Algorithms for NLP

Classification II

Sachin Kumar - CMU

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

Minimize Training Error?

- A loss function declares how costly each mistake is

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

  - 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:

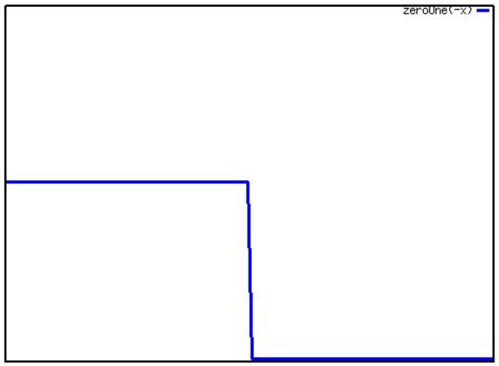
$$\min_{\mathbf{w}} \sum_i \ell_i \left(\arg\max_{\mathbf{y}} \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}) \right)$$

- This is a hard, discontinuous optimization problem

Objective Functions

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

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



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

Linear Models: Maximum Entropy

- Maximum entropy (logistic regression)
  - Use the scores as 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

- 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)

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

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

Log-Loss

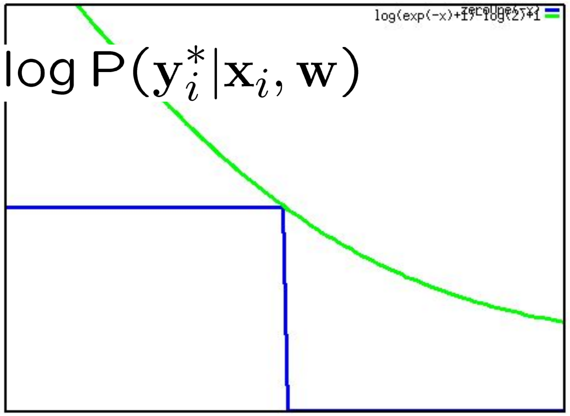
- If we view maxent as a minimization problem:

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

- This minimizes the "log loss" on each example

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

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



- One view: log loss is an *upper bound* on zero-one loss

Maximum Margin

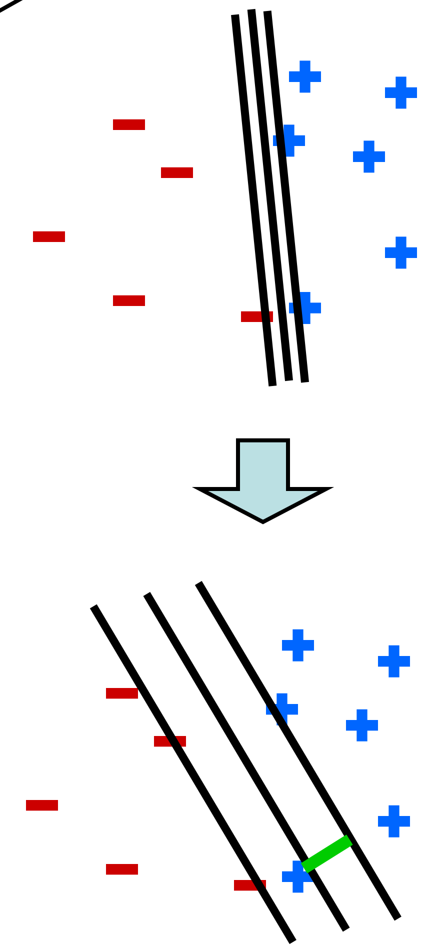
Note: exist other choices of how to penalize slacks!

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

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

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

  - 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



Remember SVMs…

- 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

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

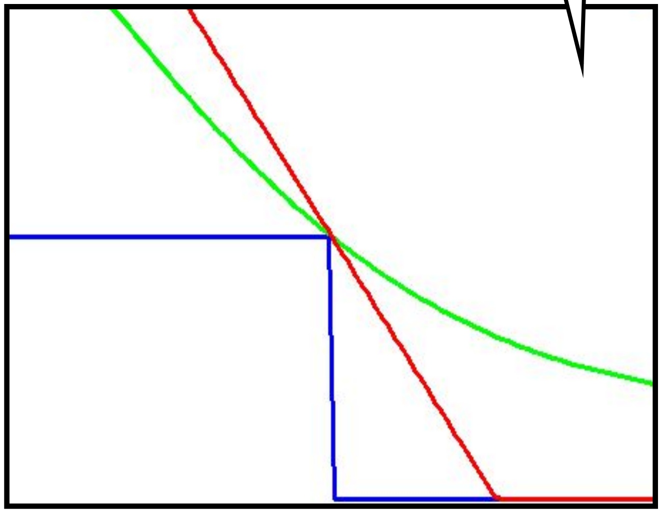
Hinge Loss

Plot really only right in binary case

- Consider the per-instance objective:

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

- 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)



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

Subgradient Descent

- Recall 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:

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

- Doesn't work for non-differentiable functions

Subgradient Descent

A **subgradient** of convex $f : \mathbb{R}^n \to \mathbb{R}$ at x is any $g \in \mathbb{R}^n$ such that

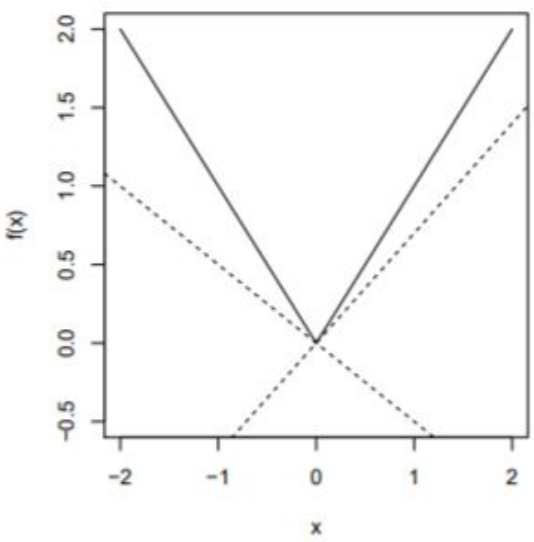
$$f(y) \geq f(x) + g^T(y - x), \quad \text{all } y$$

- Always exists
- If f differentiable at x, then $g = \nabla f(x)$ uniquely
- Actually, same definition works for nonconvex f (however, subgradient need not exist)

Subgradient Descent

- Example

Consider $f : \mathbb{R} \to \mathbb{R}$, $f(x) = |x|$

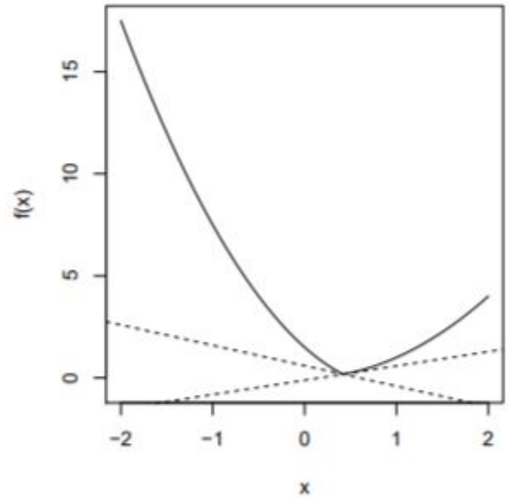


- For $x \neq 0$, unique subgradient $g = \mathrm{sign}(x)$
- For $x = 0$, subgradient g is any element of $[-1, 1]$

Subgradient Descent

- Example

Let $f_1, f_2 : \mathbb{R}^n \to \mathbb{R}$ be convex, differentiable, and consider $f(x) = \max\{f_1(x), f_2(x)\}$



- For $f_1(x) > f_2(x)$, unique subgradient $g = \nabla f_1(x)$
- For $f_2(x) > f_1(x)$, unique subgradient $g = \nabla f_2(x)$
- For $f_1(x) = f_2(x)$, subgradient g is any point on the line segment between $\nabla f_1(x)$ and $\nabla f_2(x)$

Subgradient Descent

Given convex $f : \mathbb{R}^n \to \mathbb{R}$, not necessarily differentiable

Subgradient method: just like gradient descent, but replacing gradients with subgradients. I.e., initialize $x^{(0)}$, then repeat

