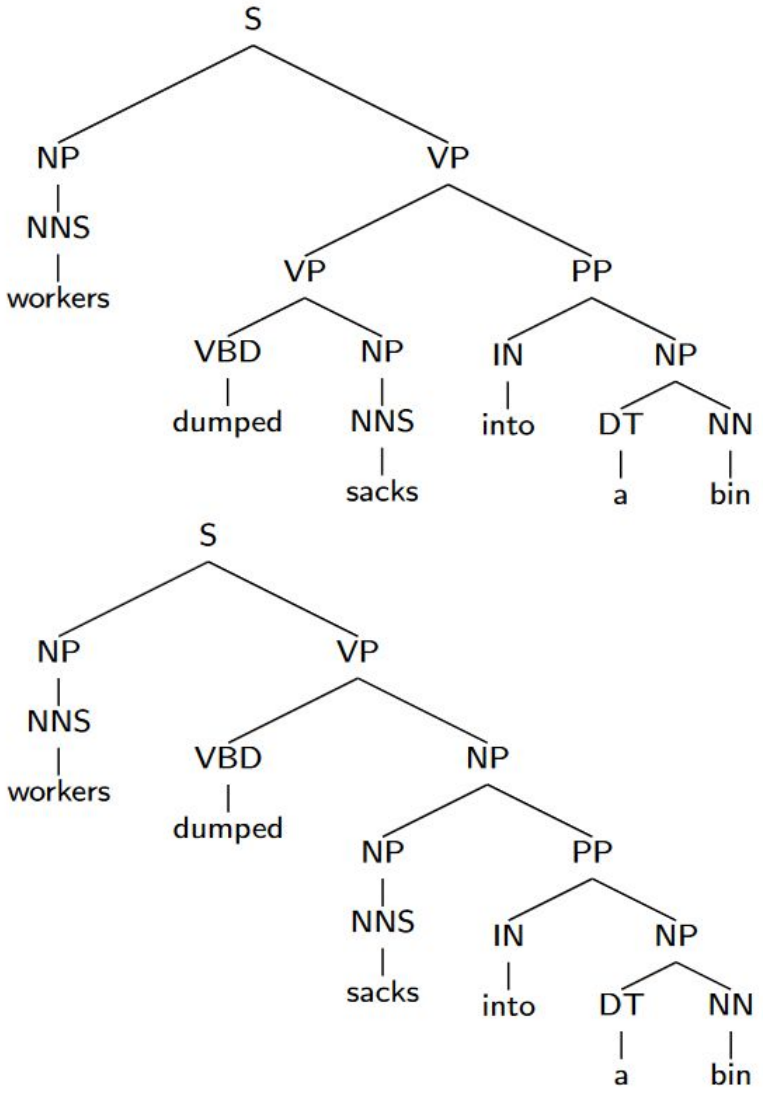
- Directly estimate the score of y given X

$$\hat{y} = \arg \max_{y \in Y(X)} score(X, y)$$

$$\hat{y} = \arg \max_{y \in Y(X)} w^T f(x, y)$$

- Distribution Free: Minimize expected los
- Advantages?
  - We get more freedom in defining features -
    - no independence assumptions required

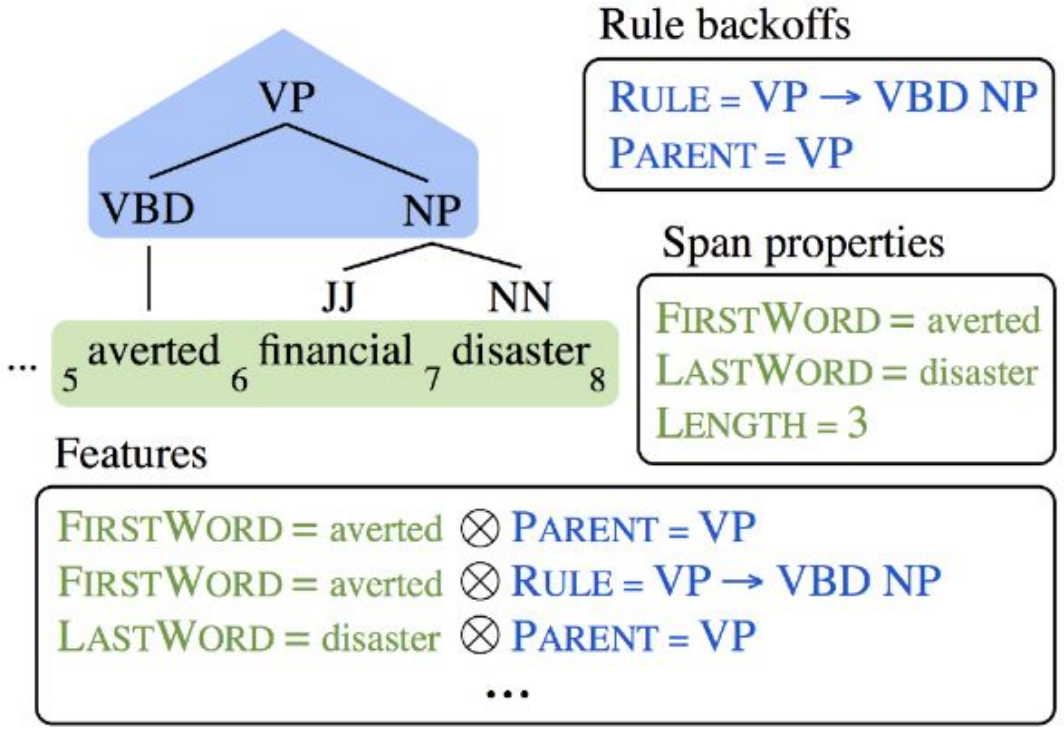
Example: Right branching



Example: Complex Features

Features can be products of multiple components:

- position
- rule
- parent label
- span features
- …



How to train?

