

# Algorithms for NLP



## Classification II

Taylor Berg-Kirkpatrick – CMU

Slides: Dan Klein – UC Berkeley



# Results

---

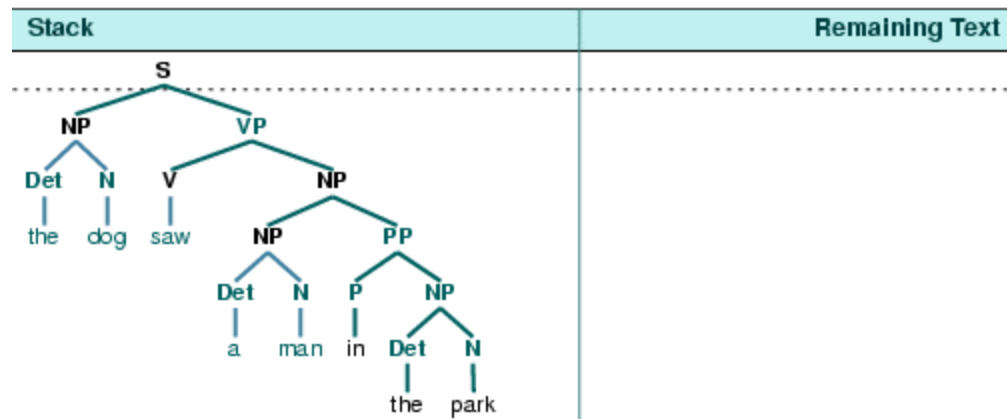
- Stanford Parser – 86.3 (unlex / struct annotation)
- Collins 99 – 88.6 F1 (lexical)
- Charniak and Johnson 05 – 89.7 / 91.3 F1 (lexical + rerank)
- McClosky et al 06 – 92.1 F1 (lexical + rerank + self-train)
  
- Petrov et al 06 – 90.7 F1 (unlex / latent vars)
- Petrov et al 10 – 91.8 (unlex / latent vars + ensemble)
  
- Socher et al 13 – 90.4 (unlex + neural rerank)
- Vinyals et al 15 – 90.5 / 92.1 (neural sequence + self-train)
- Dyer et al 16 – 92.4 (neural shift-reduce)

...many more that are really cool (e.g. Hall and Klein 12,14)



# Shift-Reduce Parsers

- Another way to derive a tree:



- Parsing**
  - No useful dynamic programming search
  - Can still use beam search [Ratnaparkhi 97]

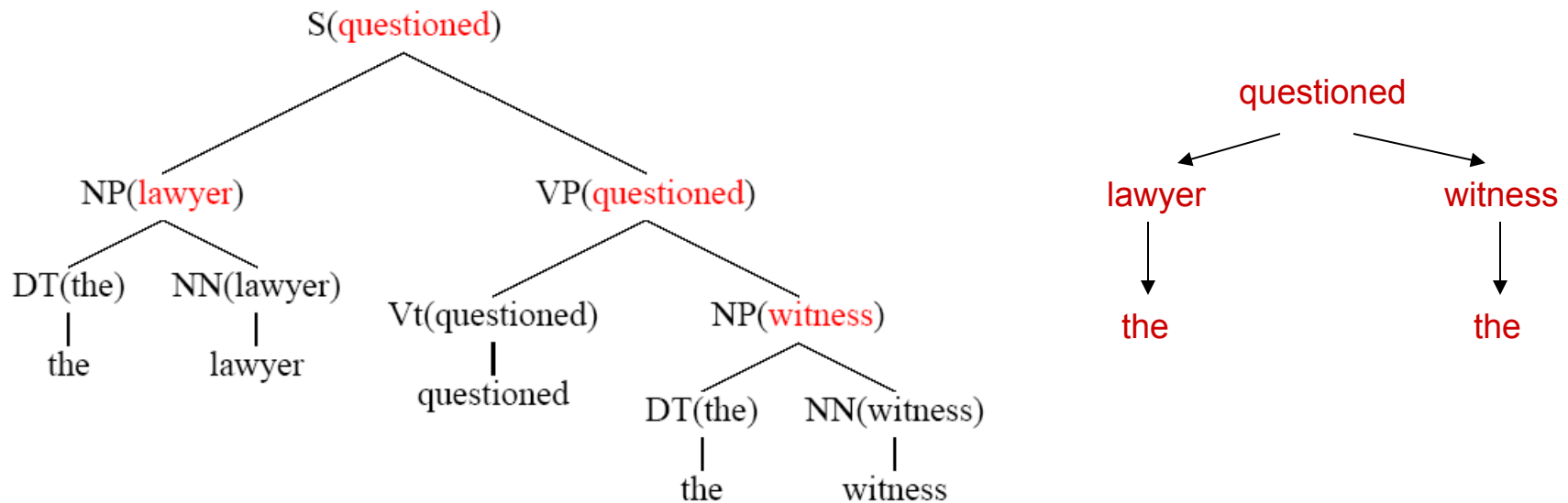
# Other Syntactic Models





# Dependency Parsing

- Lexicalized parsers can be seen as producing *dependency trees*

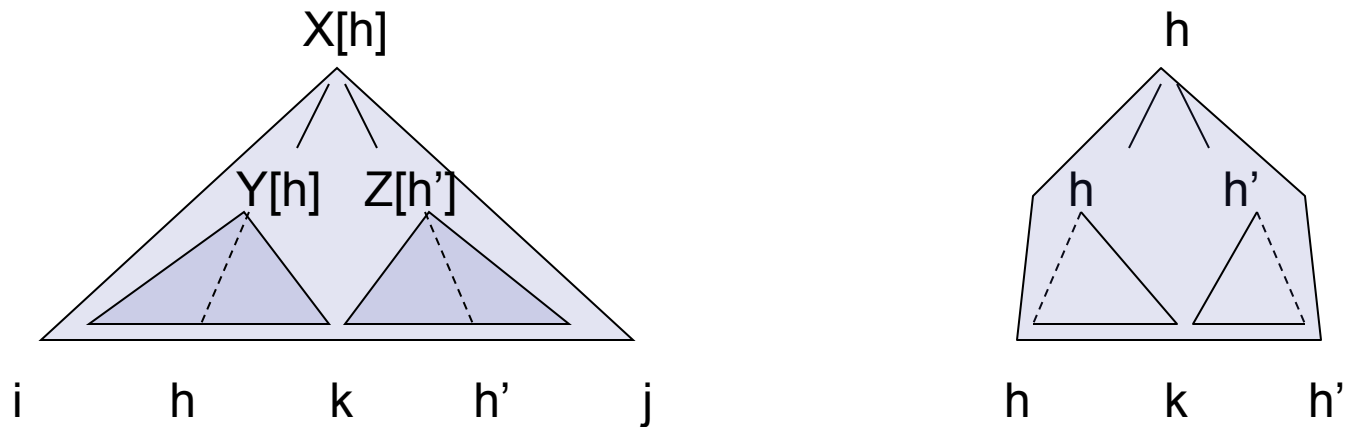


- Each local binary tree corresponds to an attachment in the dependency graph



# Dependency Parsing

- Pure dependency parsing is only cubic [Eisner 99]



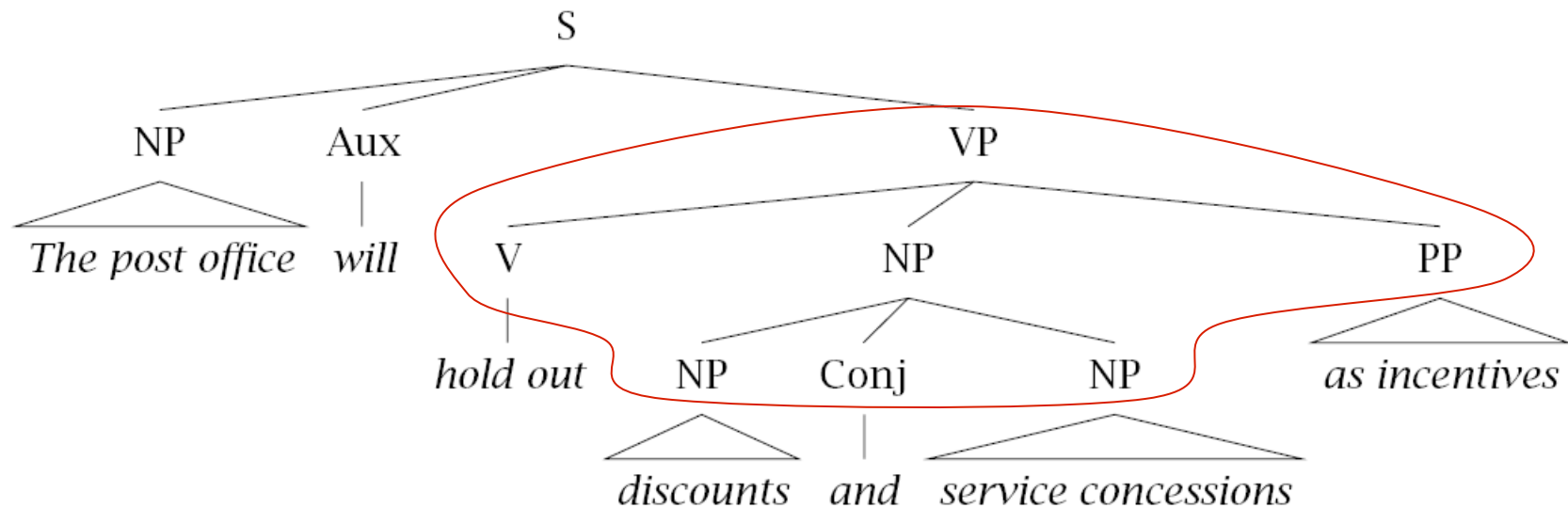
- Some work on *non-projective* dependencies
  - Common in, e.g. Czech parsing
  - Can do with MST algorithms [McDonald and Pereira 05]





# Tree Insertion Grammars

- Rewrite large (possibly lexicalized) subtrees in a single step

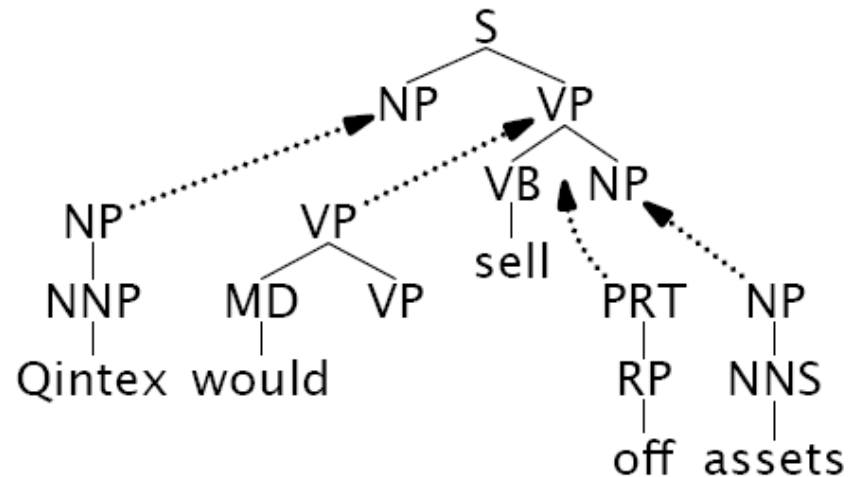
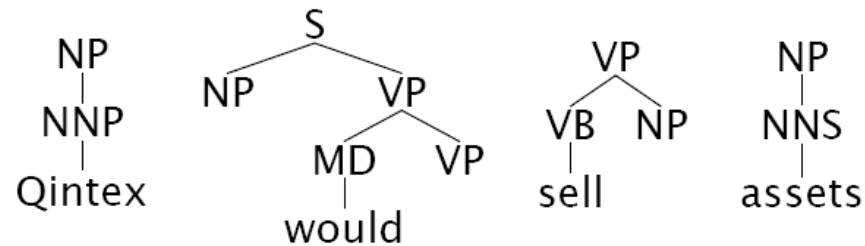


- Formally, a *tree-insertion grammar*
- Derivational ambiguity whether subtrees were generated atomically or compositionally
- Most probable *parse* is NP-complete



# Tree-adjoining grammars

- Start with *local trees*
- Can insert structure with *adjunction* operators
- Mildly context-sensitive
- Models long-distance dependencies naturally
- ... as well as other weird stuff that CFGs don't capture well (e.g. cross-serial dependencies)





# CCG Parsing

- Combinatory  
Categorial Grammar

- Fully (mono-) lexicalized grammar
- Categories encode argument sequences
- Very closely related to the lambda calculus (more later)
- Can have spurious ambiguities (why?)

