Statistical NLP
Spring 2011

Lecture 11: Classification

Dan Klein – UC Berkeley

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 / kernel functions
  - Scoring by linear functions
  - Learning by optimization

Example: Text Classification

- We want to classify documents into semantic categories

DOCUMENT	CATEGORY
… win the election …	*POLITICS*
… win the game …	*SPORTS*
… see a movie …	*OTHER*

- Classically, do this on the basis of counts of words in the document, but other information sources are relevant:
  - Document length
  - Document's source
  - Document layout
  - Document sender
  - …

Some Definitions

INPUTS	$\mathbf{x}_i$	*… win the election …*
CANDIDATE SET	$\mathcal{Y}(\mathbf{x})$	[*… win the election …* *SPORTS*, *… win the election …* *POLITICS*, *… win the election …* *OTHER*]
CANDIDATES	$\mathbf{y}$	[*… win the election …* *SPORTS*]
TRUE OUTPUTS	$\mathbf{y}_i^*$	[*… win the election …* *POLITICS*]
FEATURE VECTORS	$\mathbf{f}_i(\mathbf{y})$	[0 0 0 0 1 0 1 0 0 0 0 0]

SPORTS ∧ "*win*"

POLITICS ∧ "*win*"

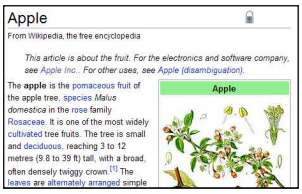
POLITICS ∧ "*election*"

Remember: if y contains x, we also write f(y)

Feature Vectors

- Example: web page ranking (not actually classification)

x_i = "Apple Computers"

$\mathbf{f}_i($ 

$) = [0.3\ 5\ 0\ 0\ \ldots]$

$\mathbf{f}_i($ 

$) = [0.8\ 4\ 2\ 1\ \ldots]$

Block Feature Vectors

- Sometimes, we think of the input as having features, which are multiplied by outputs to form the candidates

$\mathbf{x}$ … *win the election* …

"$\mathbf{f}(\mathbf{x})$" $[1\ 0\ 1\ 0]$

"*win*" "*election*"

$\mathbf{f}($ *… win the election …* $SPORTS) = [1\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$

$\mathbf{f}($ *… win the election …* $POLITICS) = [0\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 0]$

$\mathbf{f}($ *… win the election …* $OTHER) = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0]$

Linear Models: Scoring

- In a linear model, each feature gets a weight w

$$\mathbf{f}(\overset{\text{... win the election ...}}{POLITICS}) = [\ 0\ \ 0\ \ 0\ \ 0\ \ 1\ \ 0\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0]$$
$$\mathbf{f}(\overset{\text{... win the election ...}}{SPORTS}) = [\ 1\ \ 0\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0]$$
$$\mathbf{w} = [\ 1\ \ 1\ \ -1\ \ -2\ \ 1\ \ -1\ \ 1\ \ -2\ \ -2\ \ -1\ \ -1\ \ 1]$$

- We score hypotheses by multiplying features and weights:

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

$$\mathbf{f}(\overset{\text{... win the election ...}}{POLITICS}) = [\ 0\ \ 0\ \ 0\ \ 0\ \ 1\ \ 0\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0]$$
$$\mathbf{w} = [\ 1\ \ 1\ \ -1\ \ -2\ \ 1\ \ -1\ \ 1\ \ -2\ \ -2\ \ -1\ \ -1\ \ 1]$$

$$score(\overset{\text{... win the election ...}}{POLITICS}, \mathbf{w}) = 1 \times 1 + 1 \times 1 = 2$$

Linear Models: Decision Rule

- The linear decision rule:

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

$$score(\overset{\text{... win the election ...}}{SPORTS}, \mathbf{w}) = 1 \times 1 + (-1) \times 1 = 0$$
$$score(\overset{\text{... win the election ...}}{POLITICS}, \mathbf{w}) = 1 \times 1 + 1 \times 1 = 2$$
$$score(\overset{\text{... win the election ...}}{OTHER}, \mathbf{w}) = (-2) \times 1 + (-1) \times 1 = -3$$

$$prediction(\overset{\text{... win the election ...}}{}, \mathbf{w}) = \overset{\text{... win the election ...}}{POLITICS}$$

- We've said nothing about where weights come from!

Binary Classification

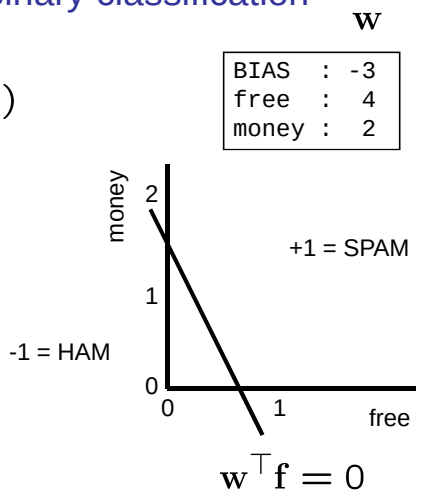
- Important special case: binary classification
  - Classes are y=+1/-1

$$\mathbf{f}(\mathbf{x}, -1) = -\mathbf{f}(\mathbf{x}, +1)$$

$$\mathbf{f}(\mathbf{x}) = 2\mathbf{f}(\mathbf{x}, +1)$$

  - Decision boundary is a hyperplane

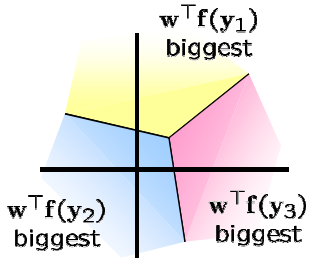
$$\mathbf{w}^\top \mathbf{f}(\mathbf{x}) = 0$$