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

where $g^{(k-1)}$ is any subgradient of f at $x^{(k-1)}$

Subgradient method is not necessarily a descent method, so we keep track of best iterate $x_{\text{best}}^{(k)}$ among $x^{(1)}, \ldots x^{(k)}$ so far, i.e.,

$$f(x_{\text{best}}^{(k)}) = \min_{i=1,\ldots k} f(x^{(i)})$$

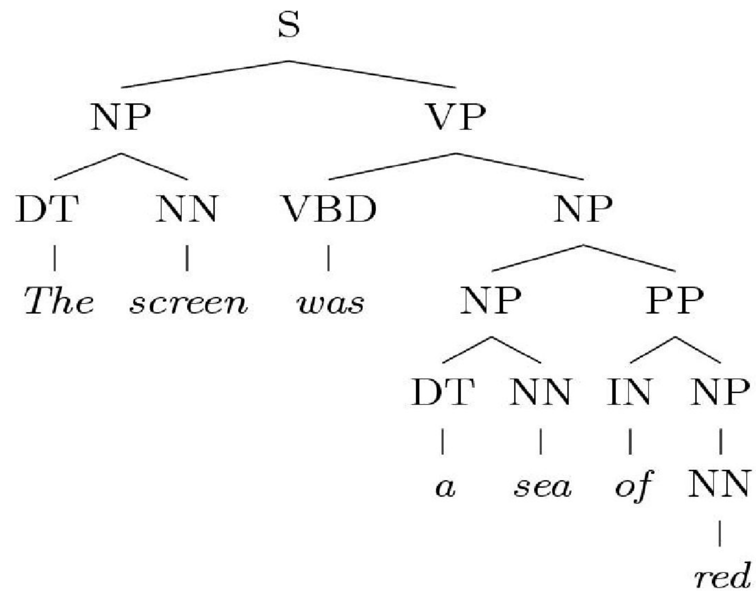
Structure

CFG Parsing

x

The screen was a sea of red

y



Recursive structure

Generative vs Discriminative

- Generative Models have many advantages
  - Can model both p(x) and p(y|x)
  - Learning is often clean and analytical: frequency estimation in penn treebank
- Disadvantages?
  - Force us to make rigid independence assumptions (context free assumption)

Generative vs **Discriminative**

- We get more freedom in defining features - no independence assumptions required
- Disadvantages?
  - Computationally intensive
  - Use of more features can make decoding harder

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

Efficient 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

- 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

Max-Ent, Structured, Global

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

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

- Assumption: Score is sum of local "part" scores

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

Max-Ent, Structured, Global

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

- what do we need to compute the gradients?
  - Log normalizer
  - Expected feature counts (inside outside algorithm)
- How to decode?
  - Search algorithms like viterbi (CKY)

Max-Ent, Structured, Local

- We assume that we can arrive at a globally optimal solution by making locally optimal choices.
- We can use arbitrarily complex features over the history and lookahead over the future.
- We can perform very efficient parsing, often with linear time complexity
- Shift-Reduce parsers

Structured Margin (Primal)

Remember our primal margin objective?

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

Still applies with structured output space!

Structured Margin (Primal)

Just need efficient loss-augmented decode:

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

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

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

Still use general subgradient descent methods! (Adagrad)

Structured Margin

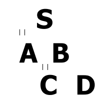
- Remember the constrained version of primal:

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

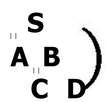
$$\forall i, \mathbf{y} \quad \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}_i^*) \geq \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}) + \ell_i(\mathbf{y}) - \xi_i$$

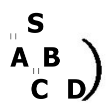
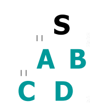
Many Constraints!

- We want:

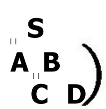
$\arg\max_{y}\ w^\top f(\text{'It was red'}, y) =$ 

- Equivalently:

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

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

...

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

a lot!

Structured Margin - Working Set

- It's enough if we enforce the **active constraints**.
  The others will be fulfilled automatically.
- We don't know which ones are active for the optimal solution.
- But it's likely to be only a small number ← can of course be formalized.

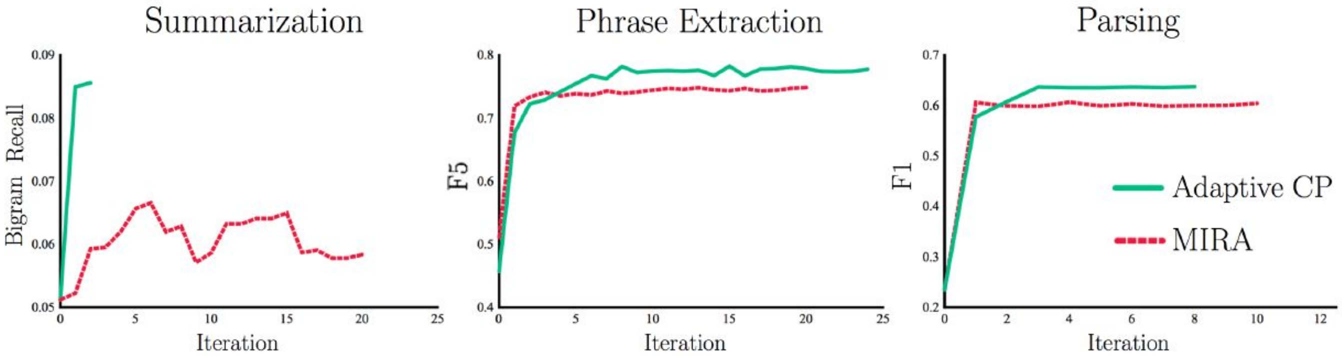
Keep a set of potentially active constraints and update it iteratively:

- Start with working set $S = \emptyset$ (no contraints)
- Repeat until convergence:
  - Solve S-SVM training problem with constraints from S
  - Check, if solution violates any of the *full* constraint set
    - if no: we found the optimal solution, *terminate*.
    - if yes: add most violated constraints to S, *iterate*.

[Tsochantaridis et al. "Large Margin Methods for Structured and Interdependent Output Variables", JMLR, 2005.]

Working Set S-SVM

- Working Set n-slack Algorithm
- Working Set 1-slack Algorithm
- Cutting Plane 1-Slack Algorithm [Joachims et al 09]
    - Requires Dual Formulation
    - Much faster convergence
    - In practice, works as fast as perceptron, more stable training



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:

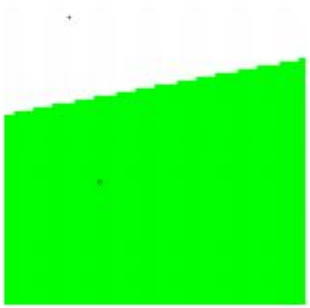
$$= \langle 0.0 \;\; 0.0 \;\; 0.3 \;\; 0.8 \;\; 0.7 \;\; 0.1 \ldots 0.0 \rangle$$

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

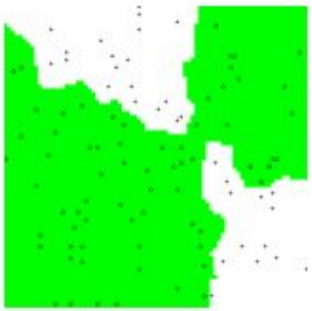
$$\text{sim}(x, y) = x^\top y = \sum_i x_i y_i$$

Non-Parametric Classification

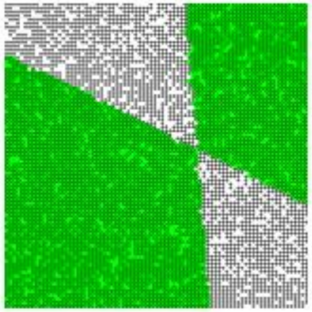
- Non-parametric: more examples means (potentially) more complex classifiers