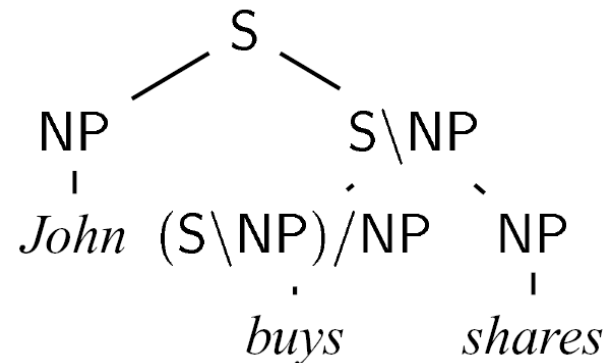
*John*  $\vdash$  NP

*shares*  $\vdash$  NP

*buys*  $\vdash$  (S\NP)/NP

*sleeps*  $\vdash$  S\NP

*well*  $\vdash$  (S\NP)\(S\NP)



# Classification



# Classification

---

- Automatically make a decision about inputs
  - Example: document → category
  - Example: image of digit → digit
  - Example: image of object → object type
  - Example: query + webpages → best match
  - Example: symptoms → diagnosis
  - ...
- Three main ideas
  - Representation as feature vectors / kernel functions
  - Scoring by linear functions
  - Learning by optimization



# Some Definitions

INPUTS

$\mathbf{x}_i$

*close the \_\_\_\_\_*

CANDIDATE SET

$\mathcal{Y}(\mathbf{x})$

*{door, table, ...}*

CANDIDATES

$y$

*table*

TRUE OUTPUTS

$y_i^*$

*door*

FEATURE VECTORS

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

$x_{-1} = \text{"the"} \wedge y = \text{"door"}$

$x_{-1} = \text{"the"} \wedge y = \text{"table"}$

*"close" in x  $\wedge$  y = "door"*

*y occurs in x*



# Features

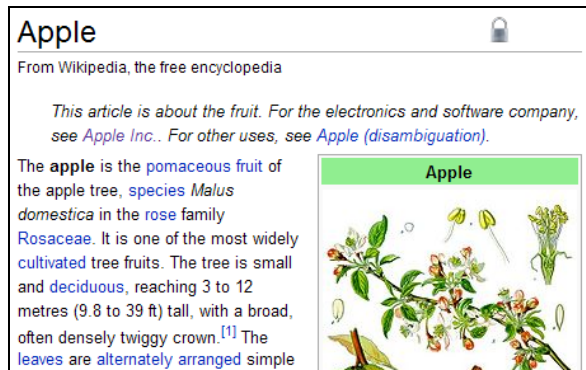


# Feature Vectors

- Example: web page ranking (not actually classification)

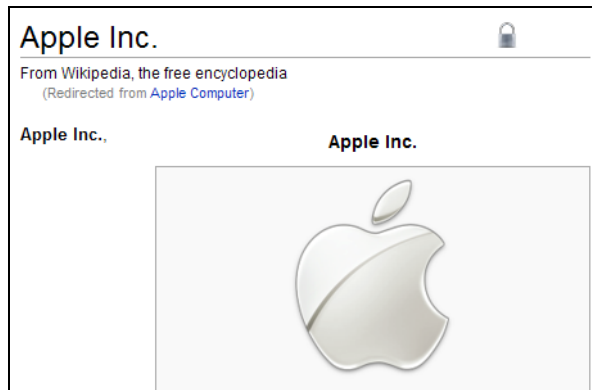
$x_i$  = "Apple Computers"

$f_i(\$



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

$f_i(\$



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



# Block Feature Vectors

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

$x$       ... win the election ...



"f(x)"

[1 0 1 0]

"win"

"election"



... win the election ...

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

... win the election ...

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

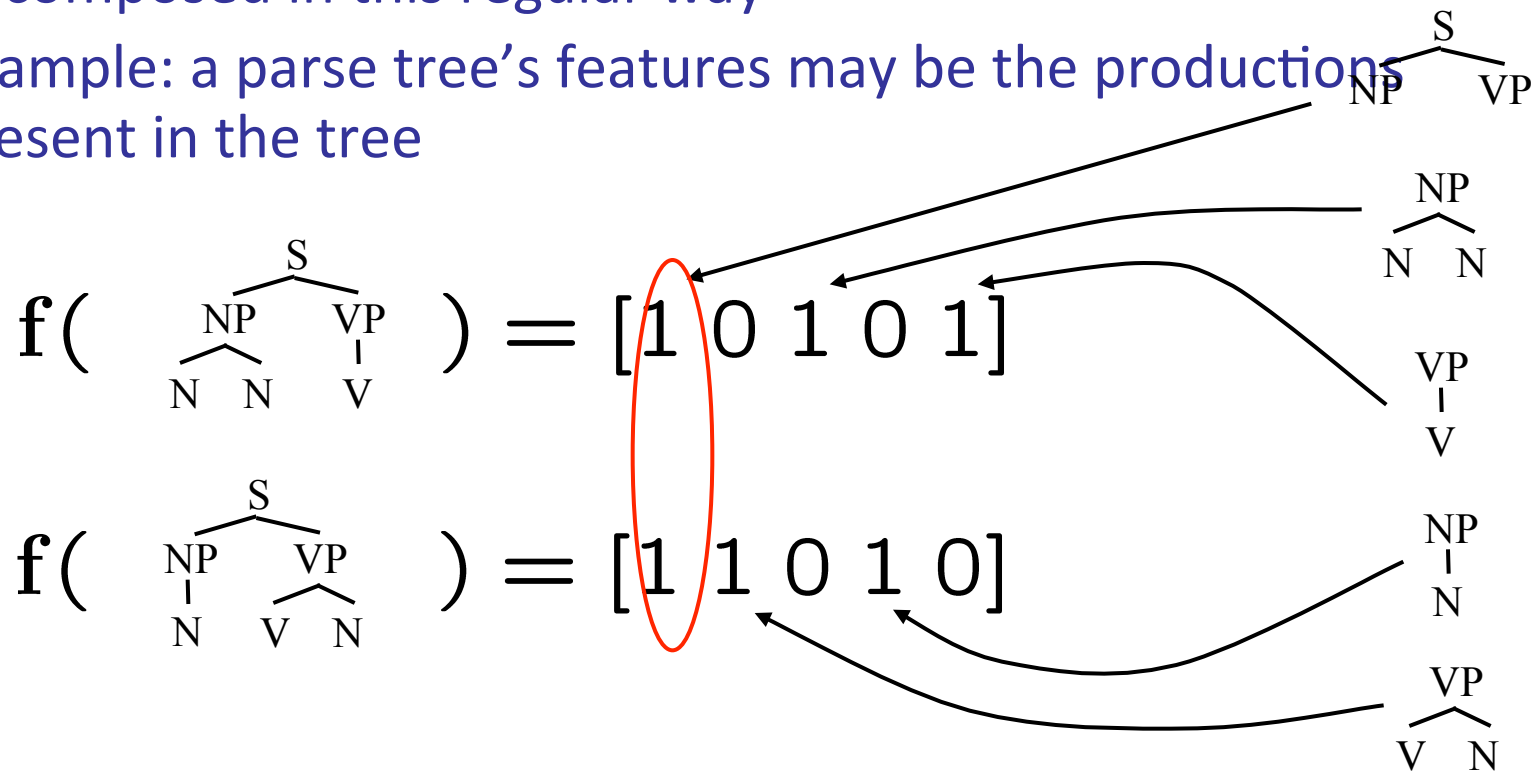
... win the election ...

$$f(\textit{OTHER}) = [0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0]$$



# Non-Block Feature Vectors

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



- Different candidates will thus often share features
- We'll return to the non-block case later

# Linear Models



# Linear Models: Scoring

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

$$\begin{aligned} \mathbf{f}(\overset{\dots \text{win the election} \dots}{POLITICS}) &= [ 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 ] \\ \mathbf{f}(\overset{\dots \text{win the election} \dots}{SPORTS}) &= [ 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 ] \\ \mathbf{w} &= [ 1 \quad 1 \quad -1 \quad -2 \quad 1 \quad -1 \quad 1 \quad -2 \quad -2 \quad -1 \quad -1 \quad 1 ] \end{aligned}$$

- We score hypotheses by multiplying features and weights:

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

$$\begin{aligned} \mathbf{f}(\overset{\dots \text{win the election} \dots}{POLITICS}) &= [ 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 ] \\ \mathbf{w} &= [ 1 \quad 1 \quad -1 \quad -2 \quad 1 \quad -1 \quad 1 \quad -2 \quad -2 \quad -1 \quad -1 \quad 1 ] \end{aligned}$$

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



# Linear Models: Decision Rule

---

- The linear decision rule:

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

$$\text{score}(\overset{\dots \text{win the election } \dots}{\text{SPORTS}}, \mathbf{w}) = 1 \times 1 + (-1) \times 1 = 0$$