- Minimize training error?
  - Loss function for each example i

$$\ell_i(\mathbf{y}) = \ell(\mathbf{y}, \mathbf{y}_i^*)$$

0 when the label is correct, 1 otherwise

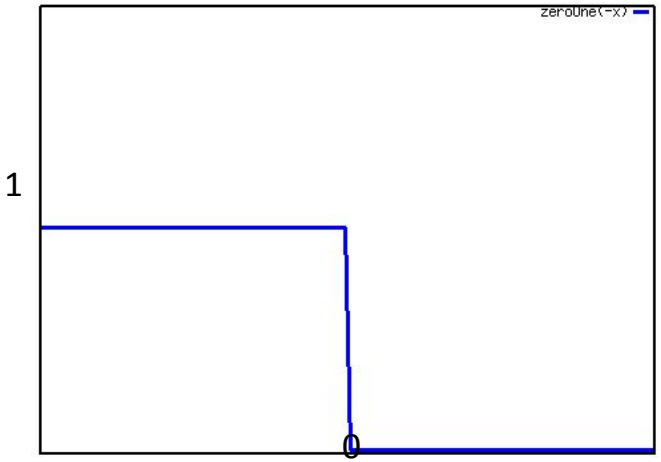
- Training Error to minimize

$$\min_{\mathbf{w}} \sum_i \ell_i \left(\arg\max_{\mathbf{y}} \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}) \right)$$

Objective Function

$$\min_{\mathbf{w}} \sum_i \ell_i \left(\arg\max_{\mathbf{y}} \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}) \right)$$

$$\sum_i step \left(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}_i^*) - \max_{\mathbf{y} \neq \mathbf{y}_i^*} \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}) \right)$$



- step function returns 1 when argument is negative, 0 otherwise
- Difficult to optimize, zero gradients everywhere.
- Solution: Optimize differentiable upper bounds of this function: MaxEnt or SVM

Linear Models: Perceptron

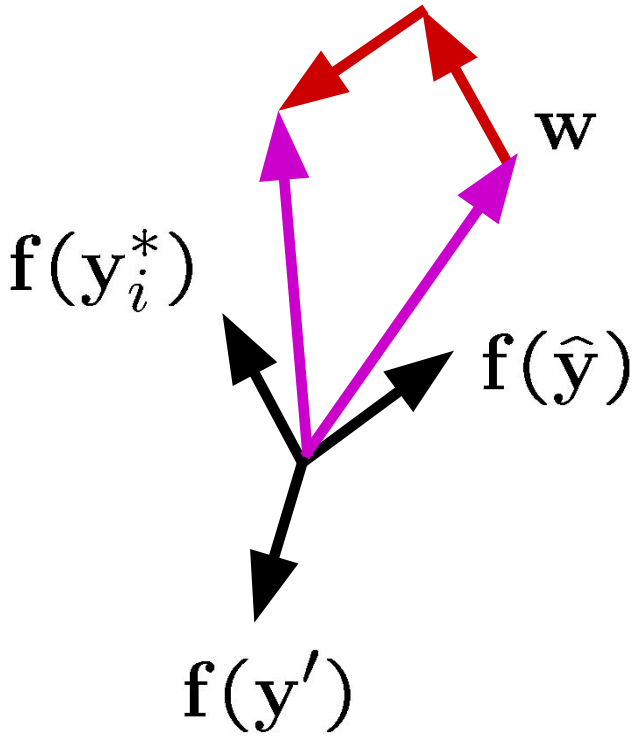
- The (online) perceptron algorithm:
  - Start with zero weights w
  - Visit training instances one by one
    - Try to classify

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

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

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

$$\mathbf{w} \leftarrow \mathbf{w} - \mathbf{f}(\hat{\mathbf{y}})$$



Linear Models: Maximum Entropy

- Maximum entropy (logistic regression)
  - Convert scores to probabilities:

$$P(\mathbf{y}|\mathbf{x},\mathbf{w}) = \frac{\exp(\mathbf{w}^\top \mathbf{f}(\mathbf{y}))}{\sum_{\mathbf{y}'} \exp(\mathbf{w}^\top \mathbf{f}(\mathbf{y}'))}$$

Make positive

Normalize

  - Maximize the (log) conditional likelihood of training data

$$L(\mathbf{w}) = \log \prod_i P(\mathbf{y}_i^*|\mathbf{x}_i,\mathbf{w}) = \sum_i \log \left(\frac{\exp(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}_i^*))}{\sum_{\mathbf{y}} \exp(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}))} \right)$$

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

Maximum Entropy II

- Regularization (smoothing)

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

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

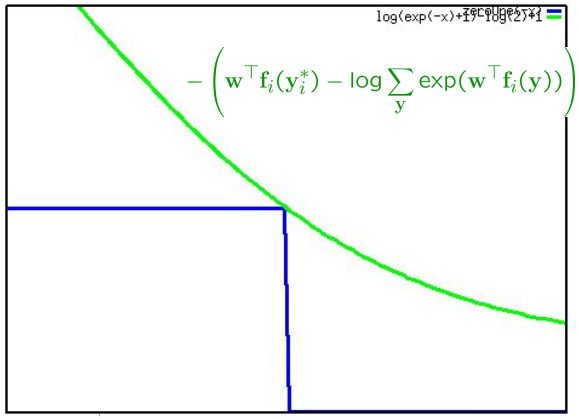
Log-Loss

- This minimizes the "log loss" on each example

$$-\left(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}_i^*) - \log \sum_{\mathbf{y}} \exp(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}))\right) = -\log \mathsf{P}(\mathbf{y}_i^*|\mathbf{x}_i, \mathbf{w})$$

$$step\left(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}_i^*) - \max_{\mathbf{y} \neq \mathbf{y}_i^*} \mathbf{w}^\top \mathbf{f}_i(\mathbf{y})\right)$$

- log loss is an *upper bound* on zero-one loss



How to update weights: Gradient Descent

We want to solve

$$\min_{x \in \mathbb{R}^n} f(x),$$

for f convex and differentiable

Gradient descent: choose initial $x^{(0)} \in \mathbb{R}^n$, repeat:

$$x^{(k)} = x^{(k-1)} - t_k \cdot \nabla f(x^{(k-1)}), \quad k = 1, 2, 3, \ldots$$

Gradient Descent: MaxEnt

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

- what do we need to compute the gradients?
  - Log normalizer
  - Expected feature counts