10

Multiclass Decision Rule

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



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

- There are other ways: e.g. reconcile pairwise decisions

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

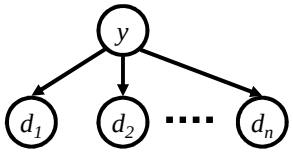
Linear Models: Naïve-Bayes

- (Multinomial) Naïve-Bayes is a linear model, where:

$$\mathbf{x}^i = d_1, d_2, \cdots d_n$$

$$\begin{array}{rcl} \mathbf{f}_i(\mathbf{y}) & = & [\ \cdots 0 \cdots \quad 1, \quad \#v_1, \quad \#v_2, \quad \cdots \quad \#v_{|V|} \quad \cdots 0 \cdots\ ] \\ \mathbf{w} & = & [\ \cdots \quad \log \mathrm{P}(y), \ \log \mathrm{P}(v_1|y), \ \log \mathrm{P}(v_2|y), \ \cdots \ \log \mathrm{P}(v_n|y) \quad \cdots\ ] \end{array}$$

$$\begin{array}{rcl} \mathrm{score}(\mathbf{x}_i, \mathbf{y}, \mathbf{w}) & = & \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}) \\ & = & \log \mathrm{P}(\mathbf{y}) + \sum_k \#v_k \log \mathrm{P}(v_k|\mathbf{y}) \\ & = & \log \left(\mathrm{P}(\mathbf{y}) \prod_k \mathrm{P}(v_k|\mathbf{y})^{\#v_k} \right) \\ & = & \log \left(\mathrm{P}(\mathbf{y}) \prod_{d \in \mathbf{x}^i} \mathrm{P}(d|\mathbf{y}) \right) \\ & = & \log \mathrm{P}(\mathbf{x}^i, \mathbf{y}) \end{array}$$



Example: Sensors

Reality

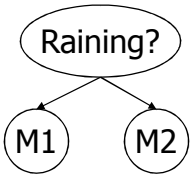
Raining

P(+,+,r) = 3/8 P(-,-,r) = 1/8

Sunny

P(+,+,s) = 1/8 P(-,-,s) = 3/8

NB Model



NB FACTORS:

- P(s) = 1/2
- P(+|s) = 1/4
- P(+|r) = 3/4

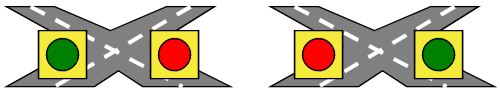
PREDICTIONS:

- P(r,+,+) = (½)(¾)(¾)
- P(s,+,+) = (½)(¼)(¼)
- P(r|+,+) = 9/10
- P(s|+,+) = 1/10

Example: Stoplights

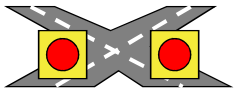
Reality

Lights Working



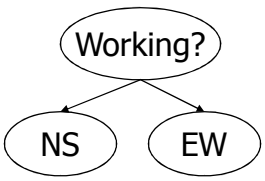
P(g,r,w) = 3/7 P(r,g,w) = 3/7

Lights Broken



P(r,r,b) = 1/7

NB Model



NB FACTORS:

- P(w) = 6/7
- P(r|w) = 1/2
- P(g|w) = 1/2
- P(b) = 1/7
- P(r|b) = 1
- P(g|b) = 0

Example: Stoplights

- What does the model say when both lights are red?
  - P(b,r,r) = (1/7)(1)(1) = 1/7 = 4/28
  - P(w,r,r) = (6/7)(1/2)(1/2) = 6/28 = 6/28
  - P(w|r,r) = 6/10!
- We'll guess that (r,r) indicates lights are working!
- Imagine if P(b) were boosted higher, to 1/2:
  - P(b,r,r) = (1/2)(1)(1) = 1/2 = 4/8
  - P(w,r,r) = (1/2)(1/2)(1/2) = 1/8 = 1/8
  - P(w|r,r) = 1/5!
- Changing the parameters bought accuracy at the expense of data likelihood

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

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

  - 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

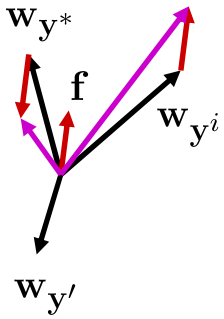
Linear Models: Perceptron

- 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

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



Example: “Best” Web Page

$\mathbf{w} = [1 \quad 2 \quad 0 \quad 0 \quad \ldots]$

x_i = “Apple Computers”

$\mathbf{f}_i($  $) = [0.3\ 5\ 0\ 0\ \ldots]$ $\quad \mathbf{w}^\top \mathbf{f} = 10.3 \quad \hat{\mathbf{y}}$

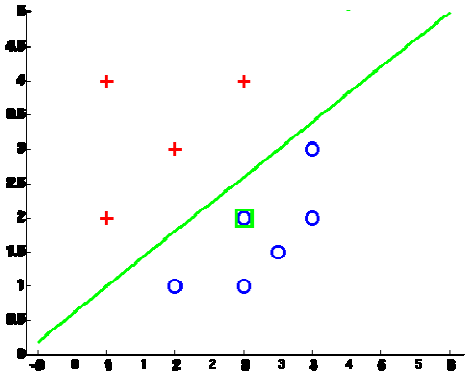
$\mathbf{f}_i($  $) = [0.8\ 4\ 2\ 1\ \ldots]$ $\quad \mathbf{w}^\top \mathbf{f} = 8.8 \quad \mathbf{y}_i^*$

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

$\mathbf{w} = [1.5 \quad 1 \quad 2 \quad 1 \quad \ldots]$

Examples: Perceptron

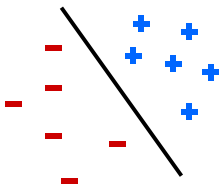
- Separable Case



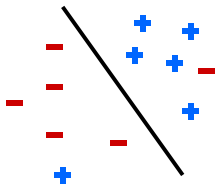
Perceptrons and Separability

- A data set is separable if some parameters classify it perfectly
- Convergence: if training data separable, perceptron will separate (binary case)
- Mistake Bound: the maximum number of mistakes (binary case) related to the *margin* or degree of separability