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

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



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

- We've said nothing about where weights come from



# Binary Classification

- Important special case: binary classification

- Classes are  $y=+1/-1$

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

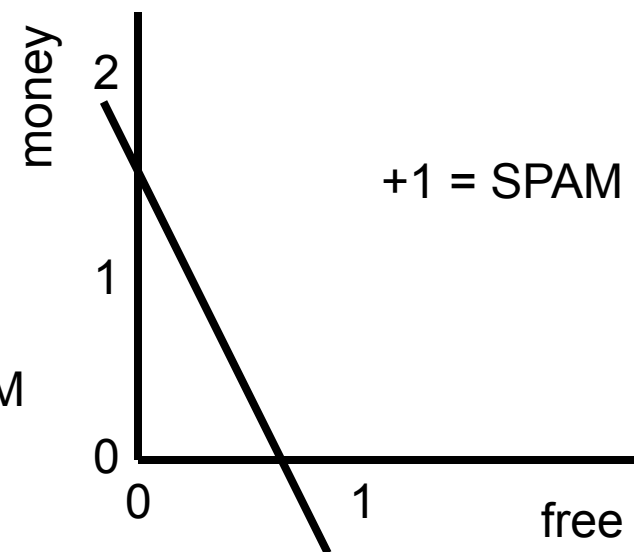
$$f(\mathbf{x}) = 2f(\mathbf{x}, +1)$$

- Decision boundary is a hyperplane

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

$\mathbf{w}$

BIAS	:	-3
free	:	4
money	:	2

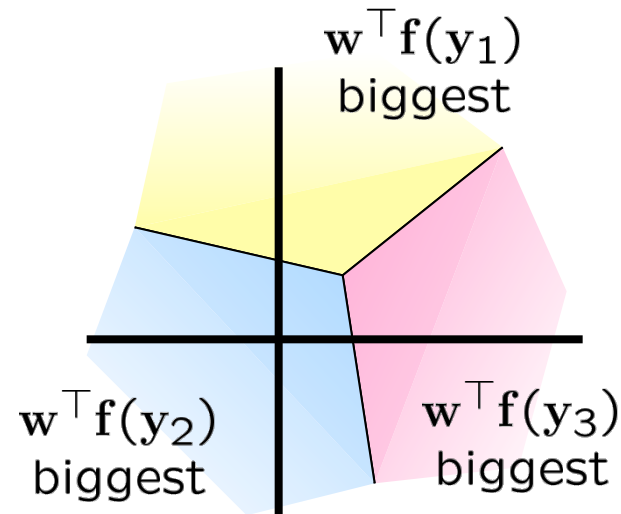






# Multiclass Decision Rule

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



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

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

Learning



# Learning Classifier Weights

---

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



# How to pick weights?

---

- Goal: choose “best” vector  $w$  given training data
  - For now, we mean “best for classification”
- The ideal: the weights which have greatest test set accuracy / F1 / whatever
  - But, don’t have the test set
  - Must compute weights from training set
- Maybe we want weights which give best training set accuracy?
  - Hard discontinuous optimization problem
  - May not (does not) generalize to test set
  - Easy to overfit

*Though, min-error training for MT does exactly this.*



# Minimize Training Error?

---

- A loss function declares how costly each mistake is

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

- 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

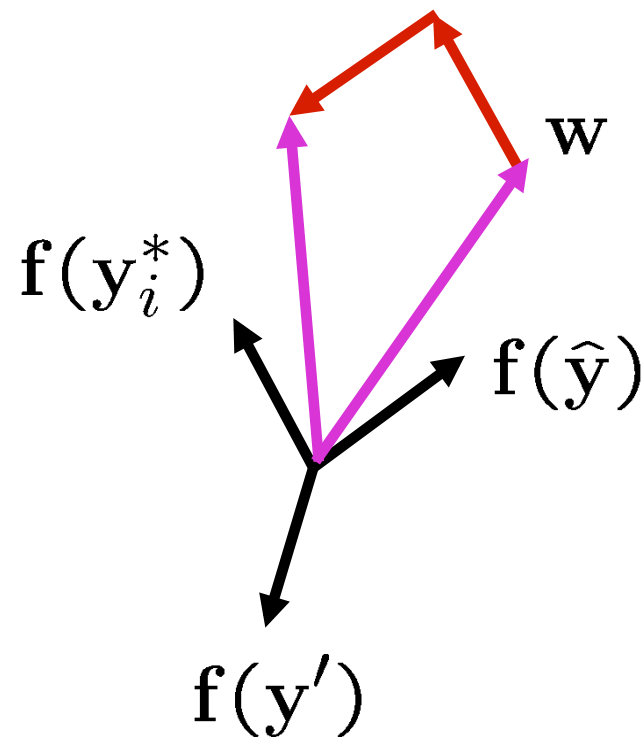
- The perceptron algorithm
  - Iteratively processes the training set, reacting to training errors
  - Can be thought of as trying to drive down training error
- The (online) perceptron algorithm:
  - Start with zero weights  $w$
  - Visit training instances one by one
    - Try to classify

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

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

$$w \leftarrow w + f(y_i^*)$$

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



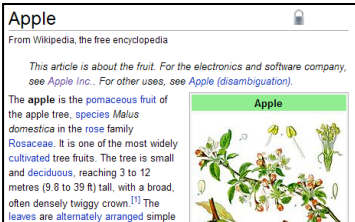


# Example: "Best" Web Page

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

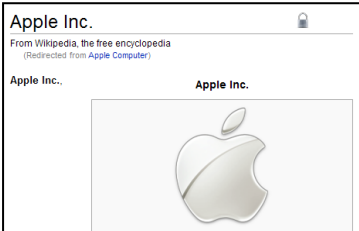
$x_i = \text{"Apple Computers"}$

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



$$\mathbf{w}^\top \mathbf{f} = 10.3 \quad \hat{y}_i$$

$f_i(\text{Apple Inc.}) = [0.8 \ 4 \ 2 \ 1 \ \dots]$



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

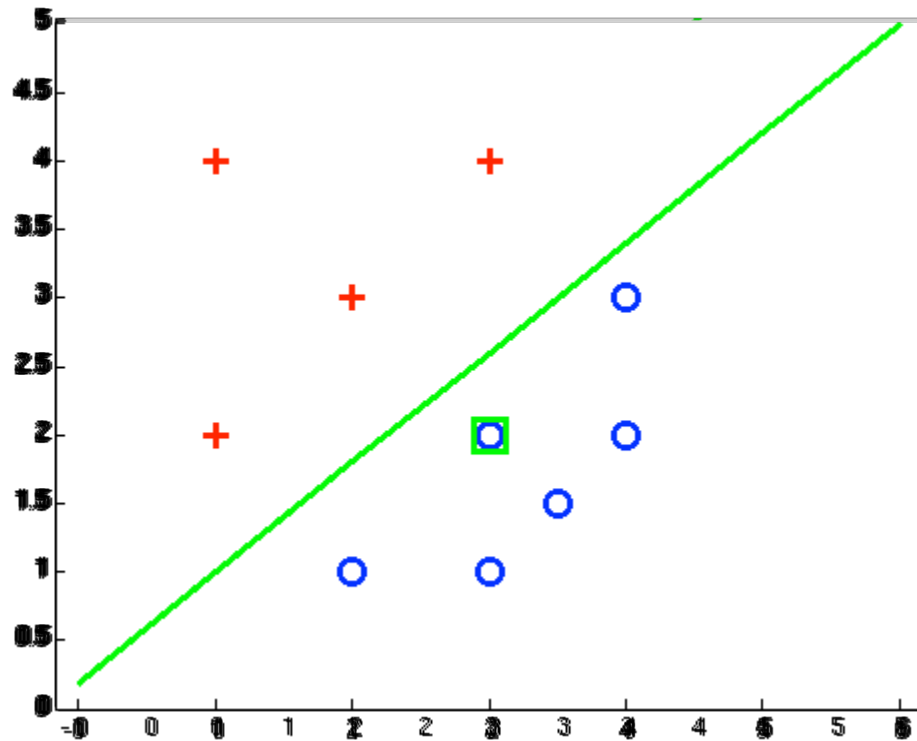
$$\mathbf{w} \leftarrow \mathbf{w} + \mathbf{f}(y_i^*) - \mathbf{f}(\hat{y}_i)$$