Maximum Margin

Linearly Separable

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

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

$$\min_{\mathbf{w}} \frac{1}{2}||\mathbf{w}||^2$$

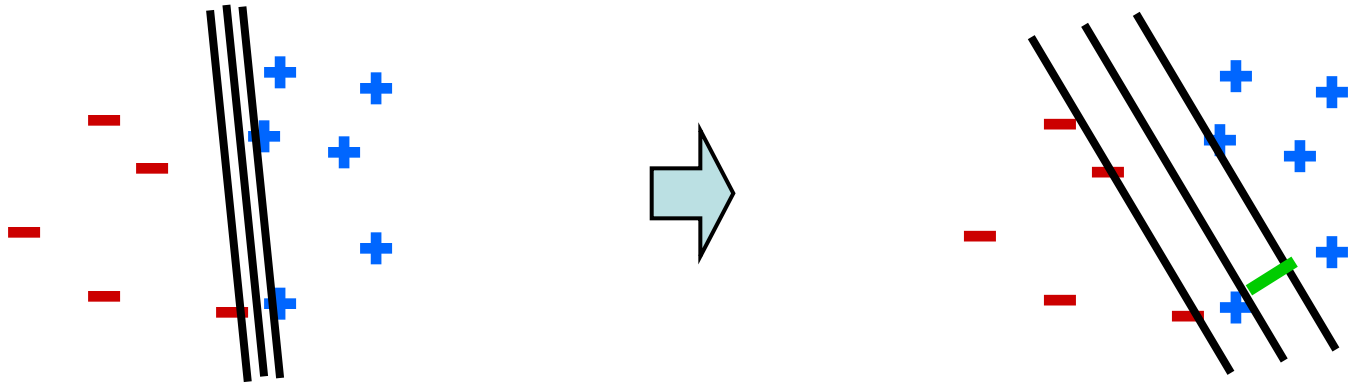
$$\forall i, \mathbf{y} \quad \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}_i^*) \geq \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}) + 1 \ell_i(\mathbf{y})$$

Maximum Margin

- Non-separable SVMs
  - Add slack to the constraints
  - Make objective pay (linearly) for slack:

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

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



Primal SVM

- We had a constrained minimization

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

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

- …but we can solve for ξ_i

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

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

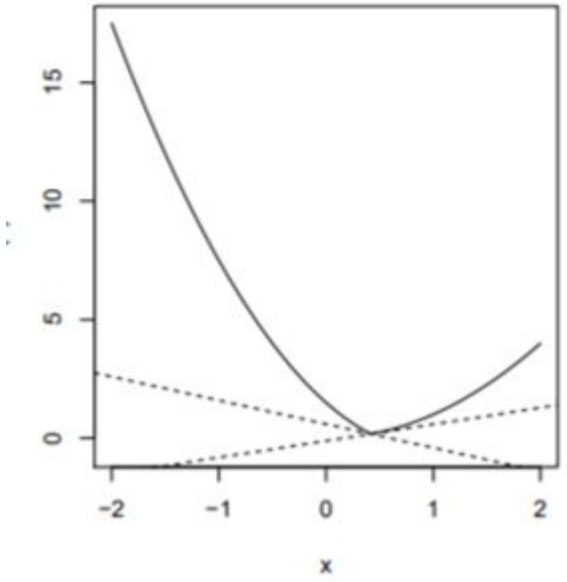
- Giving: the hinge loss

$$\min_{\mathbf{w}} \frac{1}{2}||\mathbf{w}||^2 + C\sum_i \left(\max_{\mathbf{y}} \left(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}) + \ell_i(\mathbf{y})\right) - \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}_i^*)\right)$$

How to update weights with hinge loss?

$$\min_{\mathbf{w}} \frac{1}{2}||\mathbf{w}||^2 + C\sum_i \left(\max_{\mathbf{y}} \left(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}) + \ell_i(\mathbf{y})\right) - \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}_i^*)\right)$$

- Not differentiable everywhere
- Use sub-gradients instead



Loss Functions: Comparison

- Zero-One Loss

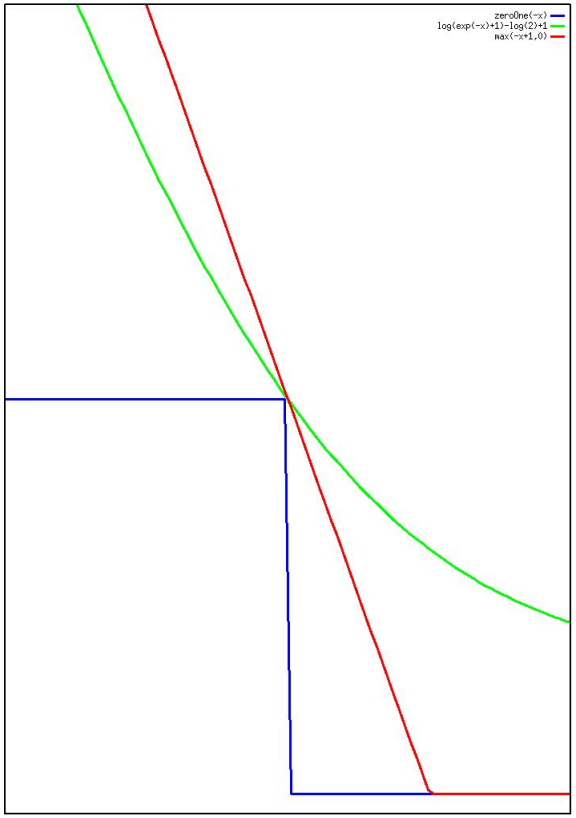
$$\sum_i step\left(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}_i^*) - \max_{\mathbf{y}\neq\mathbf{y}_i^*} \mathbf{w}^\top \mathbf{f}_i(\mathbf{y})\right)$$

- Hinge

$$\sum_i \left(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}_i^*) - \max_{\mathbf{y}} \left(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}) + \ell_i(y)\right)\right)$$

- Log

$$\sum_i \left(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}_i^*) - \log \sum_{\mathbf{y}} \exp\left(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y})\right)\right)$$



$$\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}_i^*) - \max_{\mathbf{y}\neq\mathbf{y}_i^*} \left(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y})\right)$$

Structured Margin

Just need efficient loss-augmented decode:

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

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

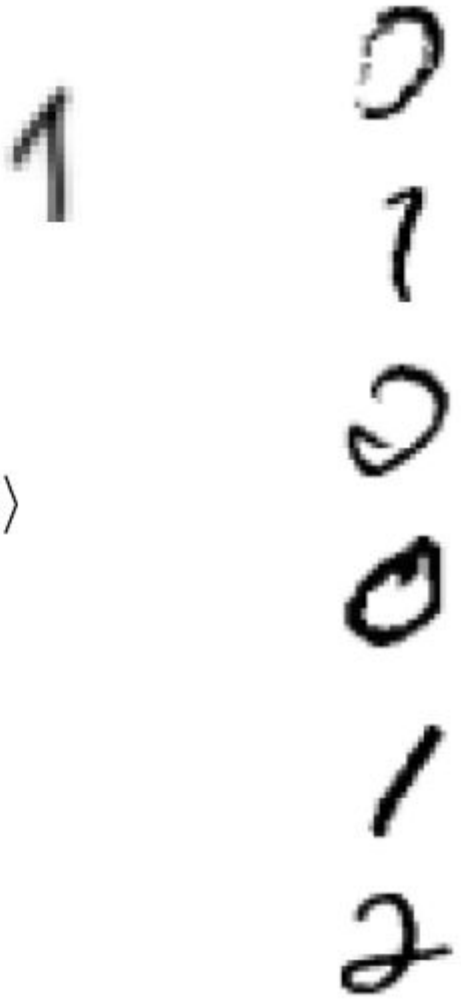
$$\nabla_w = w + C \sum_i \left(f_i(\bar{y}) - f_i(y_i^*) \right)$$

Still use general subgradient descent methods!

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:

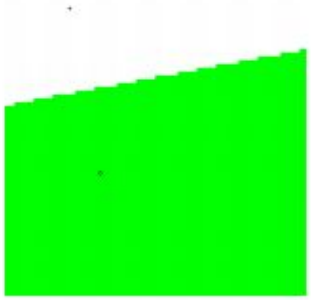
 $= \langle 0.0 \;\; 0.0 \;\; 0.3 \;\; 0.8 \;\; 0.7 \;\; 0.1 \ldots 0.0 \rangle$

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

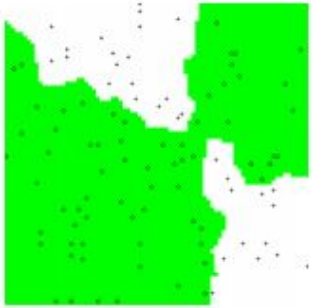
$$\text{sim}(x, y) = x^\top y = \sum_i x_i y_i$$

Non-Parametric Classification

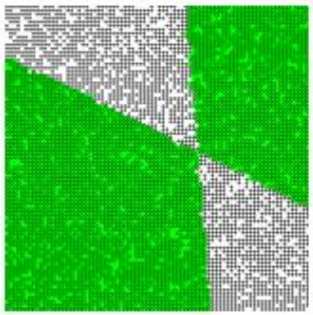
- Non-parametric: more examples means (potentially) more complex classifiers



- 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

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

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

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

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

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

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

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

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

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).

Example: Kernels

- Quadratic kernels

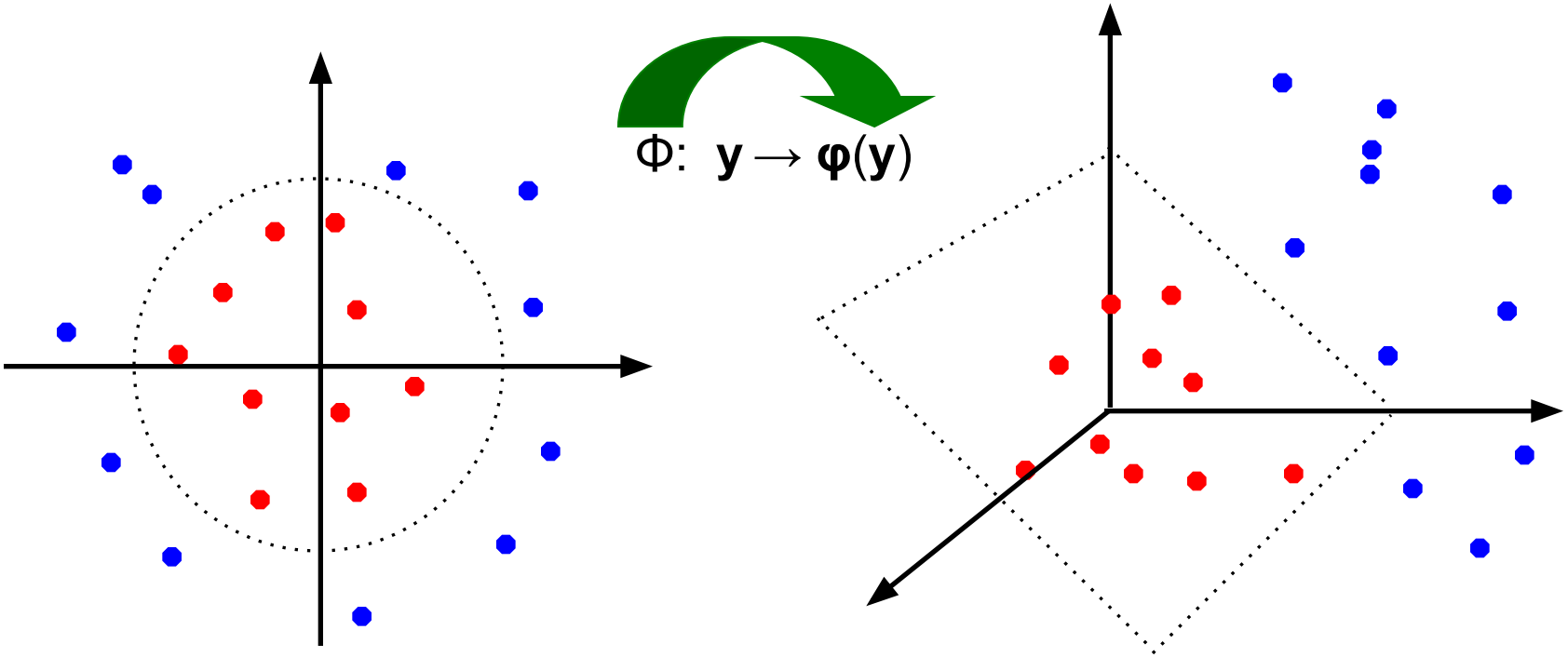
$$K(x, x') = (x \cdot x' + 1)^2$$

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

$$K(\mathbf{y}, \mathbf{y}') = (\mathbf{f}(\mathbf{y})^\top \mathbf{f}(\mathbf{y}') + 1)^2$$