Separable

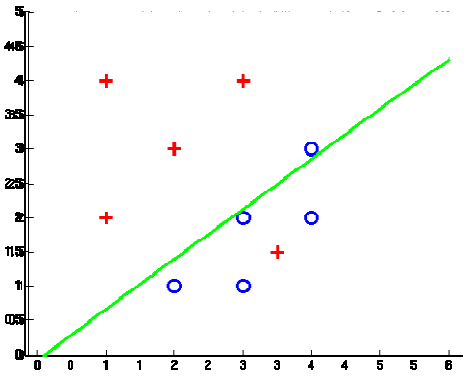


Non-Separable



Examples: Perceptron

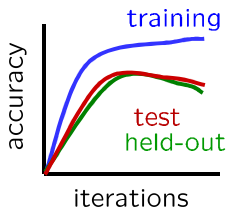
- Non-Separable Case



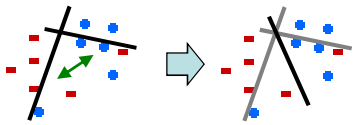
23

Issues with Perceptrons

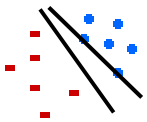
- Overtraining: test / held-out accuracy usually rises, then falls
  - Overtraining isn't quite as bad as overfitting, but is similar



- Regularization: if the data isn't separable, weights often thrash around
  - Averaging weight vectors over time can help (averaged perceptron)
  - [Freund & Schapire 99, Collins 02]



- Mediocre generalization: finds a "barely" separating solution

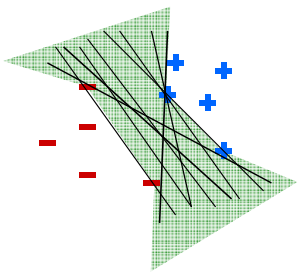


Problems with Perceptrons

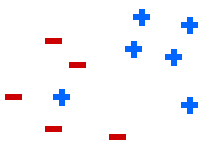
- Perceptron "goal": separate the training data

$$\forall i, \forall \mathbf{y} \neq \mathbf{y}^i \quad \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}^i) \geq \mathbf{w}^\top \mathbf{f}_i(\mathbf{y})$$

1. This may be an entire feasible space



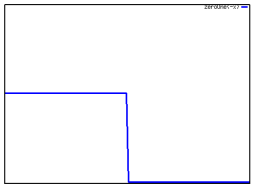
2. Or it may be impossible



Objective Functions

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

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

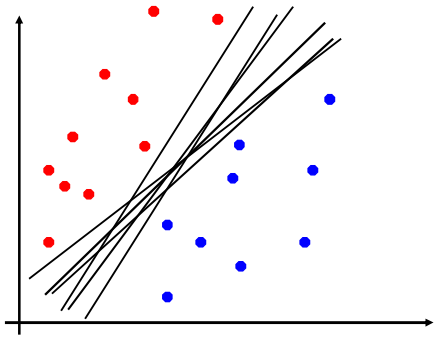


$\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}^i) - \max_{\mathbf{y}\neq\mathbf{y}_i^*} \mathbf{w}^\top \mathbf{f}_i(\mathbf{y})$

  - This is the “zero-one loss”
    - Discontinuous, minimizing is NP-complete
    - Not really what we want anyway
  - Maximum entropy and SVMs have other objectives related to zero-one loss

Linear Separators

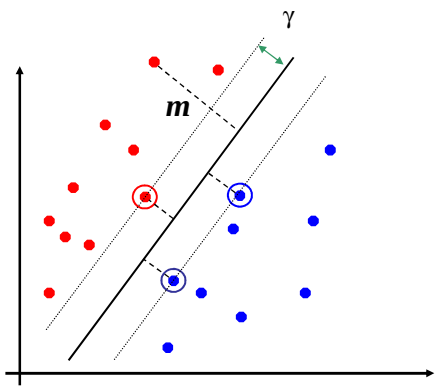
- Which of these linear separators is optimal?



28

Classification Margin (Binary)

- Distance of $\mathbf{x}_i$ to separator is its margin, m_i
- Examples closest to the hyperplane are support vectors
- Margin γ of the separator is the minimum m



Classification Margin

- For each example $\mathbf{x}_i$ and possible mistaken candidate $\mathbf{y}$, we avoid that mistake by a margin $m_i(\mathbf{y})$ (with zero-one loss)

$$m_i(\mathbf{y}) = \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}_i^*) - \mathbf{w}^\top \mathbf{f}_i(\mathbf{y})$$

- Margin γ of the entire separator is the minimum m

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

- It is also the largest γ for which the following constraints hold

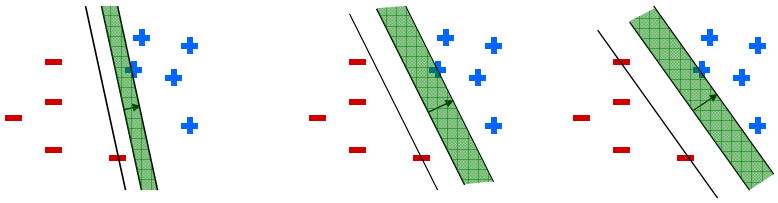
$$\forall i, \forall \mathbf{y} \quad \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}_i^*) \geq \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}) + \gamma \ell_i(\mathbf{y})$$

Maximum Margin

- Separable SVMs: find the max-margin …

$$\max_{||\mathbf{w}||=1} \gamma \qquad \ell_i(\mathbf{y}) = \begin{cases} 0 & \text{if } \mathbf{y} = \mathbf{y}_i^* \\ 1 & \text{if } \mathbf{y} \neq \mathbf{y}_i^* \end{cases}$$

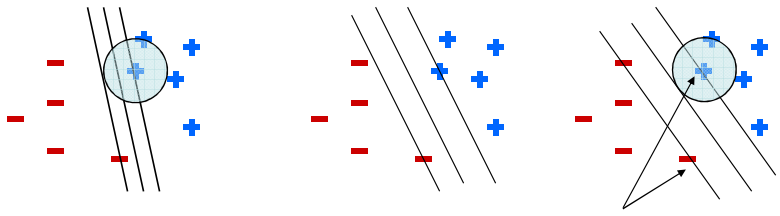
$$\forall i, \forall \mathbf{y} \quad \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}_i^*) \geq \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}) + \gamma \ell_i(\mathbf{y})$$



- 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



Support vectors

Max Margin / Small Norm

- Reformulation: find the smallest w which separates data

Remember this condition?