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



# Examples: Perceptron

- Separable Case





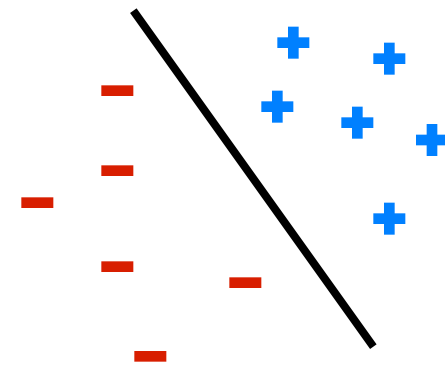


# Perceptrons and Separability

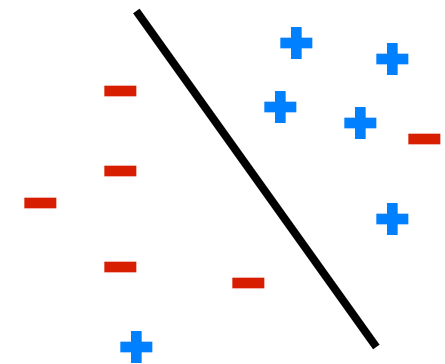
---

- 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



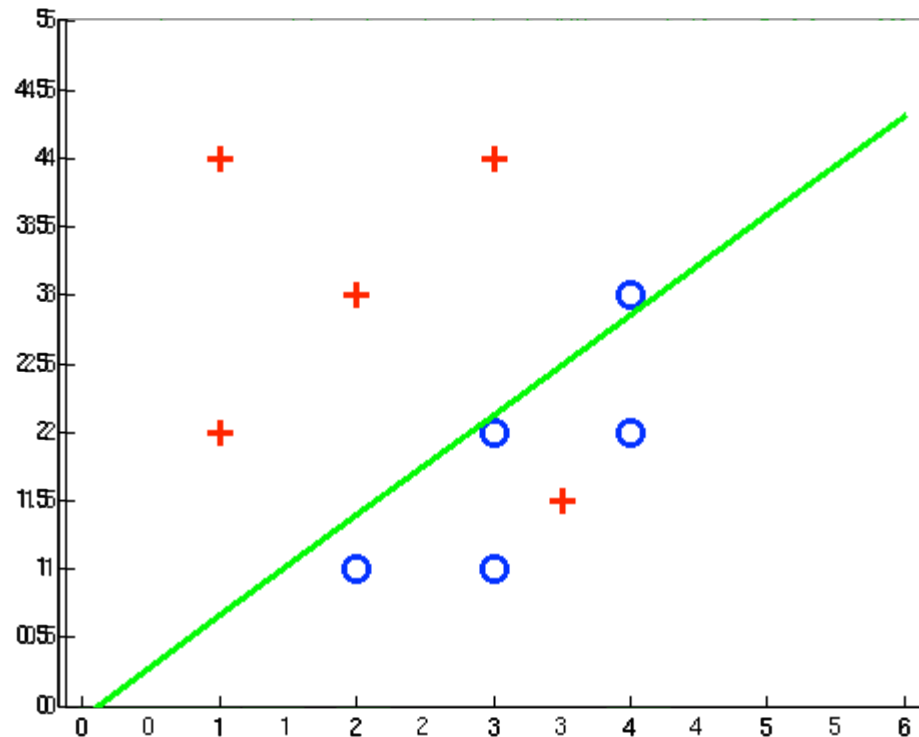
Non-Separable





# Examples: Perceptron

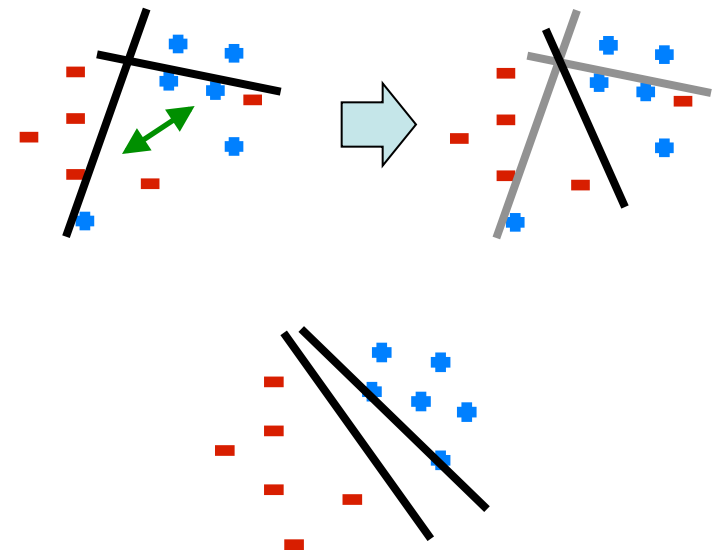
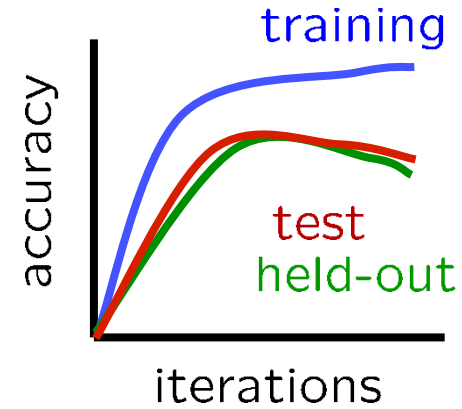
- Non-Separable Case





# Issues with Perceptrons

- **Overtraining:** test / held-out accuracy usually rises, then falls
  - Overtraining isn't the typically discussed source of overfitting, but it can be important
- **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



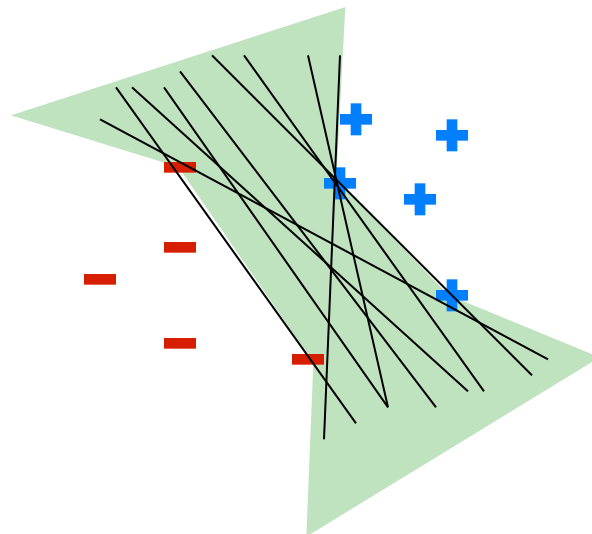


# Problems with Perceptrons

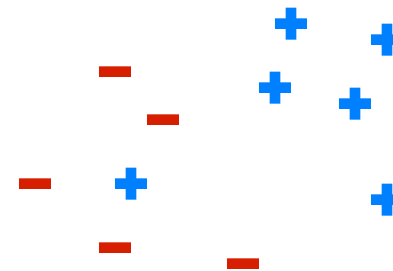
- Perceptron “goal”: separate the training data

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

1. This may be an entire feasible space



2. Or it may be impossible



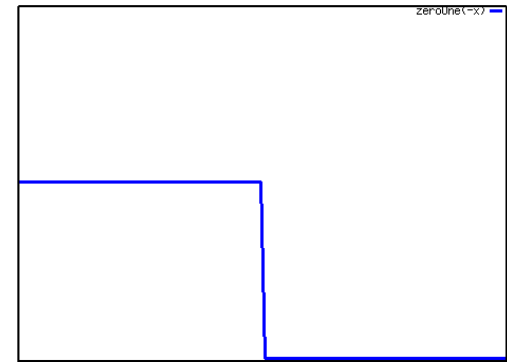
Margin



# Objective Functions

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

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



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

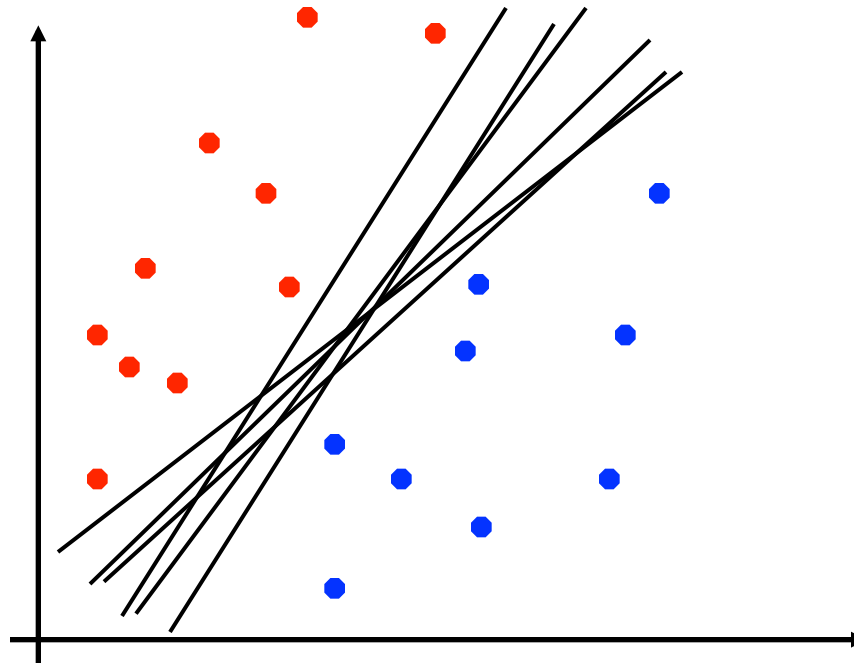
- This is the “zero-one loss”
  - Discontinuous, minimizing is NP-complete
  - Not really what we want anyway
- Maximum entropy and SVMs have other objectives related to zero-one loss



# Linear Separators

---

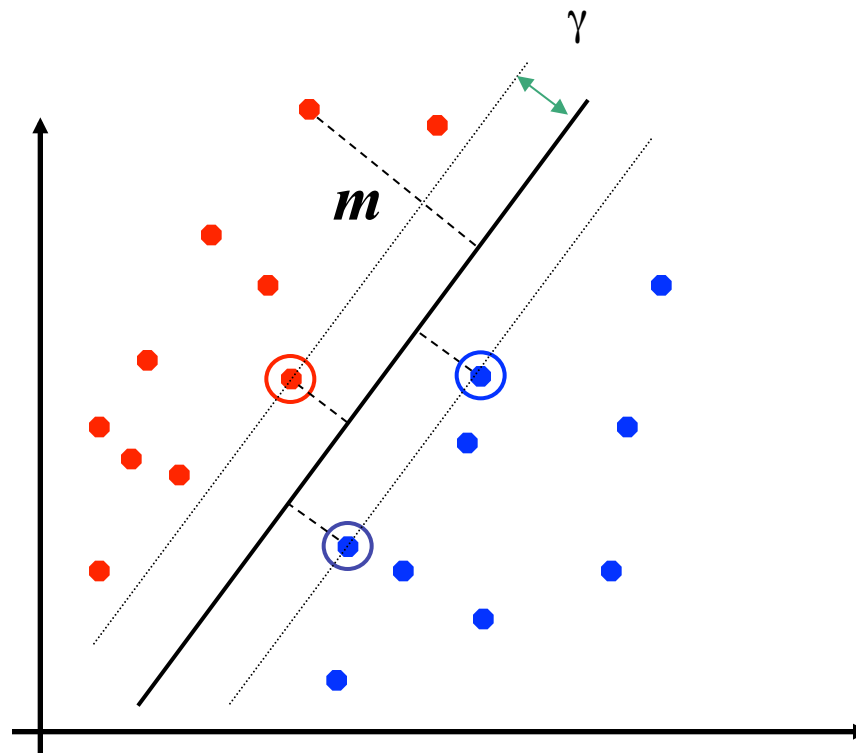
- Which of these linear separators is optimal?





# Classification Margin (Binary)

- Distance of  $x_i$  to separator is its margin,  $m_i$
- Examples closest to the hyperplane are **support vectors**
- **Margin**  $\gamma$  of the separator is the minimum  $m$







# Classification Margin

---

- For each example  $\mathbf{x}_i$  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  $\gamma$  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  $\gamma$  for which the following constraints hold