- 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

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

  - 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

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

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

  - 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

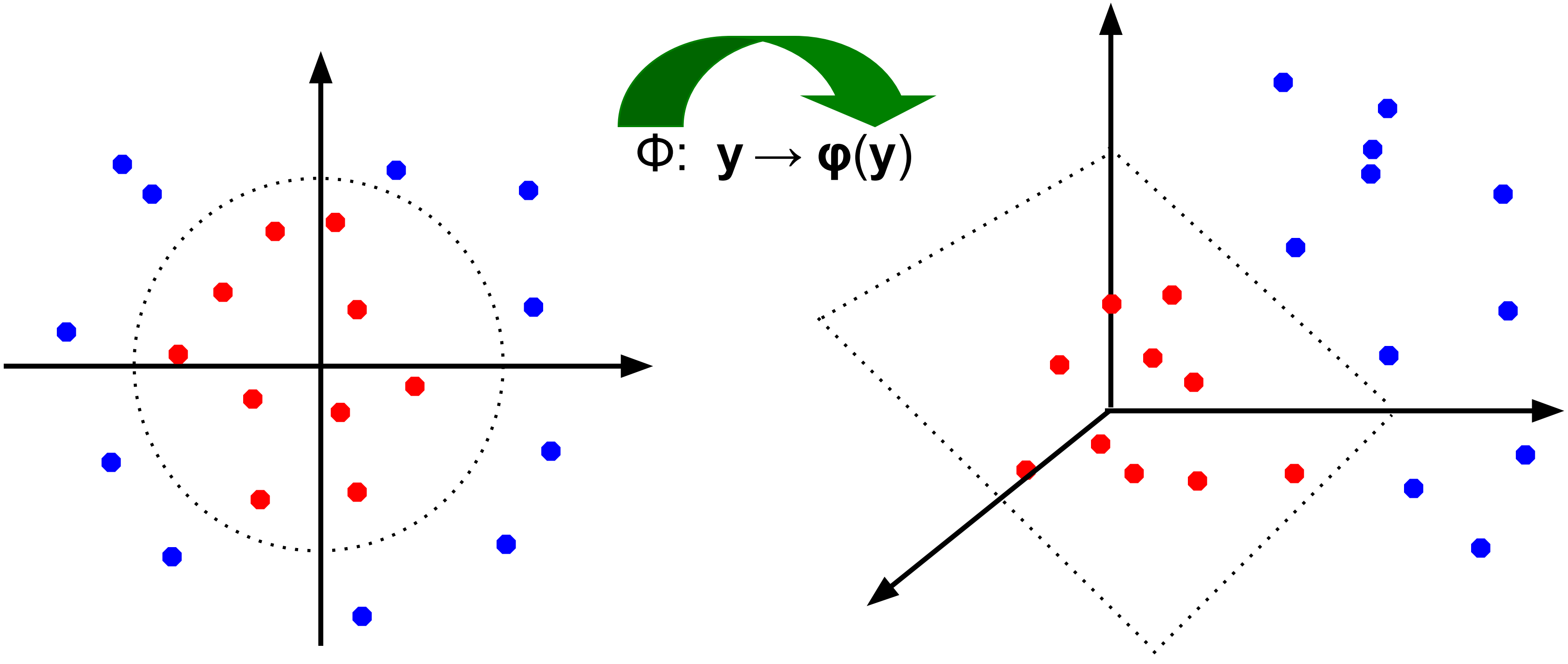
$$K(x, x') = (x \cdot x' + 1)^2$$

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

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

Non-Linear Separators

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

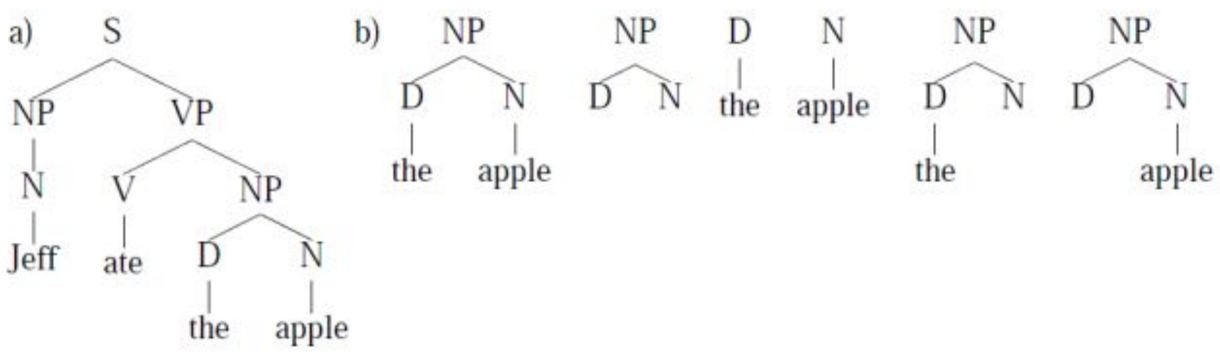


Why Kernels?

- 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…

Tree Kernels



- 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)

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

- 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

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

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

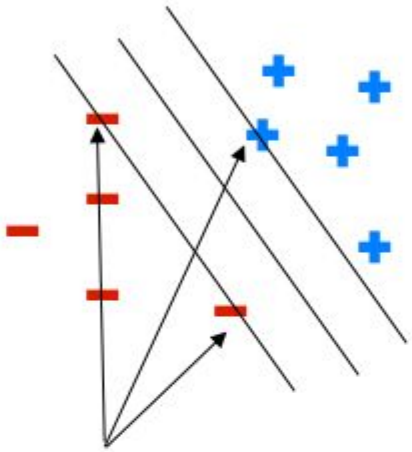
$$\forall i, \quad \sum_{\mathbf{y}} \alpha_i(\mathbf{y}) = C$$

What are these alphas?