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

- γ scales linearly in w, so if ||w|| isn't constrained, we can take any separating w and scale up our margin

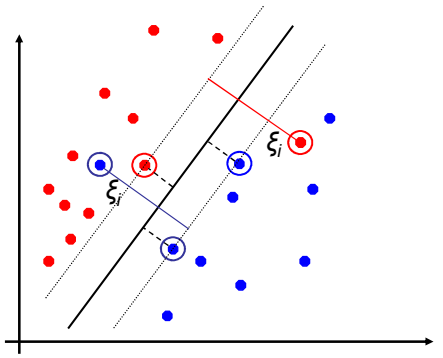
$$\gamma = \min_{i,\mathbf{y} \neq \mathbf{y}_i^*} [\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}_i^*) - \mathbf{w}^\top \mathbf{f}_i(\mathbf{y})] / \ell_i(\mathbf{y})$$

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

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

Soft Margin Classification

- What if the training set is not linearly separable?
- *Slack variables* $\boldsymbol{\xi}_i$ can be added to allow misclassification of difficult or noisy examples, resulting in a *soft margin* classifier



Maximum Margin

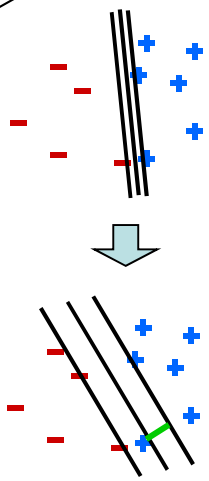
Note: exist other choices of how to penalize slacks!

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

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

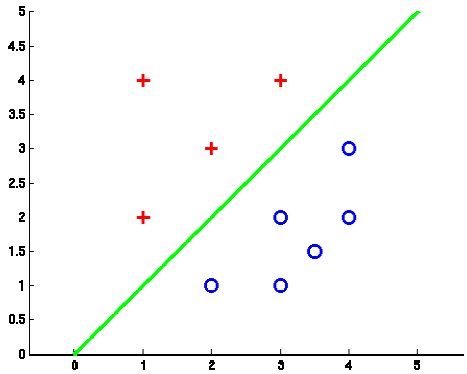
$$\forall i, \mathbf{y}, \quad \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}_i^*) + \xi_i \geq \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}) + \ell_i(\mathbf{y})$$

  - 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

Maximum Margin



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

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

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

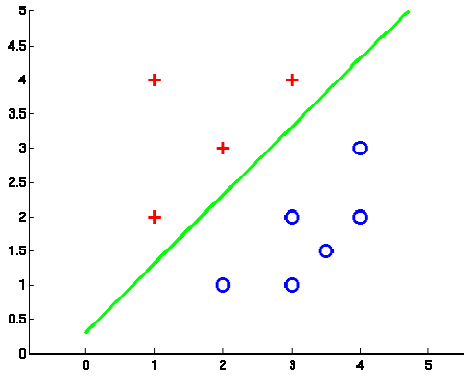
Maximum Entropy II

- 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)

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

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

Maximum Entropy



Log-Loss

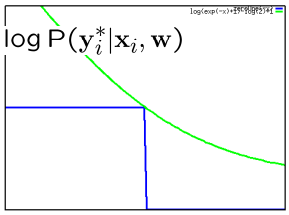
- If we view maxent as a minimization problem:

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

- This minimizes the "log loss" on each example

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

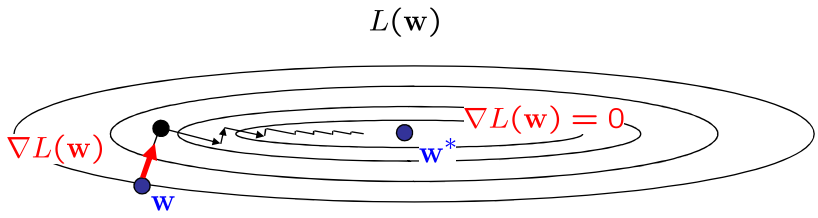
$$step\left(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}_i^*) - \max_{\mathbf{y} \neq \mathbf{y}_i^*} \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}) \right)$$



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

Unconstrained Optimization

- The maxent objective is an unconstrained optimization problem



- Basic idea: move uphill from current guess
- Gradient ascent / descent follows the gradient incrementally
- At local optimum, derivative vector is zero
- Will converge if step sizes are small enough, but not efficient
- All we need is to be able to evaluate the function and its derivative

Derivative for Maximum Entropy

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

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

Big weights are bad

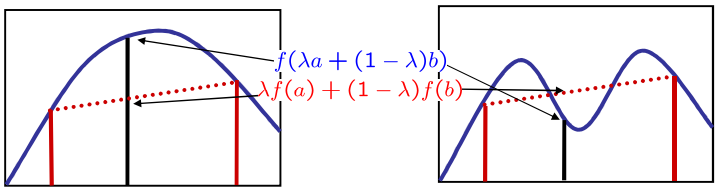
Total count of feature n in correct candidates

Expected feature vector over possible candidates

Convexity

- The maxent objective is nicely behaved:
  - *Differentiable* (so many ways to optimize)
  - *Convex* (so no local optima)

$$f(\lambda a + (1-\lambda)b) \geq \lambda f(a) + (1-\lambda) f(b)$$

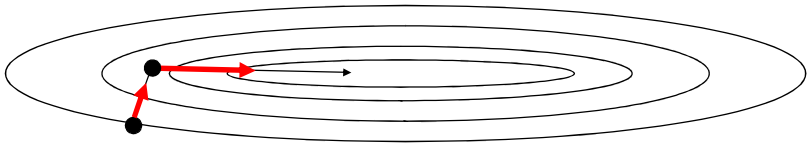


Convex Non-Convex

Convexity guarantees a single, global maximum value because any higher points are greedily reachable

Unconstrained Optimization

- Once we have a function f, we can find a local optimum by iteratively following the gradient



- For convex functions, a local optimum will be global
- Basic gradient ascent isn't very efficient, but there are simple enhancements which take into account previous gradients: conjugate gradient, L-BFGs
- There are special-purpose optimization techniques for maxent, like iterative scaling, but they aren't better

