Introduction to Classification:
Likelihoods, Margins, Features, and Kernels

Dan Klein
UC Berkeley
nlp.cs.berkeley.edu

Acknowledgements

- Many slides adapted from previous tutorials with Christopher Manning and Ben Taskar
- Includes several diagrams used or adapted from Ray Mooney, Andrew Moore, Mike Collins

Introduction

- Much of NLP can be seen as making decisions
  - About structured analyses (sequences, trees, graphs)
  - On the basis of multiple information sources (words, word classes, tree configurations, etc)
- Widespread adoption of discriminative methods
  - Use of arbitrary features
  - Various formulations: maxent, SVM, perceptron
  - Common use: local discriminative decisions, possibly chained
  - Newer: global methods which exploit model structure (CRFs, max-margin networks)
- This tutorial will cover the core ideas behind:
  - Part I: Basic Linear Classification
  - Part II: Kernels and Structure
- Non-goals: not an overview of NLP applications, not at all complete coverage of the huge literature on classification!

Outline

- Part I: Basic Linear Classification
  - Multiclass linear decision rules
  - Approaches: perceptron, maximum likelihood, maximum margin
  - Advantages / disadvantages / tradeoffs
- Part II: Kernels and Structure
  - Kernels and kernelization of classifiers
  - Basic structured classification

Example: Text Classification

- We want to classify documents into categories

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

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

Some Definitions

Sometimes, we want Y to depend on x

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

SPORTS ∧ "*win*" *POLITICS* ∧ "*election*"

POLITICS ∧ "*win*"

Either x is implicit, or y contains x

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* …

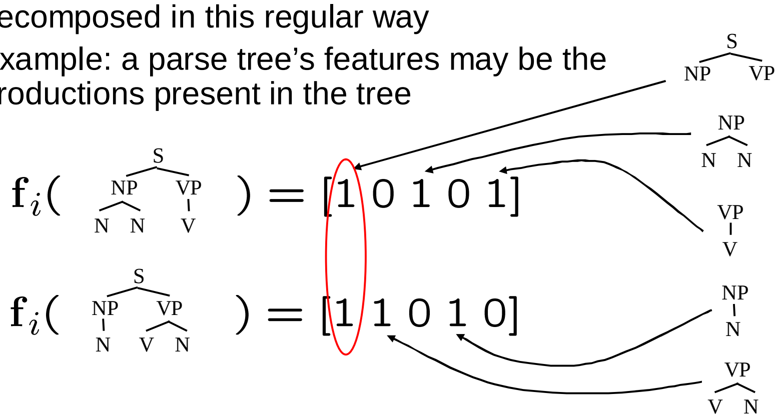
$$\text{“}\mathbf{f}_i(\mathbf{x})\text{”} \quad [1\ 0\ 1\ 0\ ]$$

“*win*” “*election*”

$$\mathbf{f}_i(\mathit{SPORTS}) = [1\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$$
$$\mathbf{f}_i(\mathit{POLITICS}) = [0\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 0]$$
$$\mathbf{f}_i(\mathit{OTHER}) = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0]$$

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



$$\mathbf{f}_i(\ \text{[tree: S → NP VP; NP → N N; VP → V]}\ ) = [1\ 0\ 1\ 0\ 1]$$
$$\mathbf{f}_i(\ \text{[tree: S → NP VP; NP → N; VP → V N]}\ ) = [1\ 1\ 0\ 1\ 0]$$

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

Linear Models: Scoring

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

$$\mathbf{f}_i(POLITICS) = [\ 0\ \ 0\ \ 0\ \ 0\ \ 1\ \ 0\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0]$$
$$\mathbf{f}_i(SPORTS) = [\ 1\ \ 0\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0]$$
$$\mathbf{w} = [\ 1\ \ 1\ \ {-1}\ \ {-2}\ \ 1\ \ {-1}\ \ 1\ \ {-2}\ \ {-2}\ \ {-1}\ \ {-1}\ \ 1]$$

- We compare hypotheses on the basis of their linear scores:

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

$$\mathbf{f}_i(POLITICS) = [\ 0\ \ 0\ \ 0\ \ 0\ \ 1\ \ 0\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0]$$
$$\mathbf{w} = [\ 1\ \ 1\ \ {-1}\ \ {-2}\ \ 1\ \ {-1}\ \ 1\ \ {-2}\ \ {-2}\ \ {-1}\ \ {-1}\ \ 1]$$

$$score(\mathbf{x}^i, POLITICS, \mathbf{w}) = 1 \times 1 + 1 \times 1 = 2$$

Linear Models: Prediction Rule

- The linear prediction rule:

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

$$score(\mathbf{x}^i, SPORTS, \mathbf{w}) = 1 \times 1 + (-1) \times 1 = 0$$
$$score(\mathbf{x}^i, POLITICS, \mathbf{w}) = 1 \times 1 + 1 \times 1 = 2$$
$$score(\mathbf{x}^i, OTHER, \mathbf{w}) = (-2) \times 1 + (-1) \times 1 = -3$$

$$\Downarrow$$

$$prediction(\mathbf{x}^i, \mathbf{w}) = POLITICS$$

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

Binary Decision Rule

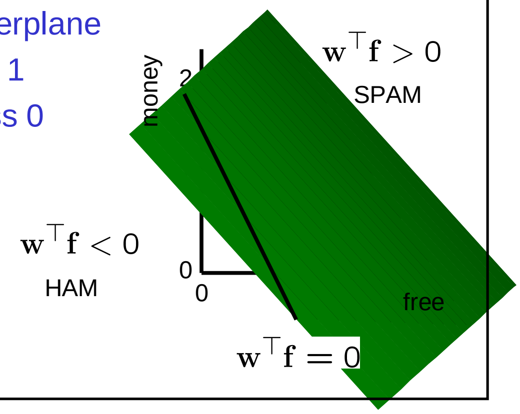
- Heavily studied case: binary classification
  - Simplifed: only class “1” has features

$$prediction(\mathbf{x}^i, \mathbf{w}) = (\mathbf{w}^\top \mathbf{f}_i > 0)$$

  - Decision rule is a hyperplane
  - One side will be class 1
  - Other side will be class 0

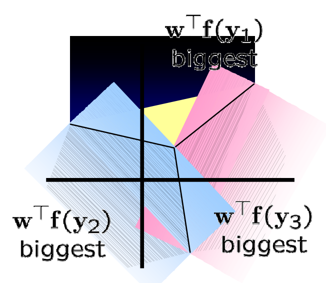
w

```
BIAS  : -3
free  :  4
money :  2
the   :  0
...
```



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

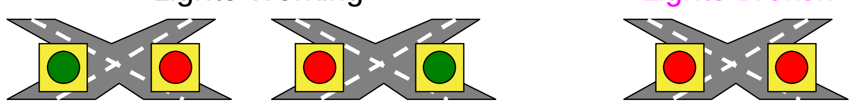
Learning Classifier Weights

- 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

Example: Stoplights

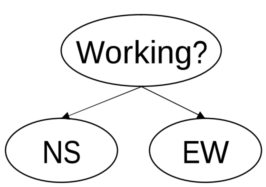
Reality

Lights Working | Lights Broken



P(g,r,w) = 3/7   P(r,g,w) = 3/7   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!
- Non-generative values can give better classification

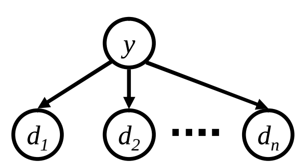
Linear Models: Naïve-Bayes

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

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

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

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



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

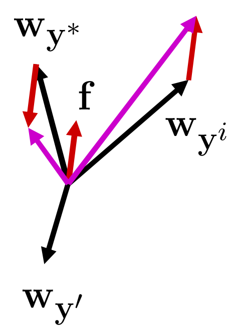
Linear Models: Perceptron

- 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
  - Visit training instances one by one
    - Try to classify