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



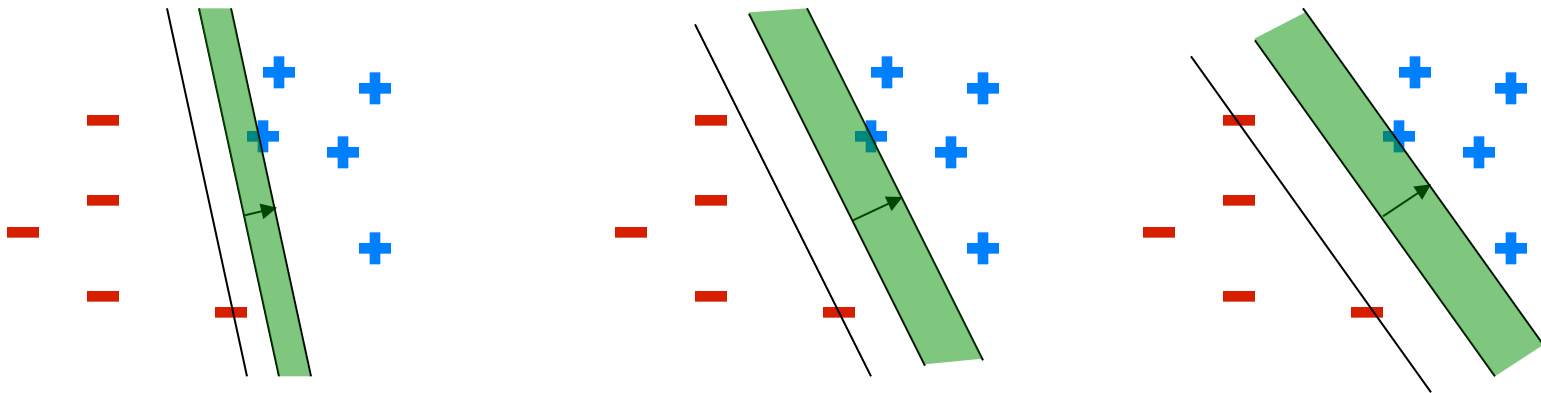
# Maximum Margin

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

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

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

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

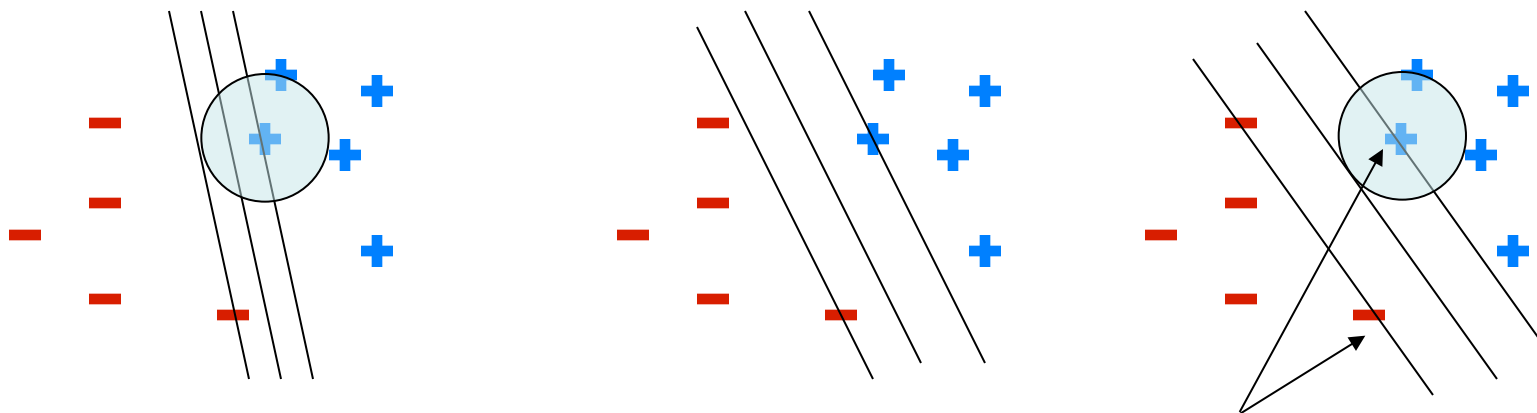


- Can stick this into Matlab and (slowly) get an SVM
- Won't work (well) if non-separable



# Why Max Margin?

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



*Support vectors*



# Max Margin / Small Norm

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

Remember this condition?

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

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

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

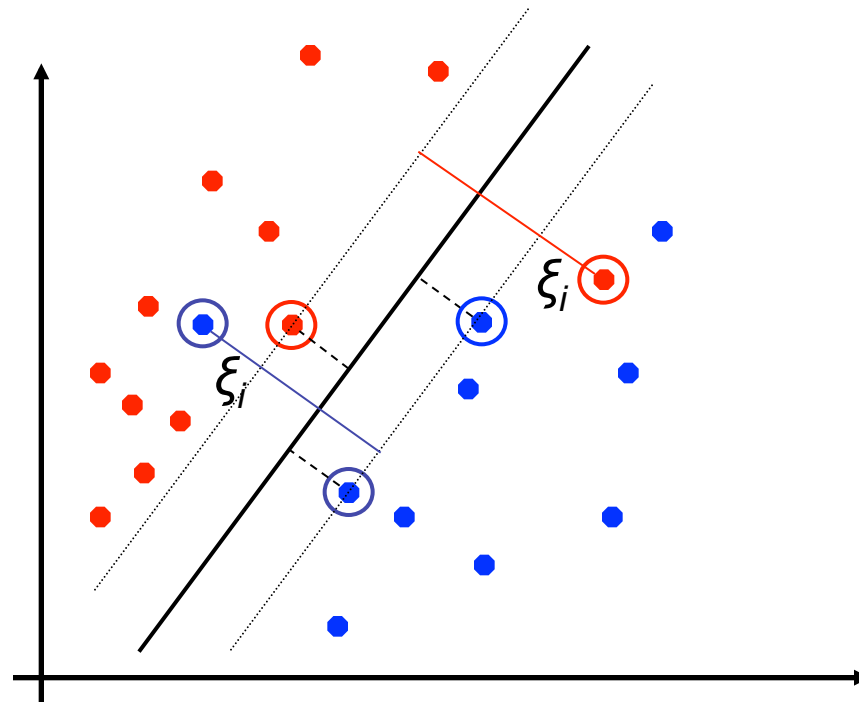
- Instead of fixing the scale of  $w$ , we can fix  $\gamma = 1$

$$\min_w \frac{1}{2} \|w\|^2$$
$$\forall i, y \quad w^\top f_i(y_i^*) \geq w^\top f_i(y) + 1 l_i(y)$$



# Soft Margin Classification

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





# Maximum Margin

Note: exist other choices of how to penalize slacks!

## ■ Non-separable SVMs

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

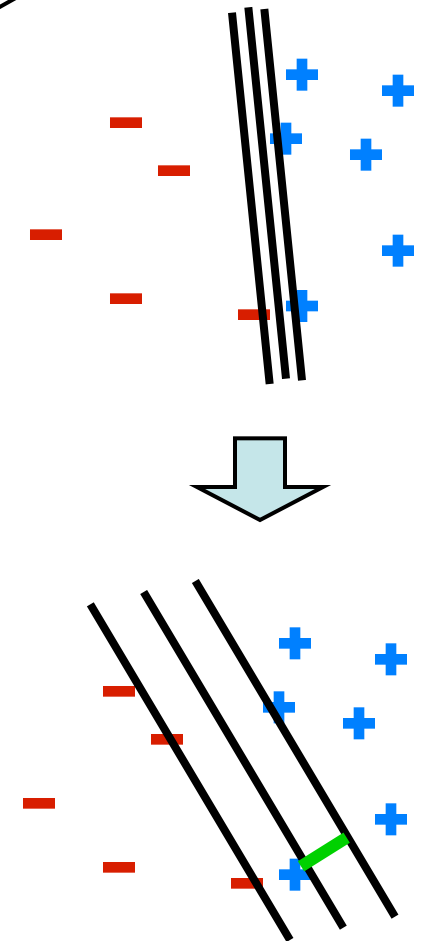
$$\min_{\mathbf{w}, \xi} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_i \xi_i$$

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

- C is called the *capacity* of the SVM – the smoothing knob

## ■ Learning:

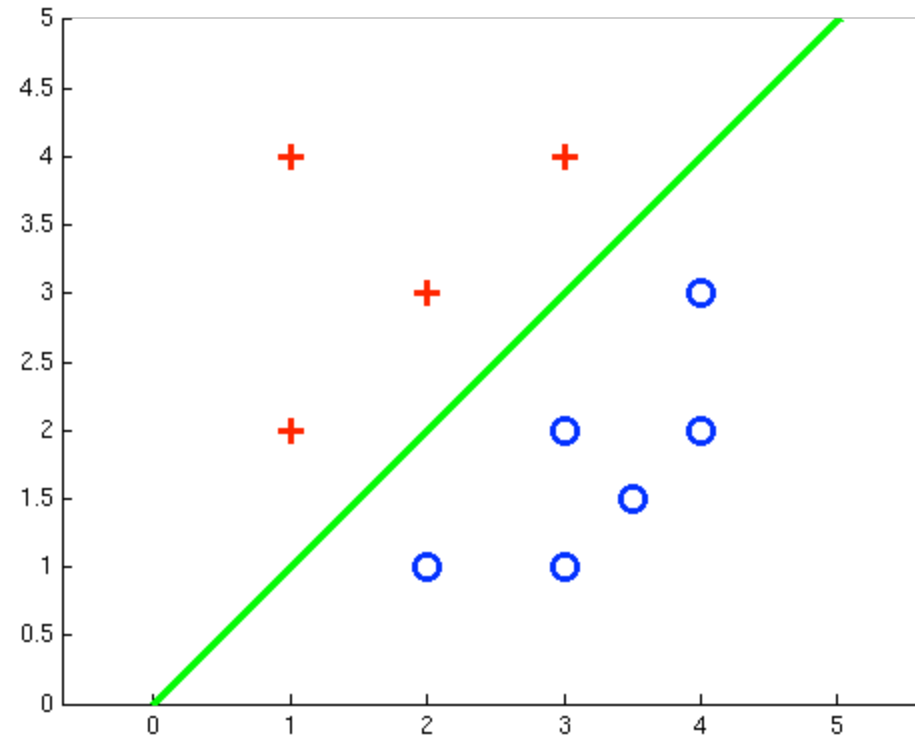
- Can still stick this into Matlab if you want
- Constrained optimization is hard; better methods!
- We'll come back to this later





# Maximum Margin

---



Likelihood





# 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  
← Normalize

- Maximize the (log) conditional likelihood of training data

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



# Maximum Entropy II

---

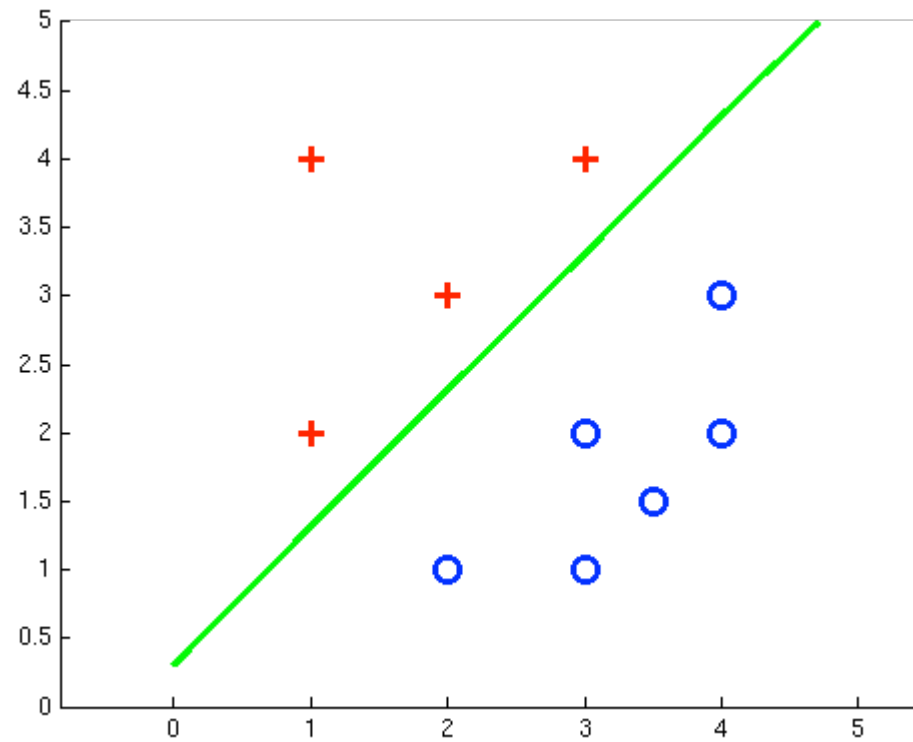
- Motivation for maximum entropy:
  - Connection to maximum entropy principle (sort of)
  - Might want to do a good job of being uncertain on noisy cases...
  - ... in practice, though, posteriors are pretty peaked
- Regularization (smoothing)