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

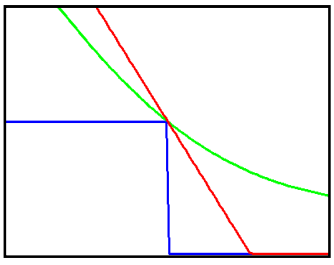
$$\min_{\mathbf{w}} \frac{1}{2}||\mathbf{w}||^2 - C\sum_i \left(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}_i^*) - \max_{\mathbf{y}} \left(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}) + \ell_i(\mathbf{y})\right)\right)$$

Hinge Loss

- Consider the per-instance objective:

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

- 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



Plot really only right in binary case

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

Max vs "Soft-Max" Margin

- SVMs:

$$\min_{\mathbf{w}} \; k||\mathbf{w}||^2 - \sum_i \left(\underbrace{\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}_i^*) - \max_{\mathbf{y}} \left(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}) + \ell_i(y) \right)}_{\text{You can make this zero}} \right)$$

- Maxent:

$$\min_{\mathbf{w}} \; k||\mathbf{w}||^2 - \sum_i \left(\underbrace{\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}_i^*) - \log \sum_{\mathbf{y}} \exp \left(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}) \right)}_{\text{… but not this one}} \right)$$

- 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

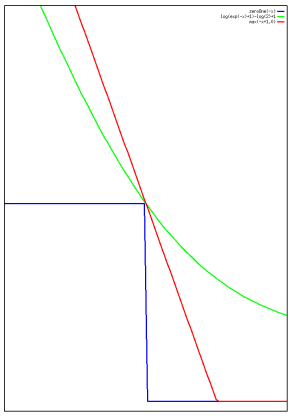
$$\sum_i step \left(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}_i^*) - \max_{\mathbf{y} \neq \mathbf{y}_i^*} \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}) \right)$$

- Hinge

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

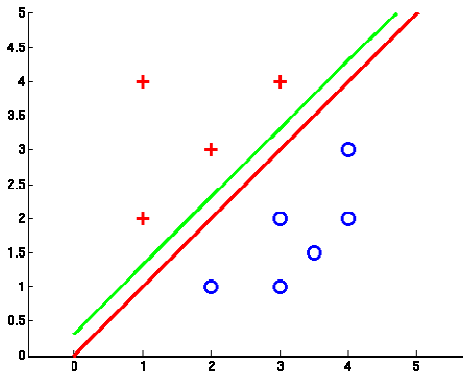
- Log

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



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

Separators: Comparison



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:

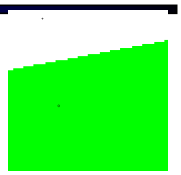
 $= \langle 0.0 \;\; 0.0 \;\; 0.3 \;\; 0.8 \;\; 0.7 \;\; 0.1 \ldots 0.0 \rangle$

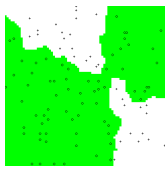
- Similarity function:
  - E.g. dot product of two images' vectors

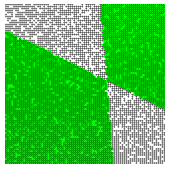
$$\text{sim}(x, y) = x^\top y = \sum_i x_i y_i$$

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

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

- 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

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

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

Example: Kernels

- Quadratic kernels

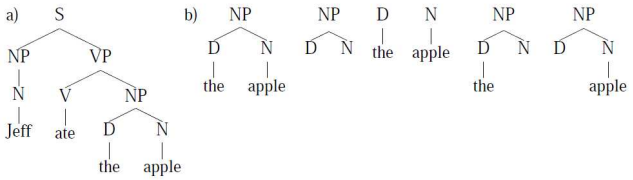
$$K(x, x') = (x \cdot x' + 1)^2$$

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

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

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:

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

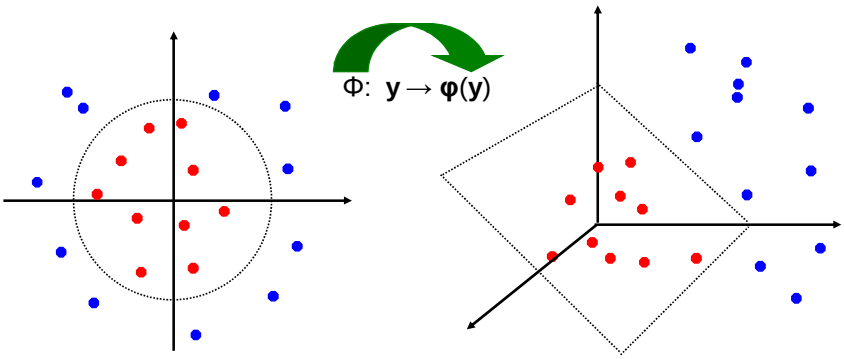
  - Base: if n, n' are share the same root production:

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

62

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…

Dual Formulation for SVMs

- 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:

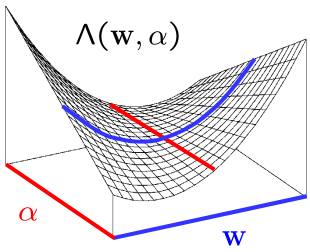
$$\min_{\mathbf{w}} \max_{\alpha \geq 0} \quad \Lambda(\mathbf{w}, \alpha) = \frac{1}{2}||\mathbf{w}||^2 - \sum_{i,\mathbf{y}} \alpha_i(\mathbf{y}) \left(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}_i^*) - \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}) - \ell_i(\mathbf{y}) \right)$$

- 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

Lagrange Duality

- We start out with a constrained optimization problem:

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



- 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):

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

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

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

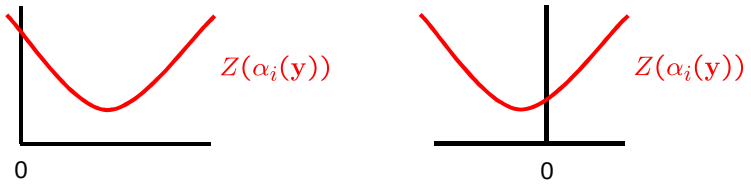
$$\forall i, \quad \sum_{\mathbf{y}} \alpha_i(\mathbf{y}) = C$$

- 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 $\alpha_i(\mathbf{y})$
- 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

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

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

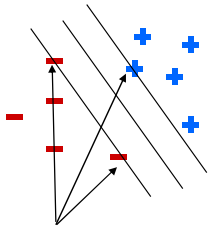
- This approach works fine for the separable case
- For the non-separable case, we just gain a simplex constraint and so we need slightly more complex methods (SMO, exponentiated gradient)

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

What are the Alphas?