- Each candidate corresponds to a primal constraint

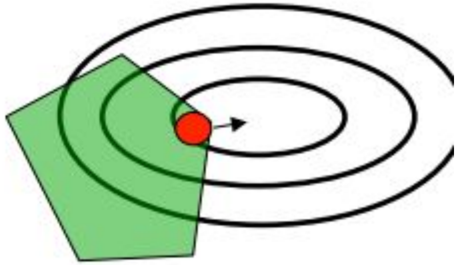
$$\min_{\mathbf{w},\xi} \quad \frac{1}{2}||\mathbf{w}||^2 + C\sum_i \xi_i$$

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



Support vectors

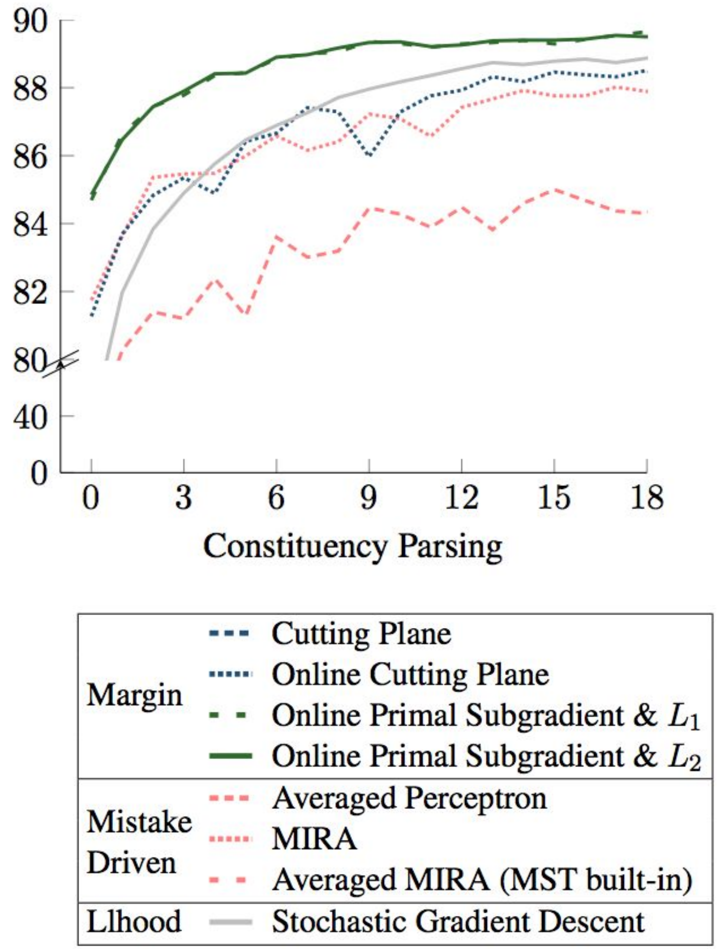
- In the solution, an $\alpha_i(\mathrm{y})$ will be:
  - Zero if that constraint is inactive
  - Positive if that constrain is active
  - i.e. positive on the support vectors



- Support vectors contribute to weights:

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

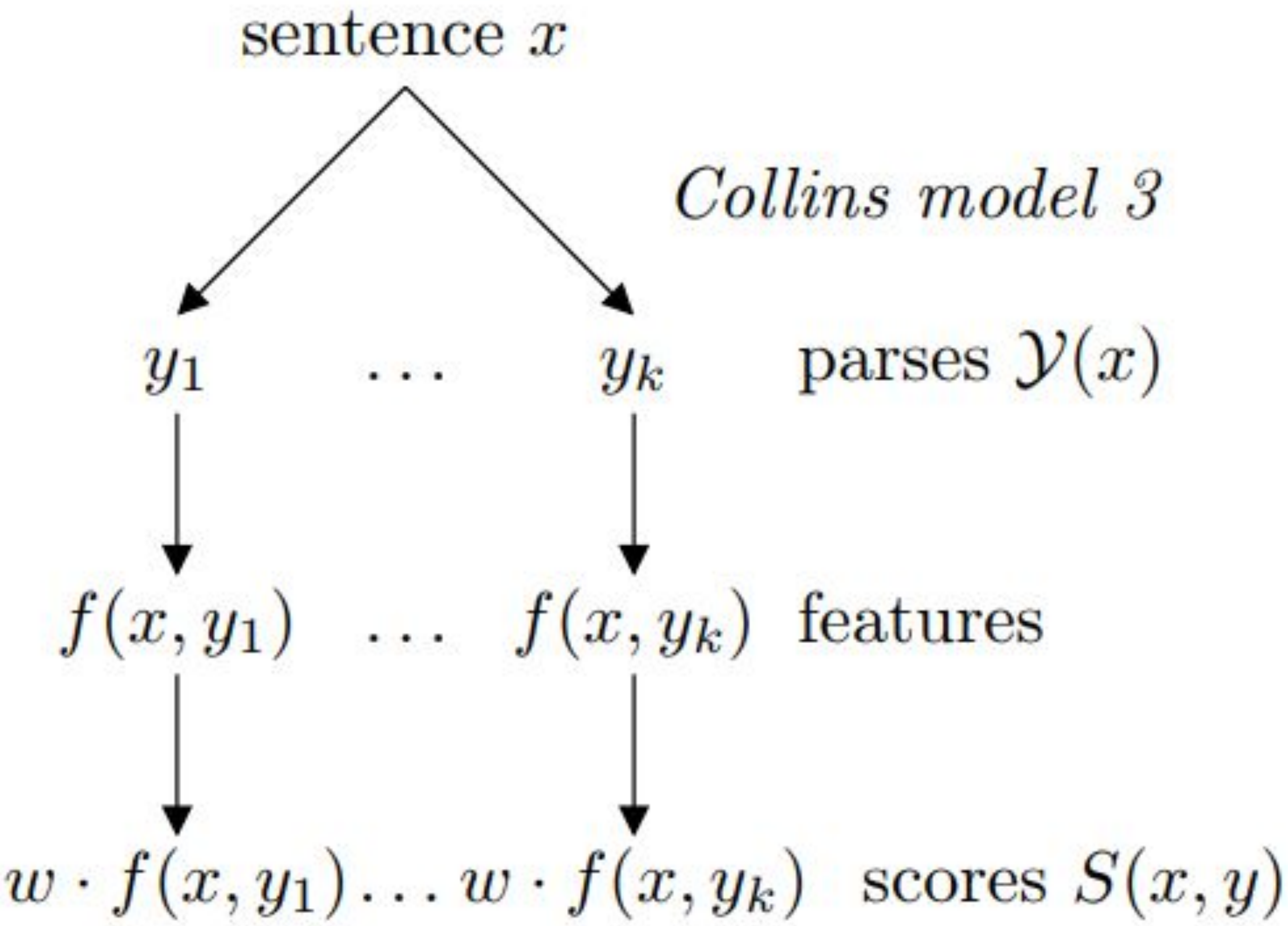
Comparison


90
88
86
84
82
80
40
0
0
3
6
9
12
15
18
Constituency Parsing
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

To summarize

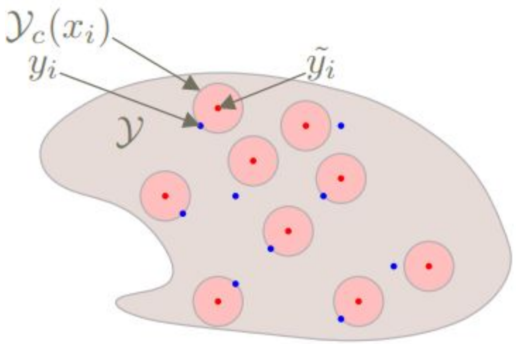
- Can solve Structural versions of Max-Ent and SVMs
  - our feature model factors into reasonably local, non-overlapping structures (why?)
- Issues?
  - Limited Scope of Features

Reranking


sentence x
Collins model 3
$y_1 \quad \ldots \quad y_k$
parses $\mathcal{Y}(x)$
$f(x, y_1) \quad \ldots \quad f(x, y_k)$
features
$w \cdot f(x, y_1) \ldots w \cdot f(x, y_k)$
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 $\tilde{y}_i$)

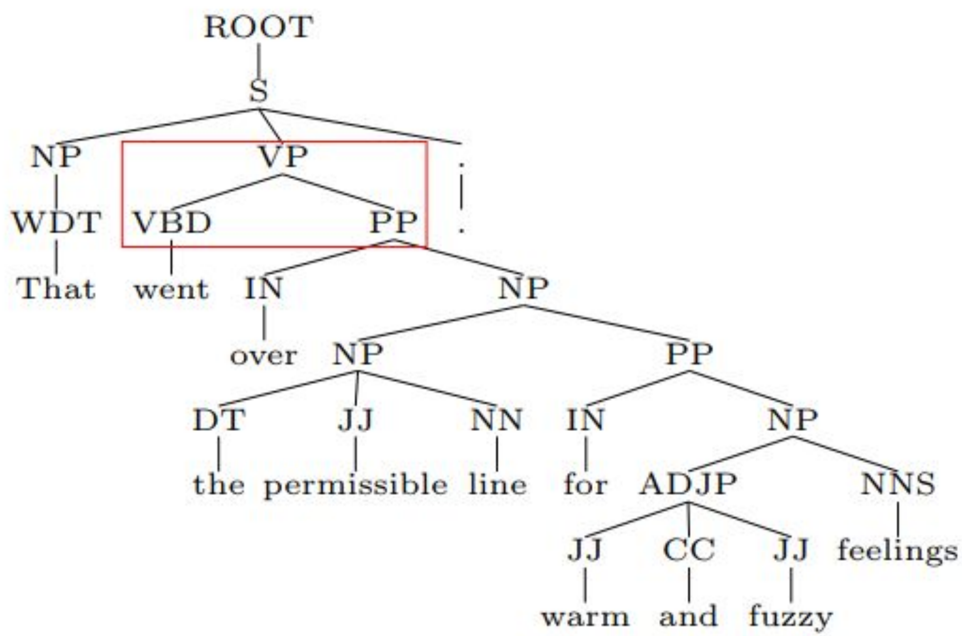
$\Rightarrow$ Oracle (maximum possible) error reduction 0.601