Non-Linear Separators

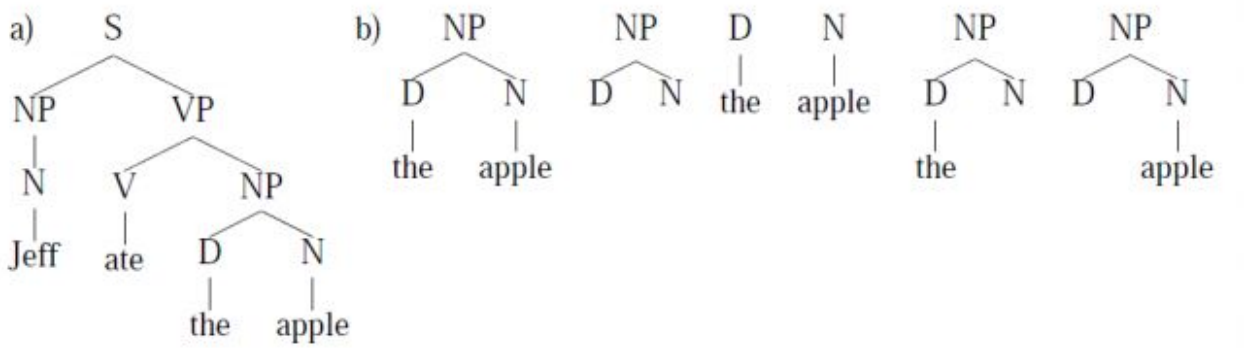
- Another view: kernels map an original feature space to some higher-dimensional feature space where the training set is (more) separable



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…



- 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 of SVM

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

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

- 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

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:

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

Back to Learning SVMs

- We want to find α which minimize

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

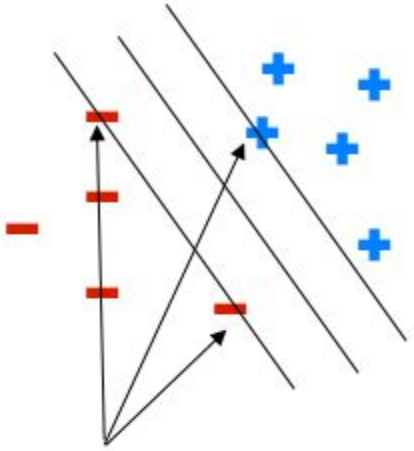
$$\forall i, \quad \sum_{\mathbf{y}} \alpha_i(\mathbf{y}) = C$$

What are these alphas?

- Each candidate corresponds to a primal constraint

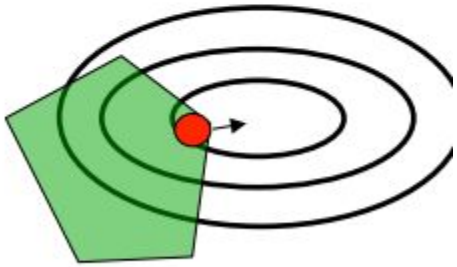
$$\min_{\mathbf{w},\xi} \quad \frac{1}{2}||\mathbf{w}||^2 + C\sum_i \xi_i$$

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



Support vectors

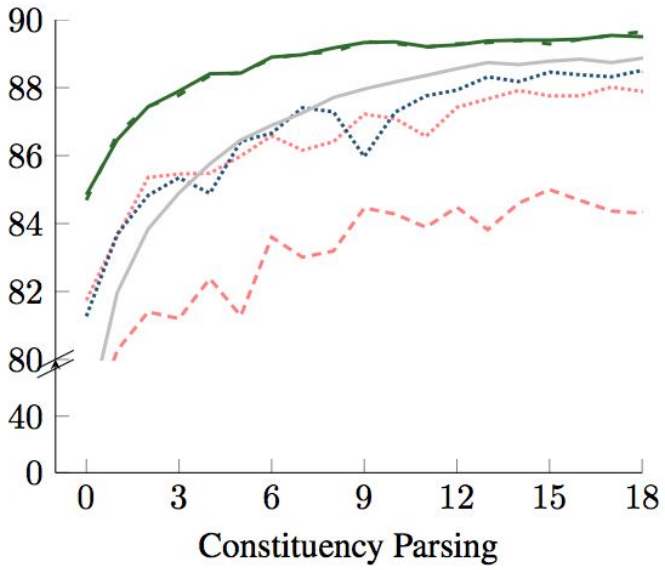
- In the solution, an $\alpha_i(\mathbf{y})$ will be:
  - Zero if that constraint is inactive
  - Positive if that constrain is active
  - i.e. positive on the support vectors



- Support vectors contribute to weights:

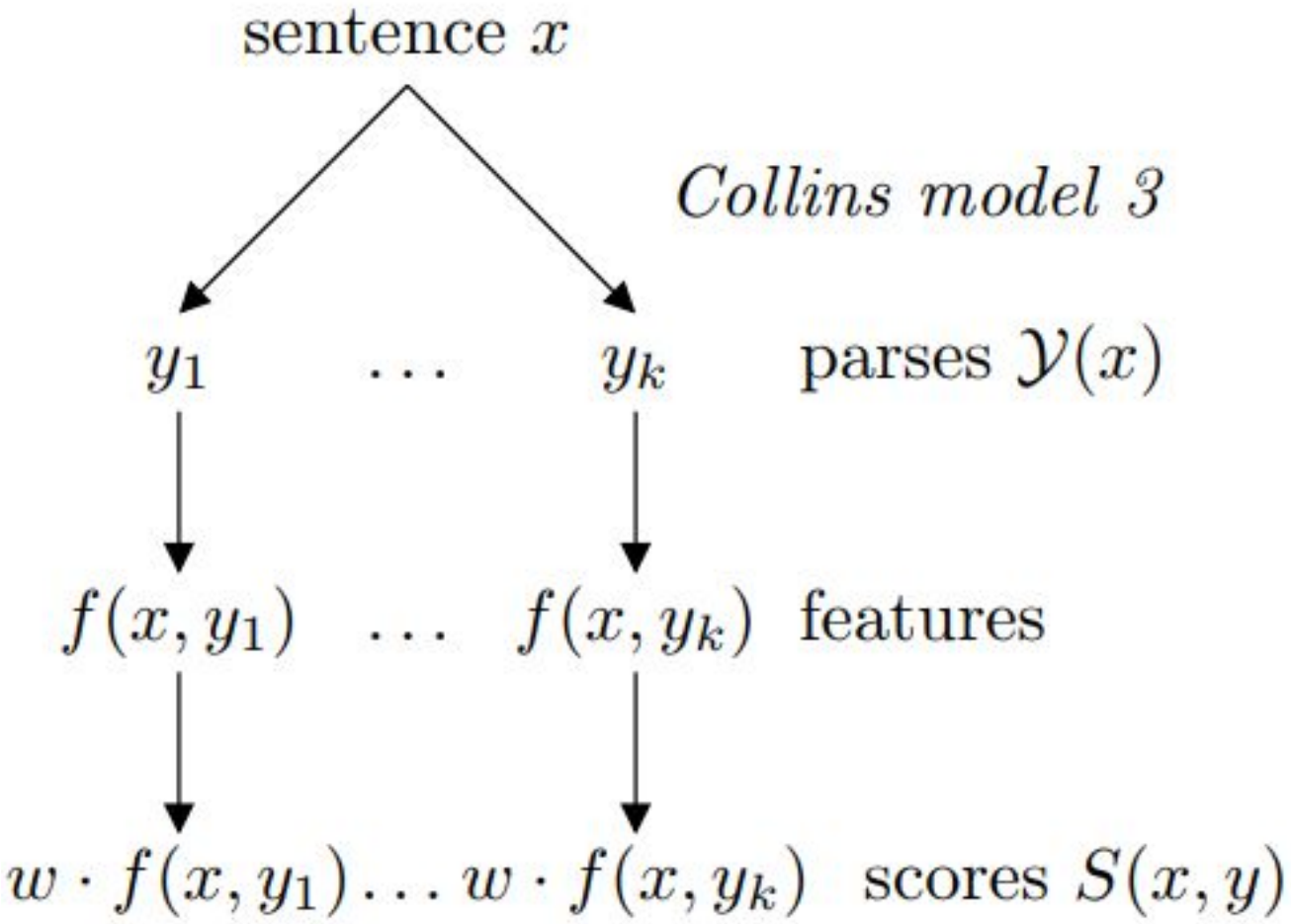
$$\mathbf{w} = \sum_{i,\mathbf{y}} \alpha_i(\mathbf{y}) \left(\mathbf{f}_i(\mathbf{y}_i^*) - \mathbf{f}_i(\mathbf{y})\right)$$

Comparison



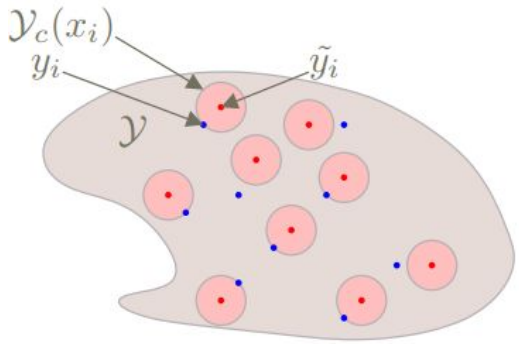
Margin	Cutting Plane
	Online Cutting Plane
	Online Primal Subgradient & L_1
	Online Primal Subgradient & L_2
Mistake Driven	Averaged Perceptron
	MIRA
	Averaged MIRA (MST built-in)
Llhood	Stochastic Gradient Descent

Reranking


sentence x
Collins model 3
y_1 ... y_k parses $\mathcal{Y}(x)$
$f(x, y_1)$ ... $f(x, y_k)$ features
$w \cdot f(x, y_1)$ ... $w \cdot f(x, y_k)$ scores $S(x, y)$

Training the reranker

- Training Data: $((x_1, y_1), \ldots, (x_n, y_n))$
- Generate candidate parses for each x



- Loss function:

$$\min_{\mathbf{w}} \frac{1}{2}||\mathbf{w}||^2 + C \sum_i \left(\max_{\mathbf{y}} \left(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}) + \ell_i(\mathbf{y}) \right) - \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}_i^*) \right)$$

Baseline and Oracle Results

- Training corpus: 36,112 Penn treebank trees, development corpus 3,720 trees from sections 2-21
- Collins Model 2 parser failed to produce a parse on 115 sentences
- Average $|\mathcal{Y}(x)| = 36.1$
- Model 2 f-score $= 0.882$ (picking parse with highest Model 2 probability)
- Oracle (maximum possible) f-score $= 0.953$ (i.e., evaluate f-score of closest parses $\tilde{y}_i$)

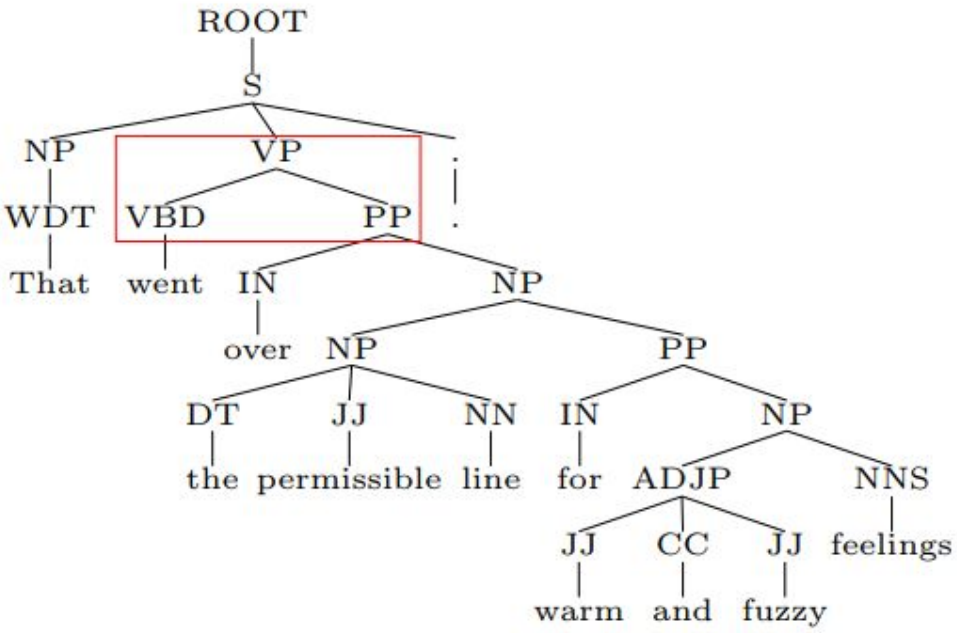
$\Rightarrow$ Oracle (maximum possible) error reduction 0.601