- Each candidate corresponds to a primal constraint

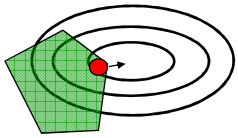
$$\min_{\mathbf{w},\xi} \quad \frac{1}{2}||\mathbf{w}||^2 + C\sum_i \xi_i$$
$$\forall i, \mathbf{y} \quad \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}_i^*) \geq \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}) + \ell_i(\mathbf{y}) - \xi_i$$



Support vectors

- In the solution, an $\alpha_i(\mathbf{y})$ 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:

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

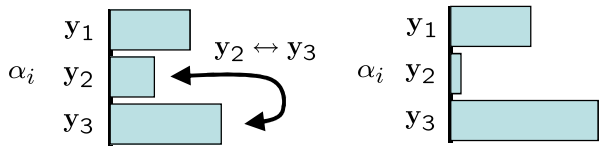


Bi-Coordinate Descent I

- In the non-separable case, it's (a little) harder:

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

- Here, we can't update just a single alpha, because of the sum-to-C constraints
- Instead, we can optimize two at once, shifting "mass" from one $\mathbf{y}$ to another:



Bi-Coordinate Descent II

- Choose an example *i*, and two labels y_1 and y_2:

$$t = \frac{(\ell_i(\mathbf{y}_1) - \ell_i(\mathbf{y}_2)) - \left(\sum_{i,\mathbf{y}} \alpha_i(\mathbf{y}) \left(\mathbf{f}_i(\mathbf{y}^i) - \mathbf{f}_i(\mathbf{y})\right)\right)^\top (\mathbf{f}_i(\mathbf{y}_2) - \mathbf{f}_i(\mathbf{y}_1))}{||\mathbf{f}_i(\mathbf{y}_2) - \mathbf{f}_i(\mathbf{y}_1)||^2}$$

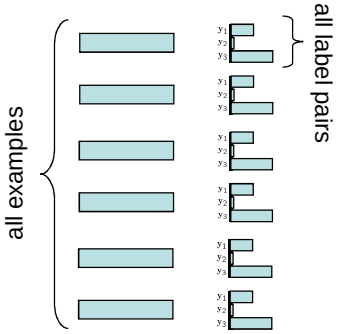
$$\mathbf{y}_1 \rightarrow \min(\mathbf{y}_1 + t, \mathbf{y}_1 + \mathbf{y}_2)$$
$$\mathbf{y}_2 \rightarrow \max(\mathbf{y}_2 - t, 0)$$

- This is a sequential minimal optimization update, but it's not the same one as in [Platt 98]

SMO

- Naïve SMO:

$\forall i \quad \alpha_i(\mathbf{y}^i) = C$
while (not converged) {
  visit each example *i* {
    for each pair of labels (y_1, y_2) {
      bi-coordinate-update(*i*, y_1, y_2)
    }
  }
}



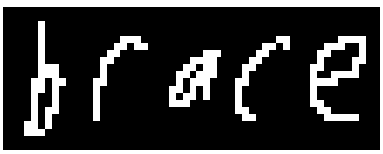
- Time per iteration: $O(|x||\mathcal{Y}|^2)$
- Smarter SMO:
  - Can speed this up by being clever about skipping examples and label pairs which will make little or no difference

Structure

77

Handwriting recognition

x **y**



brace

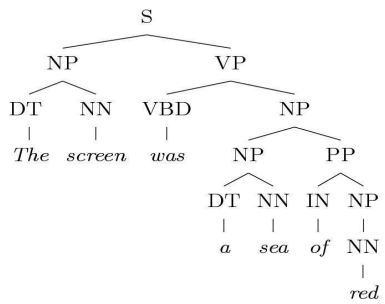
Sequential structure

CFG Parsing

x

The screen was a sea of red

y



Recursive structure

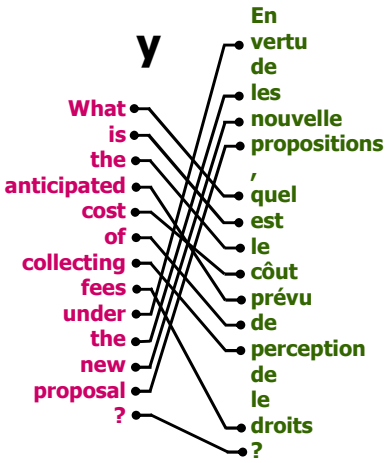
Bilingual word alignment

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

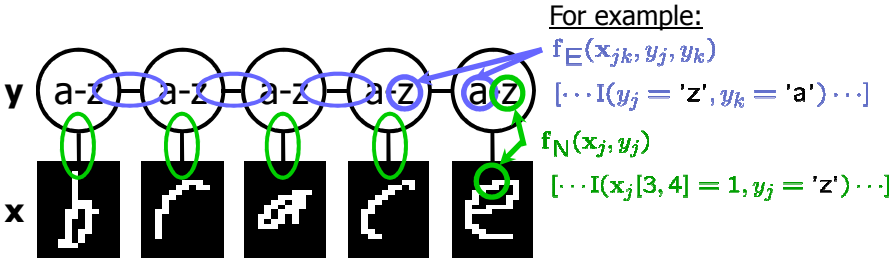
Parts = nodes, edges, productions

Chain Markov Net (aka CRF*)

$$P(\mathbf{y} \mid \mathbf{x}) \propto \Pi_j \phi(\mathbf{x}_j, y_j) \Pi_{jk} \phi(y_j, y_k)$$

$$\phi(\mathbf{x}_j, y_j) = \exp\left\{\mathbf{w}^\top \mathbf{f}_N(\mathbf{x}_j, y_j)\right\} \qquad N = \text{Node}$$

$$\phi(y_j, y_k) = \exp\left\{\mathbf{w}^\top \mathbf{f}_E(\mathbf{x}_{jk}, y_j, y_k)\right\} \qquad E = \text{Edge}$$