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



# Maximum Entropy

---



# Loss Comparison



# Log-Loss

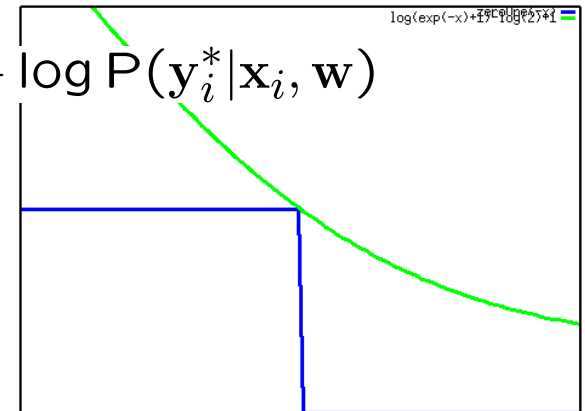
- If we view maxent as a minimization problem:

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

- This minimizes the “log loss” on each example

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

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



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



# 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  $\xi_i$

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

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

- Giving

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



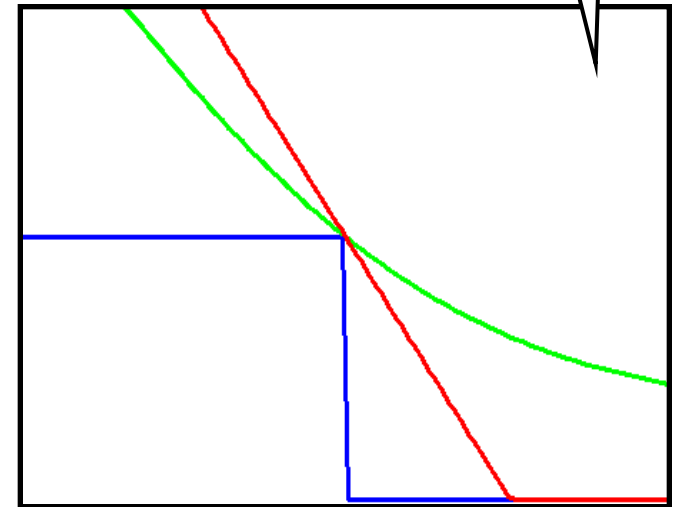
# Hinge Loss

Plot really only right in binary case

- Consider the per-instance objective:

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

- 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)



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



# Max vs “Soft-Max” Margin

- SVMs:

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

You can make this zero

- Maxent:

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

... but not this one

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





# Loss Functions: Comparison

- Zero-One Loss

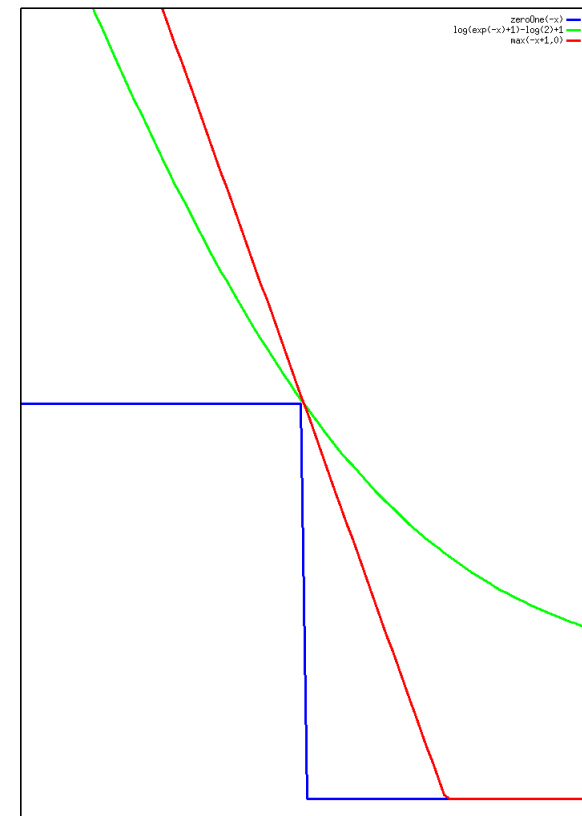
$$\sum_i \text{step} \left( \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}_i^*) - \max_{\mathbf{y} \neq \mathbf{y}_i^*} \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}) \right)$$

- Hinge

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

- Log

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

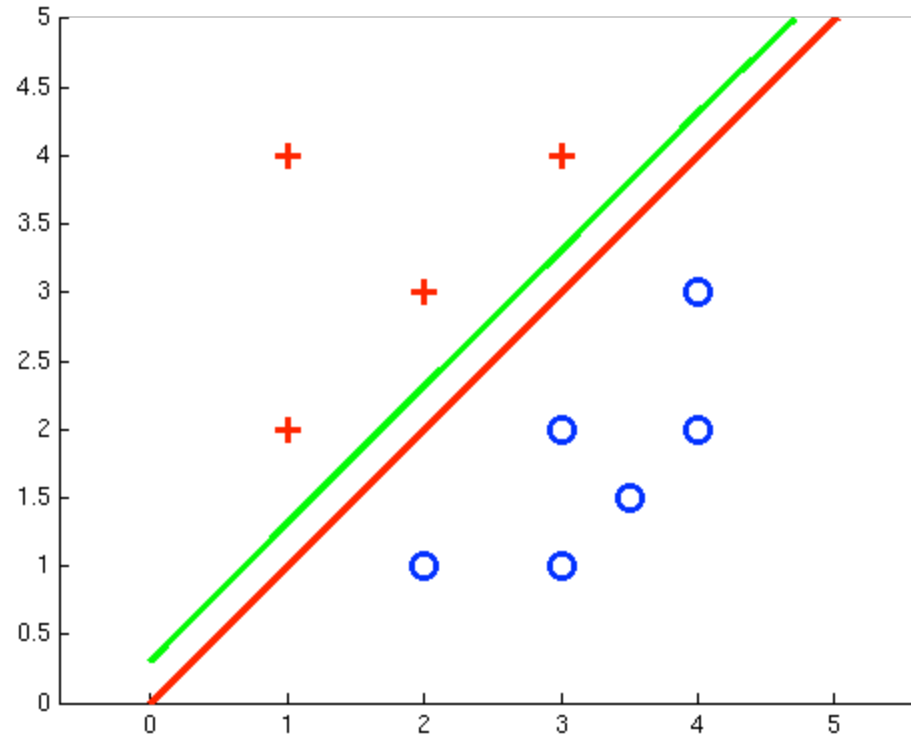


$$\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}_i^*) - \max_{\mathbf{y} \neq \mathbf{y}_i^*} \left( \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}) \right)$$



# Separators: Comparison

---

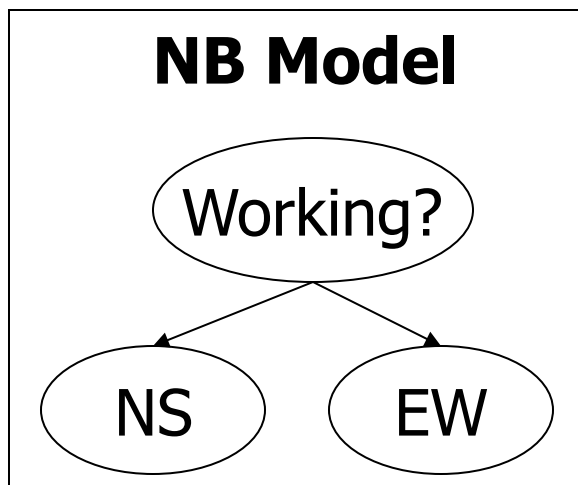
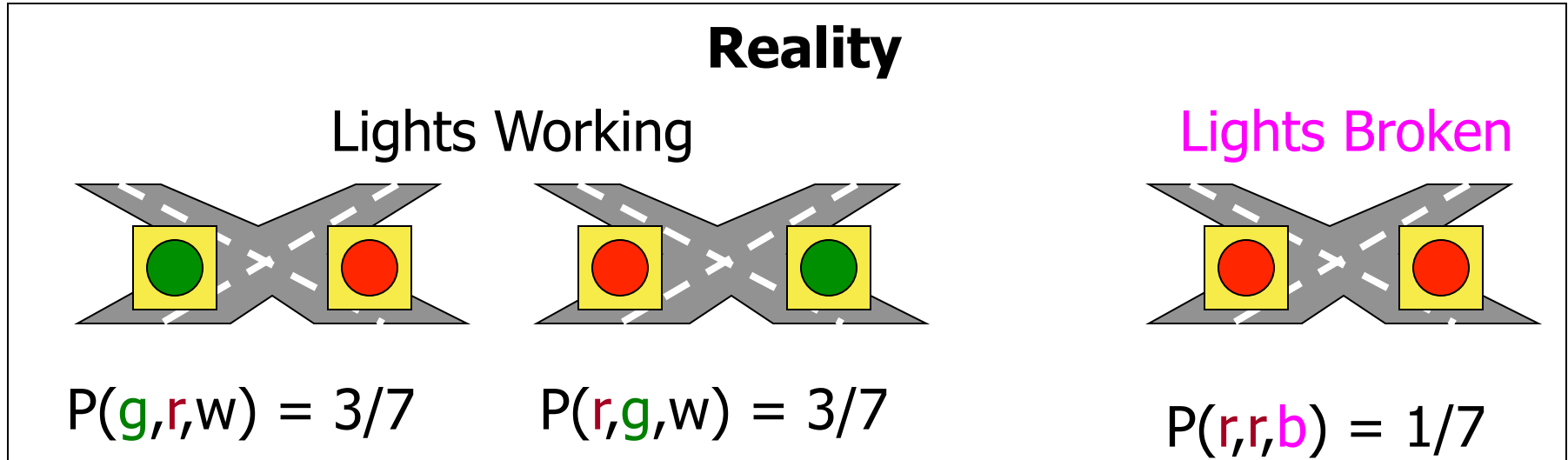


# Conditional vs Joint Likelihood





# Example: Stoplights



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

# 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:

$$1 = \langle 0.0 \ 0.0 \ 0.3 \ 0.8 \ 0.7 \ 0.1 \ \dots 0.0 \rangle$$

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

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

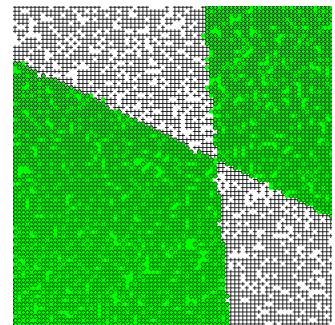
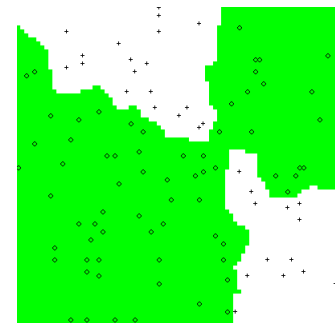
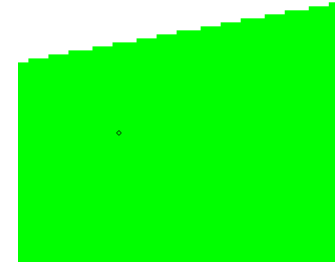




# Non-Parametric Classification

---

- 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

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

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

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

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



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



$$\mathbf{w} \leftarrow \mathbf{w} + \boxed{\Delta_i(\hat{\mathbf{y}})} \quad \textit{mistake vectors}$$



# Perceptron Weights

- What is the final value of  $w$ ?

$$w \leftarrow w + \Delta_i(y)$$

- Can it be an arbitrary real vector?
- No! It's built by adding up feature vectors (mistake vectors).