Collins Model 2

Experiment 1: Only "old" features

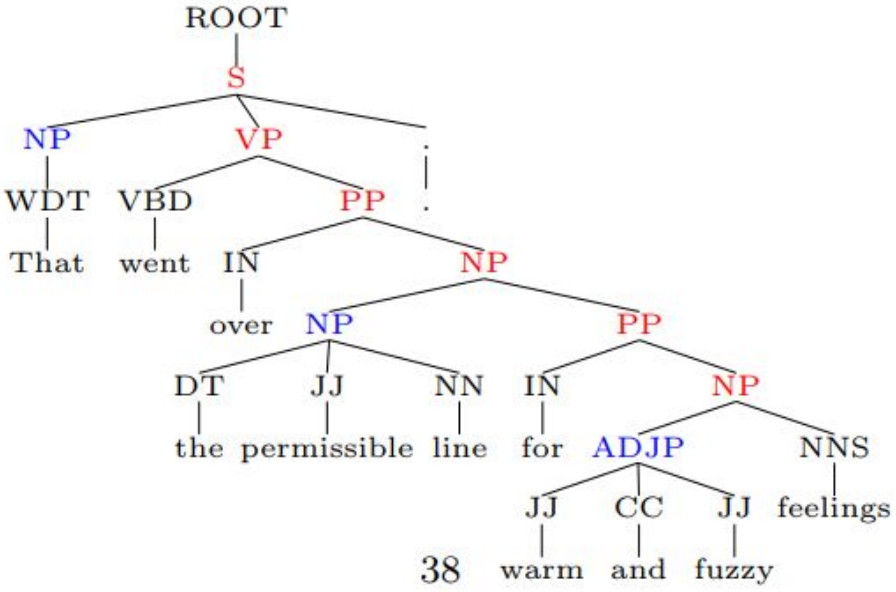
- Features: (1) *log Model 2 probability*, (9717) local tree features
- Model 2 already conditions on local trees!
- Feature selection: features must vary on 5 or more sentences
- Results: f-score $= 0.886$; $\approx 4\%$ error reduction

$\Rightarrow$ *discriminative training alone can improve accuracy*



Right Branching Bias

- The RightBranch feature's value is the number of nodes on the right-most branch (ignoring punctuation)
- Reflects the tendancy toward right branching
- LogProb + RightBranch: f-score $= 0.884$ (probably significant)
- LogProb + RightBranch + Rule: f-score $= 0.889$



Other Features

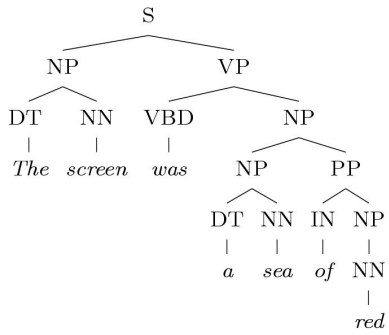
- Heaviness
  - What is the span of a rule
- Neighbors of a span
- Span shape
- Ngram Features
- Probability of the parse tree
- …

Results with all the features

- Features must vary on parses of at least 5 sentences in training data
- In this experiment, 692,708 features
- regularization term: $4 \sum_j |w_j|^2$
- dev set results: *f-score* $= 0.904$ (20% error reduction)