*Lafferty et al. 01

Chain Markov Net (aka CRF*)

$$P(\mathbf{y} \mid \mathbf{x}) \propto \prod_j \phi(\mathbf{x}_j, y_j) \prod_{jk} \phi(y_j, y_k) = \exp\left\{\mathbf{w}^\top \mathbf{f}(\mathbf{x}, \mathbf{y})\right\}$$

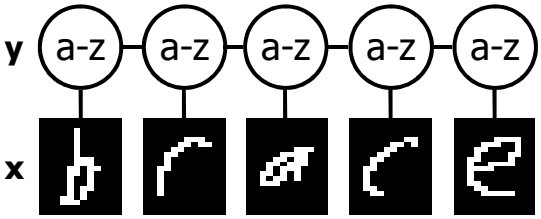
$$\prod_j \phi(\mathbf{x}_j, y_j) = \exp\left\{\sum_j \mathbf{w}^\top \mathbf{f}_\mathsf{N}(\mathbf{x}_j, y_j)\right\} = \exp\left\{\mathbf{w}^\top \mathbf{f}_\mathsf{N}(\mathbf{x}, \mathbf{y})\right\}$$

$$\prod_{jk} \phi(y_j, y_k) = \exp\left\{\sum_{jk} \mathbf{w}^\top \mathbf{f}_\mathsf{E}(\mathbf{x}_{jk}, y_j, y_k)\right\} = \exp\left\{\mathbf{w}^\top \mathbf{f}_\mathsf{E}(\mathbf{x}, \mathbf{y})\right\}$$

$$\mathbf{f}_\mathsf{E}(\mathbf{x}, \mathbf{y}) \equiv \sum_{jk} \mathbf{f}_\mathsf{E}(\mathbf{x}_{jk}, y_j, y_k)$$

$$\mathbf{f}_\mathsf{N}(\mathbf{x}, \mathbf{y}) \equiv \sum_{j} \mathbf{f}_\mathsf{E}(\mathbf{x}_j, y_j)$$

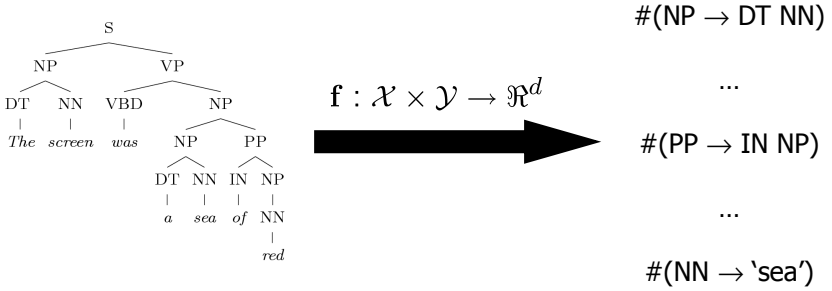
$$\mathbf{f}(\mathbf{x}, \mathbf{y}) \equiv \begin{pmatrix} \mathbf{f}_\mathsf{N}(\mathbf{x}, \mathbf{y}) \\ \mathbf{f}_\mathsf{E}(\mathbf{x}, \mathbf{y}) \end{pmatrix}$$



*Lafferty et al. 01

CFG Parsing

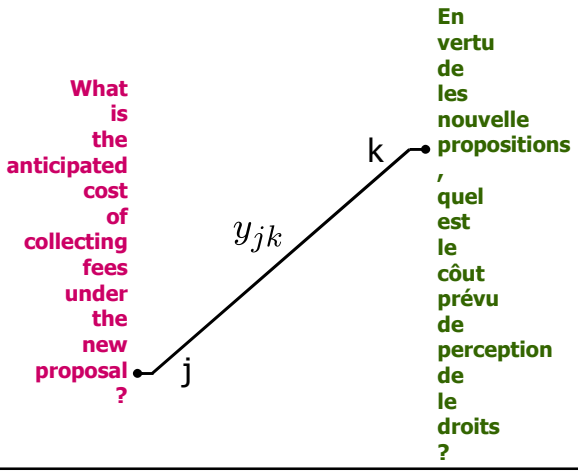
$$P(\mathbf{y} \mid \mathbf{x}) \propto \prod_{A \to \alpha \in (\mathbf{x}, \mathbf{y})} \phi(A \to \alpha)$$



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

Bilingual word alignment

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



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

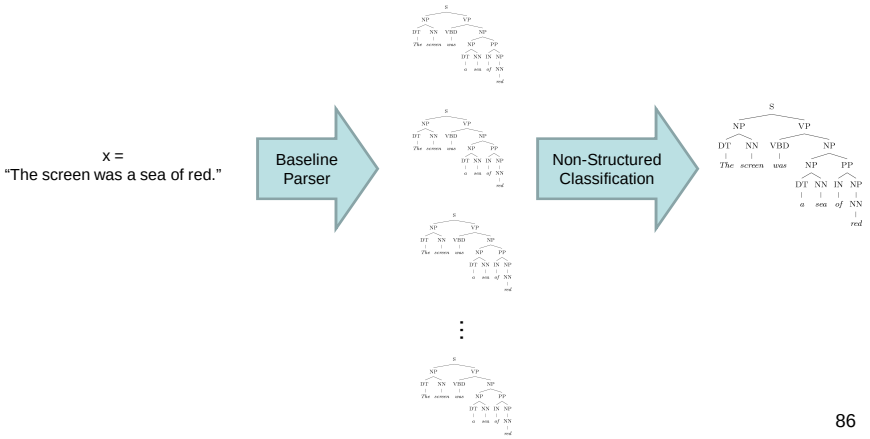
- association
- position
- orthography

Option 0: Reranking

[e.g. Charniak and Johnson 05]

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

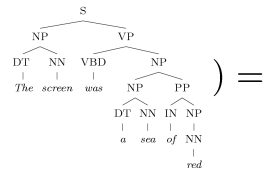
x = "The screen was a sea of red."