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

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

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

$$\alpha_i = \langle \alpha_i(y_1) \quad \alpha_i(y_2) \quad \dots \quad \alpha_i(y_n) \rangle$$



# 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 ( $\alpha$ )
- For each instance  $\mathbf{x}$ 
  - Try to classify

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

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

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

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

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



# Dual / Kernelized Perceptron

- How to classify an example  $x$ ?

$$\begin{aligned} \text{score}(\mathbf{y}) &= \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}) = \left( \sum_{i', \mathbf{y}'} \alpha_{i'}(\mathbf{y}') \Delta_{i'}(\mathbf{y}') \right)^\top \mathbf{f}_i(\mathbf{y}) \\ &= \sum_{i', \mathbf{y}'} \alpha_{i'}(\mathbf{y}') \left( \Delta_{i'}(\mathbf{y}')^\top \mathbf{f}_i(\mathbf{y}) \right) \\ &= \sum_{i', \mathbf{y}'} \alpha_{i'}(\mathbf{y}') \left( \mathbf{f}_{i'}(\mathbf{y}_{i'}^*)^\top \mathbf{f}_i(\mathbf{y}) - \mathbf{f}_{i'}(\mathbf{y}')^\top \mathbf{f}_i(\mathbf{y}) \right) \\ &= \sum_{i', \mathbf{y}'} \alpha_{i'}(\mathbf{y}') \left( K(\mathbf{y}_{i'}^*, \mathbf{y}) - K(\mathbf{y}', \mathbf{y}) \right) \end{aligned}$$

- 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:

$$\begin{aligned} K(x, x') &= (x \cdot x' + 1)^2 \\ &= \sum_{i,j} x_i x_j x'_i x'_j + 2 \sum_i x_i x'_i + 1 \end{aligned}$$

- RBF: infinite dimensional representation

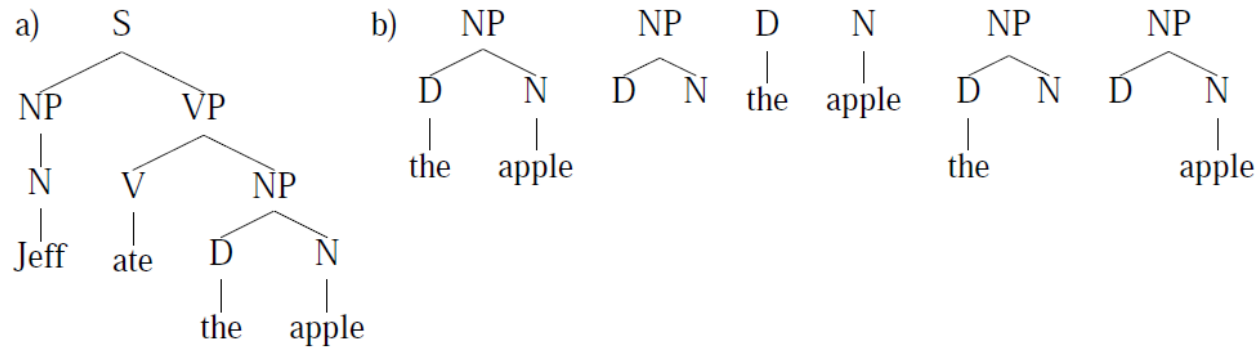
$$K(x, x') = \exp(-\|x - x'\|^2)$$

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



# Tree Kernels

[Collins and  
Duffy 01]



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

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

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

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



# Dual Formulation for SVMs

---

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

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

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

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

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



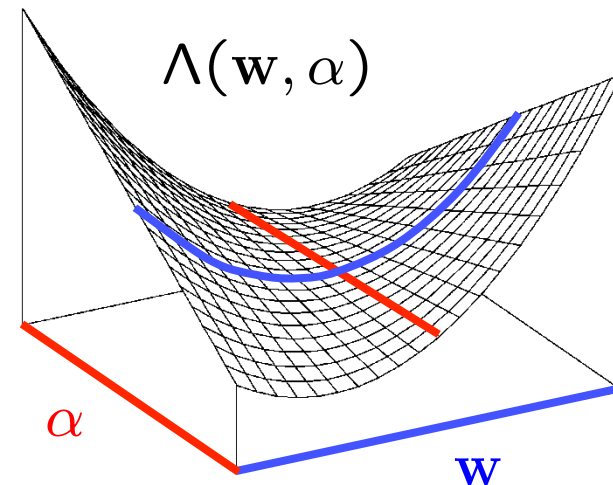
# Lagrange Duality

- We start out with a constrained optimization problem:

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

- We form the Lagrangian:

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



- This is useful because the constrained solution is a saddle point of  $\Lambda$  (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

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

has the same value as

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

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



# Dual Formulation III

- Minimize the Lagrangian for fixed  $\alpha$ 's:

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

$$\left[ \begin{array}{l} \frac{\partial \Lambda(\mathbf{w}, \alpha)}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i, \mathbf{y}} \alpha_i(\mathbf{y}) (\mathbf{f}_i(\mathbf{y}_i^*) - \mathbf{f}_i(\mathbf{y})) \\ \frac{\partial \Lambda(\mathbf{w}, \alpha)}{\partial \mathbf{w}} = 0 \end{array} \right. \Rightarrow \mathbf{w} = \sum_{i, \mathbf{y}} \alpha_i(\mathbf{y}) (\mathbf{f}_i(\mathbf{y}_i^*) - \mathbf{f}_i(\mathbf{y}))$$

- So we have the Lagrangian as a function of only  $\alpha$ 's:

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



# Back to Learning SVMs

---

- We want to find  $\alpha$  which minimize

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

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

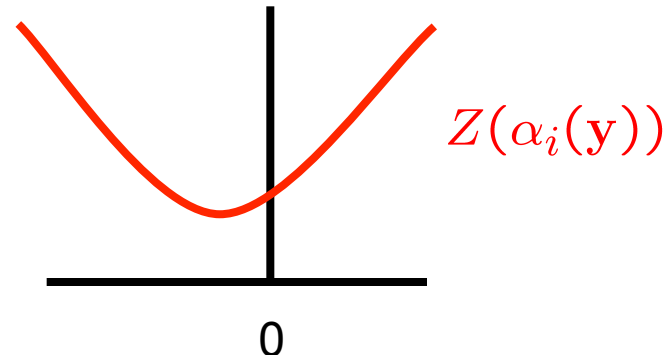
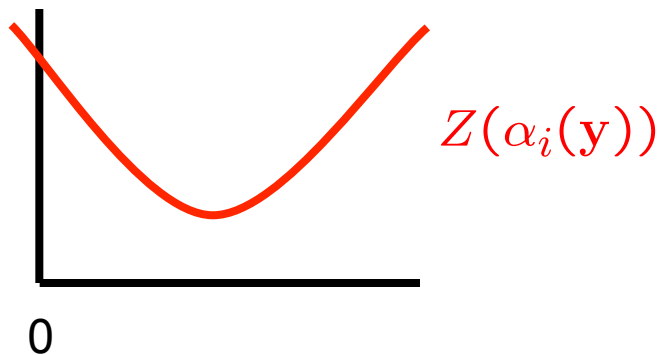
- This is a quadratic program:
  - Can be solved with general QP or convex optimizers
  - But they don't scale well to large problems
  - Cf. maxent models work fine with general optimizers (e.g. CG, L-BFGS)
- How would a special purpose optimizer work?



# Coordinate Descent I

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

- 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

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

- 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

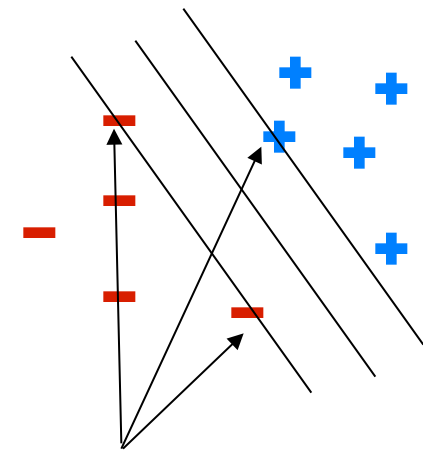
$$\min_{\mathbf{w}, \xi} \quad \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_i \xi_i$$

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

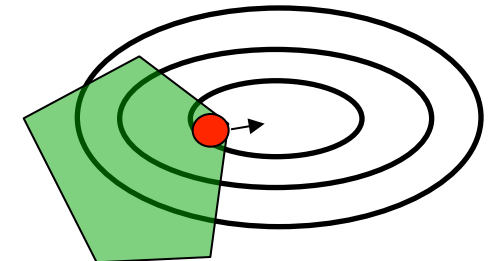
- In the solution, an  $\alpha_i(\mathbf{y})$  will be:
  - Zero if that constraint is inactive
  - Positive if that constraint is active
  - i.e. positive on the support vectors

- Support vectors contribute to weights:

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



*Support vectors*



# Structure

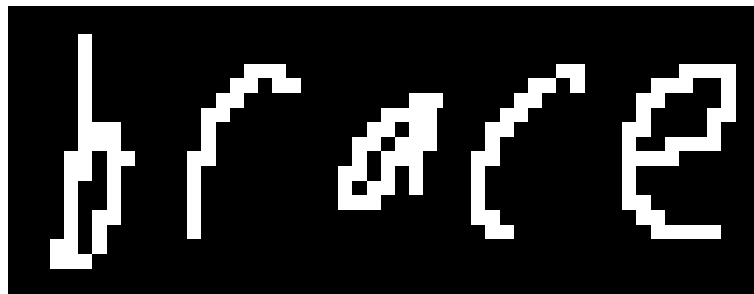


# Handwriting recognition

---

**x**

**y**



**brace**

Sequential structure



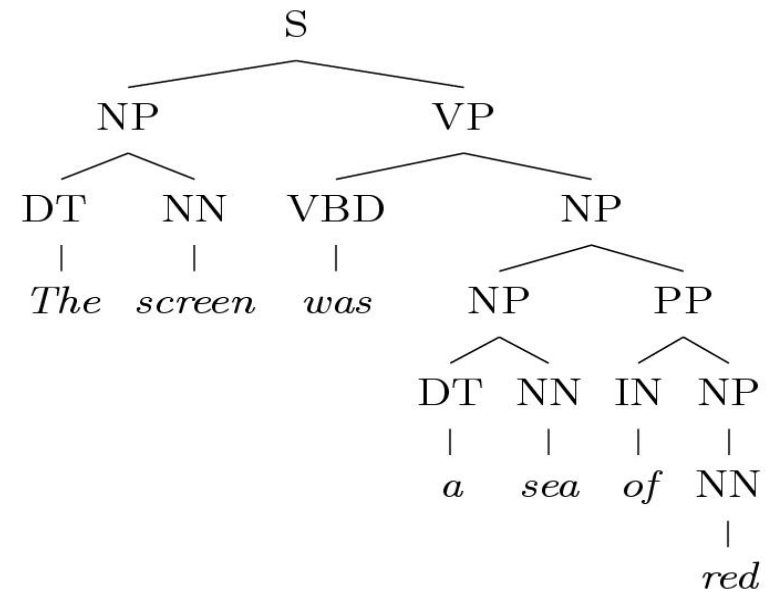
# CFG Parsing

**x**

The screen was  
a sea of red



**y**



Recursive structure



# Bilingual word alignment

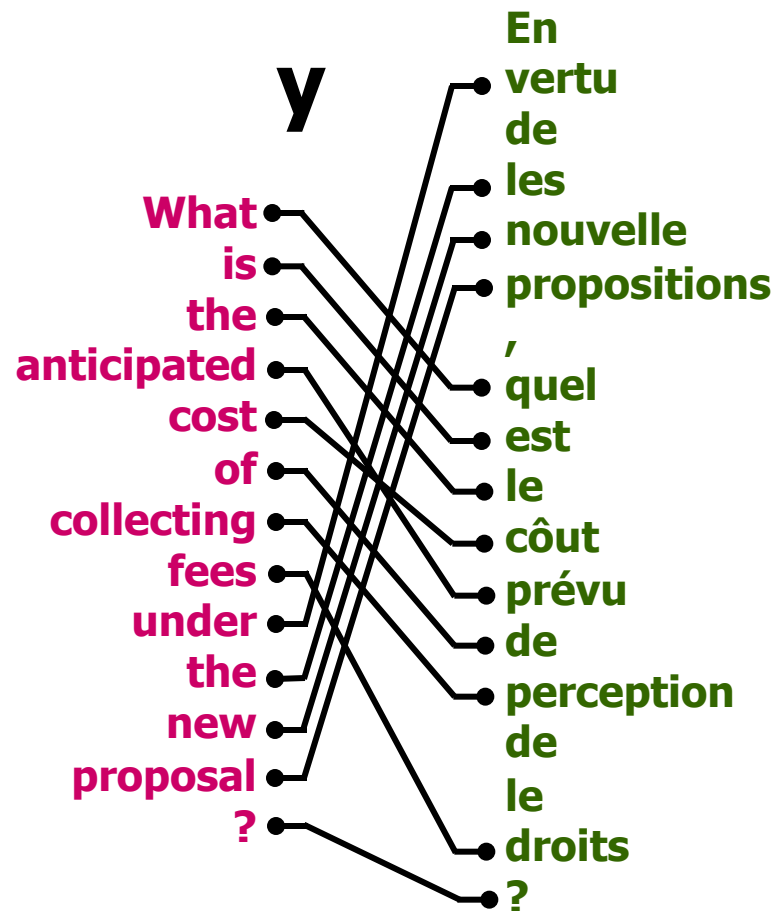
**X**

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

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



**Y**



Combinatorial structure



# Structured Models

---

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

↑  
space of feasible outputs

Assumption:

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

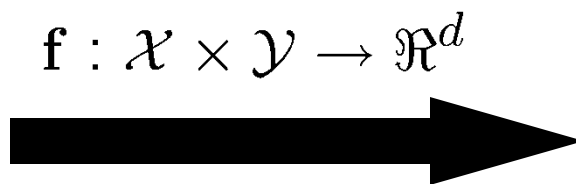
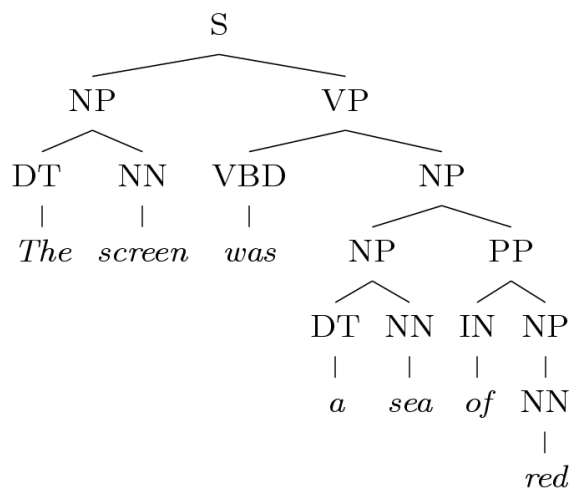
Score is a sum of local “part” scores

Parts = nodes, edges, productions



# CFG Parsing

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



$$f : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}^d$$

#(NP → DT NN)

...

#(PP → IN NP)

...

#(NN → 'sea')

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





# Bilingual word alignment

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

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

j

$y_{jk}$

k

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

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

- association
- position
- orthography



# Option 0: Reranking

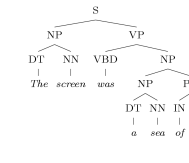
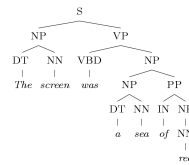
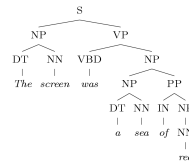
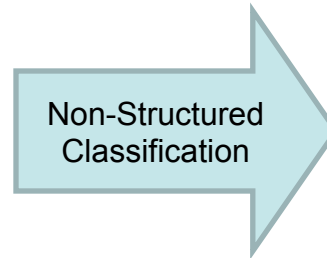
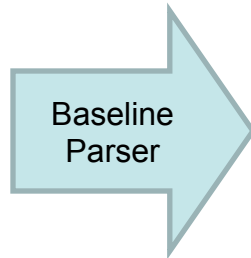
[e.g. Charniak and Johnson 05]

Input

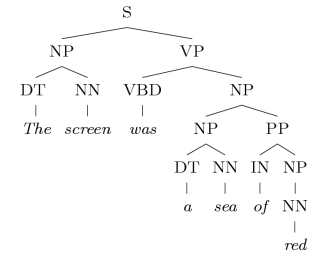
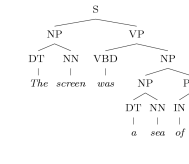
N-Best List  
(e.g. n=100)

Output

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



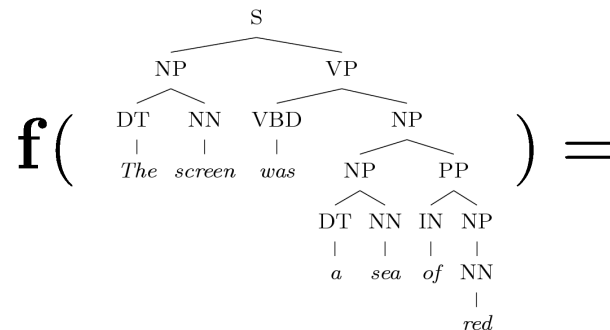
⋮





# Reranking

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



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



# Efficient Primal Decoding

---

- Common case: you have a black box which computes

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

at least approximately, and you want to learn  $w$

- Many learning methods require more (expectations, dual representations, k-best lists), but the most commonly used options do not
- Easiest option is the structured perceptron [Collins 01]
  - Structure enters here in that the search for the best  $y$  is typically a combinatorial algorithm (dynamic programming, matchings, ILPs, A\*...)
  - Prediction is structured, learning update is not



# Structured Margin

---

- Remember the margin objective:

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

- This is still defined, but lots of constraints



# Full Margin: OCR

- We want:

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

- Equivalently:

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

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

...

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

**a lot!**



# Parsing example

- We want:

$$\arg \max_y w^\top f(\text{'It was red'}, y) = \begin{matrix} S \\ \swarrow \searrow \\ A \quad B \\ \swarrow \searrow \\ C \quad D \end{matrix}$$

- Equivalently:

$$w^\top f(\text{'It was red'}, \begin{matrix} S \\ \swarrow \searrow \\ A \quad B \\ \swarrow \searrow \\ C \quad D \end{matrix}) > w^\top f(\text{'It was red'}, \begin{matrix} S \\ \swarrow \searrow \\ A \quad B \\ \swarrow \searrow \\ D \quad F \end{matrix})$$

$$w^\top f(\text{'It was red'}, \begin{matrix} S \\ \swarrow \searrow \\ A \quad B \\ \swarrow \searrow \\ C \quad D \end{matrix}) > w^\top f(\text{'It was red'}, \begin{matrix} S \\ \swarrow \searrow \\ C \quad A \\ \swarrow \searrow \\ B \quad D \end{matrix})$$

...

$$w^\top f(\text{'It was red'}, \begin{matrix} S \\ \swarrow \searrow \\ A \quad B \\ \swarrow \searrow \\ C \quad D \end{matrix}) > w^\top f(\text{'It was red'}, \begin{matrix} S \\ \swarrow \searrow \\ G \quad E \\ \swarrow \searrow \\ F \quad H \end{matrix})$$

} a lot!



# Alignment example

- We want:

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

- Equivalently:

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

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

...

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

} 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:

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

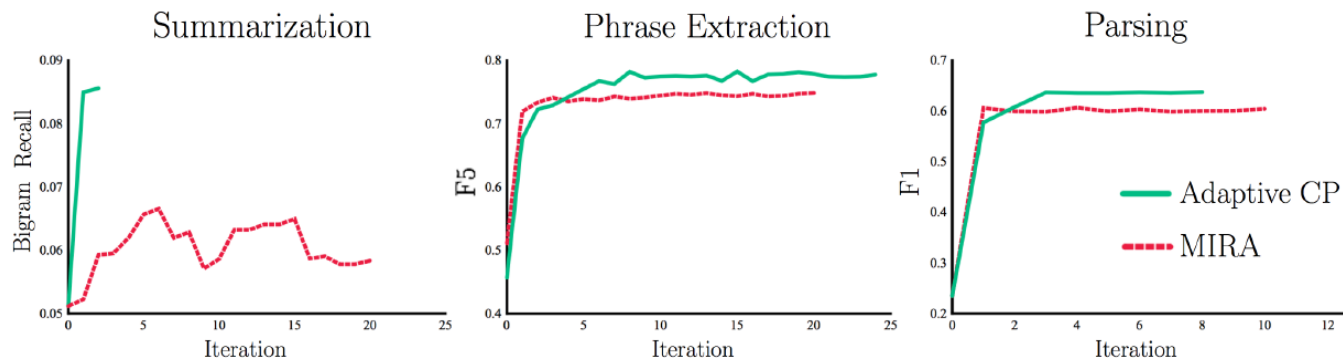
$$\arg \max_{\mathbf{y}} \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}) + \ell_i(\mathbf{y})$$

- 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  $P(\text{DT-NN}|\text{sentence})$  for each position and sum
- Also works with latent variables (more later)