Reranking

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

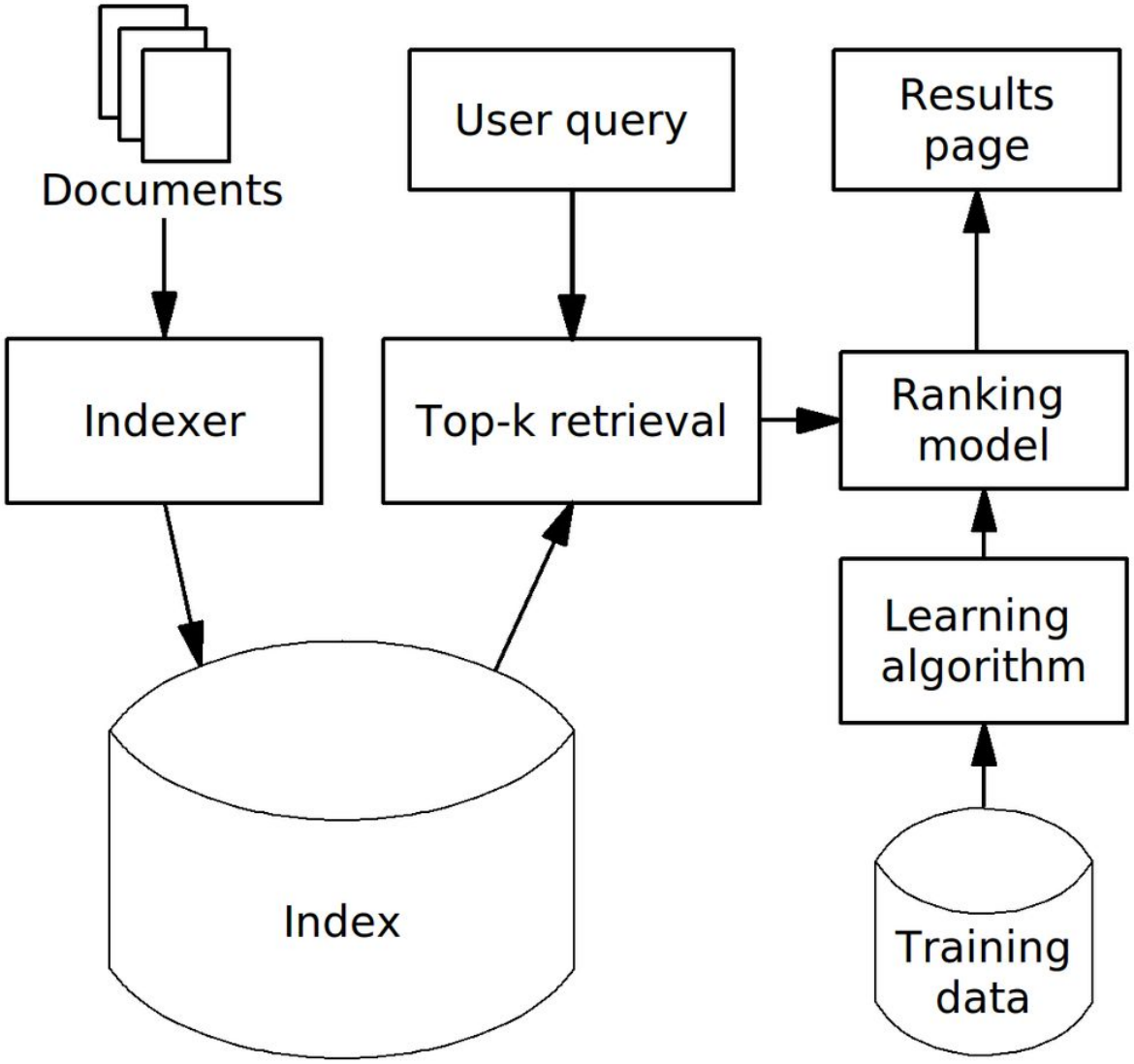
$\mathbf{f}($  $) =$

- Disadvantages:
  - Stuck with errors of baseline parser
  - Baseline system must produce n-best lists
  - But, feedback is possible [McCloskey, Charniak, Johnson 2006]
- But, a reranker (almost) never performs worse than a generative parser, and in practice performs substantially better.

Summary

- Generative parsing has many disadvantages
  - Independence assumptions
  - Difficult to express certain features without making grammar too large or parsing too complex
- Discriminative Parsing allows to add complex features while still being easy to train
- Candidate set for discriminative parsing is too large: Use reranking instead

Another Application of Reranking: Information Retrieval

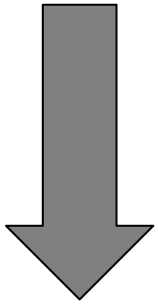


Modern Reranking Methods

Learn features using neural networks

$$\hat{y} = \arg\max_{y \in Y(X)} score(X, y)$$

$$\hat{y} = \arg\max_{y \in Y(X)} w^T f(x, y)$$



Replace by a neural network

Reranking for code generation

Input Utterance

x *download the file from url `url` and save it under file `file_name`*

System Predictions (n-best list of MRs)

z_1 `json.loads(['url', 'file_name', 'file_name'])`

$z \mapsto x : -34.7 \quad z \leftrightarrow x : -0.8$

z_2 `urllib.request.urlretrieve('url', 'r')`

$z \mapsto x : -35.2 \quad z \leftrightarrow x : -3.4$

z_3 `urllib.request.urlretrieve('url', 'file_name')`

$z \mapsto x : -31.4 \quad z \leftrightarrow x : -0.7$

$\vdots$

z_9 `r = urllib.request.urlretrieve(str_0)`

$z \mapsto x : -49.8 \quad z \leftrightarrow x : -5.8$

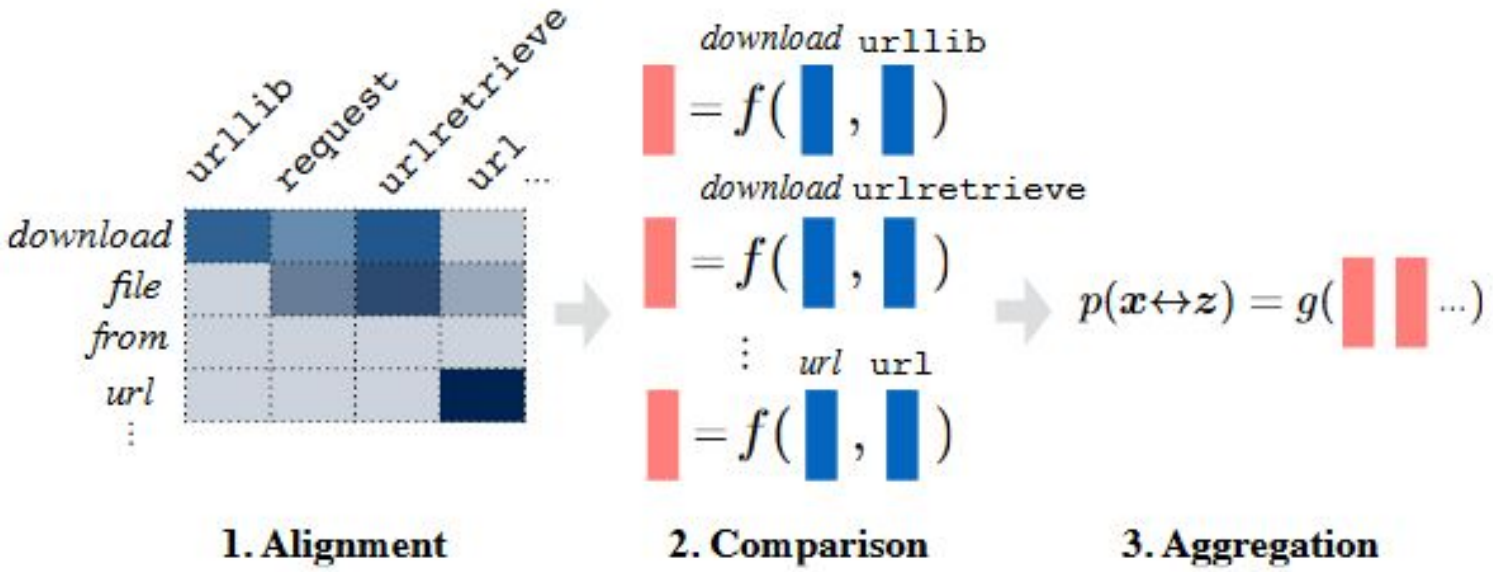
Reranker Output

z_3 `urllib.request.urlretrieve('url', 'file_name')`

Reranking for code generation (2)

$$R(\boldsymbol{z}, \boldsymbol{x}) = \alpha_0 \log p(\boldsymbol{z}|\boldsymbol{x}) + \sum_{k=1}^{K} \alpha_k f_k(\boldsymbol{z}, \boldsymbol{x})$$

- Matching features



Reranking for semantic parsing

Dataset	Example
GEO	x : *"which states adjoin alabama ?"* y : `answer(state(next_to_2(stateid(alabama))))`
ATIS	x : *"get flights between st. petersburg and charlotte"* y : `(_lambda $0 e (_and (_flight $0)` `(_from $0 st_petersburg:_ci) (_to $0 charlotte:_ci)))`
OVERNIGHT	x : *"show me all meetings not ending at 10 am"* y : `Type.Meeting ⊓ EndTime. != 10`