86

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]

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

88

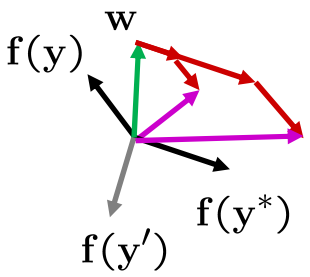
MIRA / Perceptron

- Idea: use perceptron but be smarter about the updates
- MIRA*: choose an update size that fixes the current mistake…
- … but, minimizes the change to w

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

$$\mathbf{w}^\top(\mathbf{y}^*) \geq \mathbf{w}^\top(\mathbf{y}) + \ell(\mathbf{y})$$

- This should remind you of the margin objective



Guessed $\mathbf{y}$ instead of $\mathbf{y}^*$ on example $\mathbf{x}$

$$\mathbf{w} = \mathbf{w}' - \tau \mathbf{f}(\mathbf{y})$$
$$\mathbf{w} = \mathbf{w}' + \tau \mathbf{f}(\mathbf{y}^*)$$

* Margin Infused Relaxed Algorithm, [Crammer and Singer 03]

Minimum Correcting Update

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

$$\mathbf{w}^\top \mathbf{f}(\mathbf{y}^*) \geq \mathbf{w}^\top \mathbf{f}(\mathbf{y}) + \ell(\mathbf{y})$$

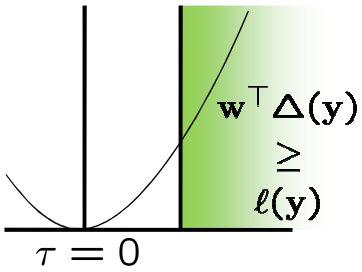
$$\min_{\tau} ||\tau \Delta(\mathbf{y})||^2$$

$$\mathbf{w}^\top \mathbf{f}(\mathbf{y}^*) \geq \mathbf{w}^\top \mathbf{f}(\mathbf{y}) + \ell(\mathbf{y})$$

$$\left(\mathbf{w}' + \tau \Delta(\mathbf{y})\right)^\top \Delta(\mathbf{y}) = \ell(\mathbf{y})$$

$$\tau = \frac{\ell(\mathbf{y}) - \mathbf{w}'^\top \Delta(\mathbf{y})}{||\Delta(\mathbf{y})||^2}$$

$$\boxed{\mathbf{w} = \mathbf{w}' + \tau \Delta(\mathbf{y})}$$



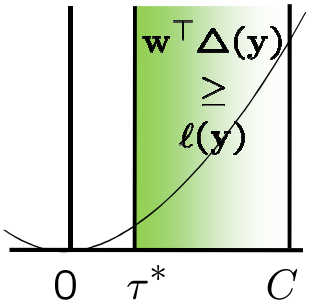
min not τ=0, or would not have made an error, so min will be where equality holds

Maximum Step Size

- In practice, it's also bad to make updates that are too large
  - Example may be labeled incorrectly
  - You may not have enough features
  - Solution: cap the maximum possible value of τ with some constant C

$$\tau^* = \min(\tau, C)$$

  - This should remind you of the sum-to-C constraints in the soft-margin SVM



91

MIRA

- Some important points:
  - The general version of MIRA considers the top-K predictions when choosing the update; no one uses this
  - MIRA needs to be averaged (just like the perceptron)

92

Structured Margin

- Remember the margin objective:

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

- This is still defined, but lots of constraints

Full Margin: OCR

- We want:

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

- Equivalently:

$\mathbf{w}^\top \mathbf{f}($  , "brace" $) > \mathbf{w}^\top \mathbf{f}($, "aaaaa" $)$

$\mathbf{w}^\top \mathbf{f}($  , "brace" $) > \mathbf{w}^\top \mathbf{f}($, "aaaab" $)$

...

$\mathbf{w}^\top \mathbf{f}($  , "brace" $) > \mathbf{w}^\top \mathbf{f}($, "zzzzz" $)$

a lot!

Parsing example

- We want:

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

- Equivalently:

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

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

$$\dots$$

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

a lot!

Alignment example

- We want:

$$\arg\max_y \; w^\top f(\text{'What is the'}, \text{'Quel est le'}, y) = \begin{matrix}1 & 1\\ 2 & 2\\ 3 & 3\end{matrix}$$

- Equivalently:

$$w^\top f(\text{'What is the'}, \text{'Quel est le'}, \begin{matrix}1 & 1\\ 2 & 2\\ 3 & 3\end{matrix}) > w^\top f(\text{'What is the'}, \text{'Quel est le'}, \begin{matrix}1 & 1\\ 2 & 2\\ 3 & 3\end{matrix})$$

$$w^\top f(\text{'What is the'}, \text{'Quel est le'}, \begin{matrix}1 & 1\\ 2 & 2\\ 3 & 3\end{matrix}) > w^\top f(\text{'What is the'}, \text{'Quel est le'}, \begin{matrix}1 & 1\\ 2 & 2\\ 3 & 3\end{matrix})$$

$$\dots$$

$$w^\top f(\text{'What is the'}, \text{'Quel est le'}, \begin{matrix}1 & 1\\ 2 & 2\\ 3 & 3\end{matrix}) > w^\top f(\text{'What is the'}, \text{'Quel est le'}, \begin{matrix}1 & 1\\ 2 & 2\\ 3 & 3\end{matrix})$$

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:

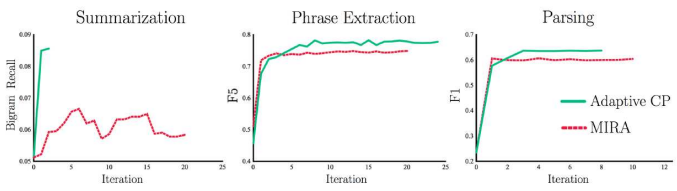
$$\forall \mathbf{y} \quad \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}_i^*) \geq \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}) + \ell_i(\mathbf{y})$$

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

  - 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 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)