Collins Model 2

Experiment 1: Only "old" features

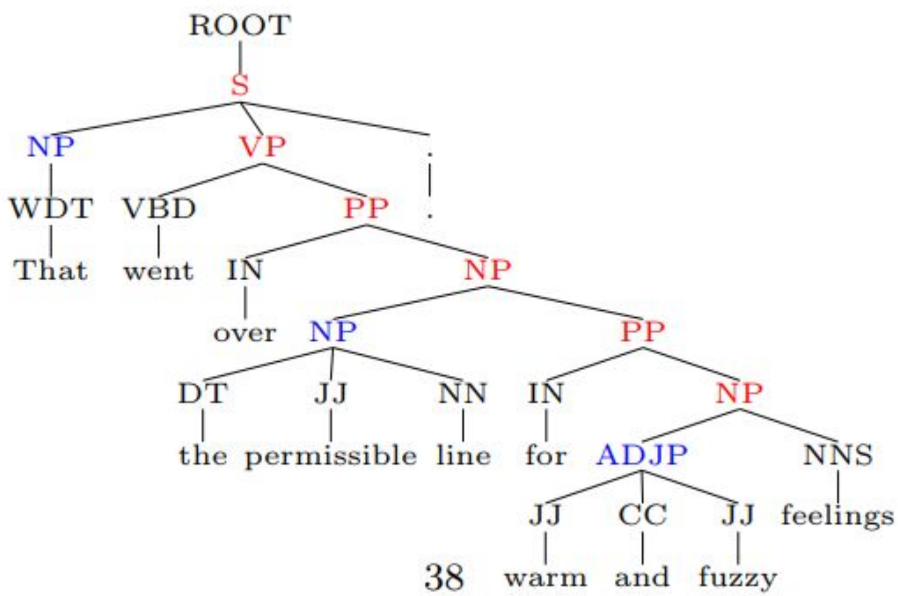
- Features: (1) *log Model 2 probability*, (9717) local tree features
- Model 2 already conditions on local trees!
- Feature selection: features must vary on 5 or more sentences
- Results: f-score $= 0.886$; $\approx 4\%$ error reduction

$\Rightarrow$ *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

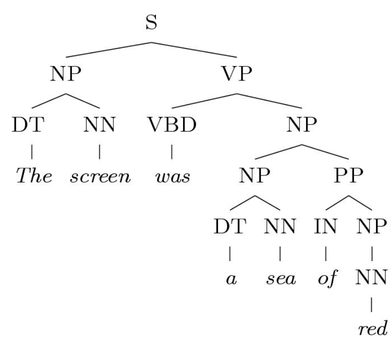
- Heaviness
  - What is the span of a rule
- Neighbors of a span
- Span shape
- Ngram Features
- Probability of the parse tree
- …

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.

Reranking in other settings

- Speech recognition
  - Take x to be the acoustic signal, $\mathcal{Y}(x)$ all strings in recognizer lattice for x
  - Training data: $D = ((y_1, x_1), \ldots, (y_n, x_n))$, where y_i is correct transcript for x_i
  - Features could be n-grams, log parser prob, cache features
- Machine translation
  - Take x to be input language string, $\mathcal{Y}(x)$ a set of target language strings (e.g., generated by an IBM-style model)
  - Training data: $D = ((y_1, x_1), \ldots, (y_n, x_n))$, where y_i is correct translation of x_i
  - Features could be n-grams of target language strings, word and phrase correspondences, ...