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

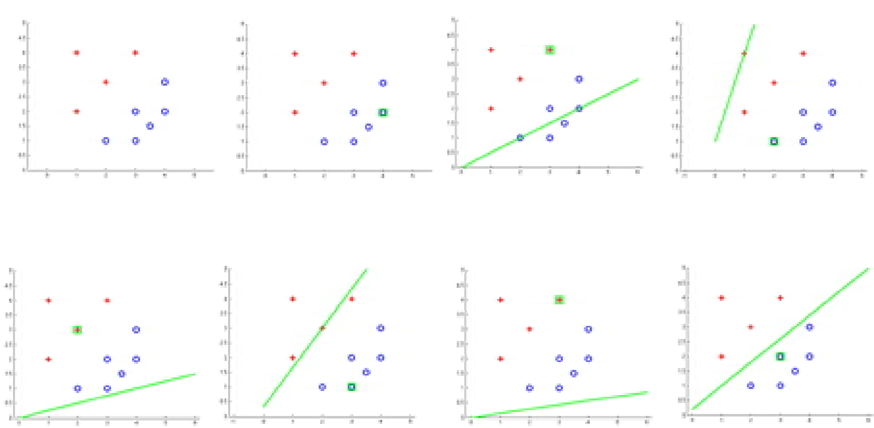
    - If correct, no change!
    - If wrong: adjust weights

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



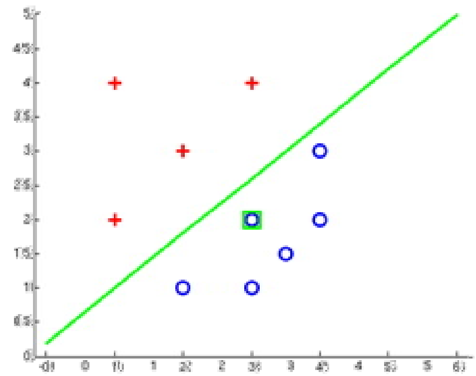
Examples: Perceptron

- Separable Case



Examples: Perceptron

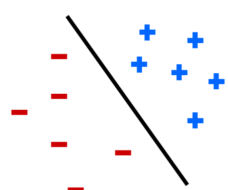
- Separable Case



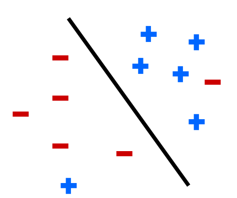
Perceptrons and Separability

- 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

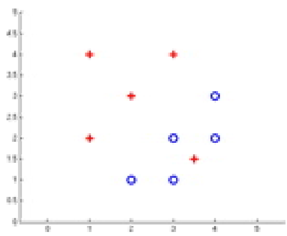


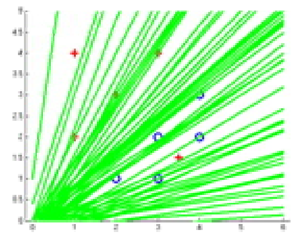
Non-Separable

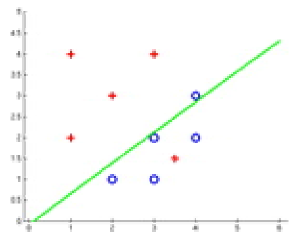


Examples: Perceptron

- Non-Separable Case

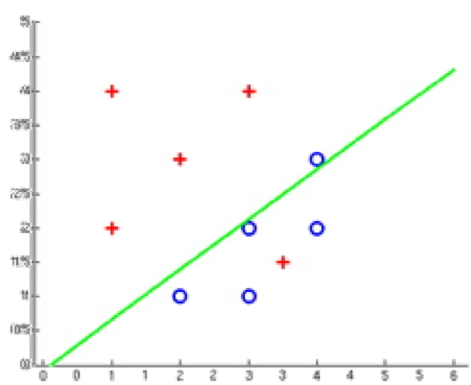






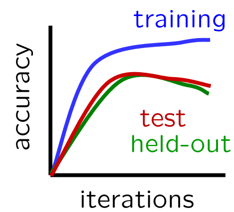
Examples: Perceptron

- Non-Separable Case

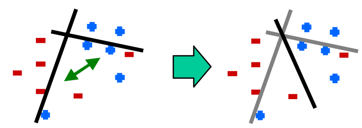


Issues with Perceptrons

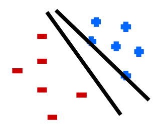
- Overtraining: test / held-out accuracy usually rises, then falls
  - Overtraining isn't quite as bad as overfitting, but is similar



- 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

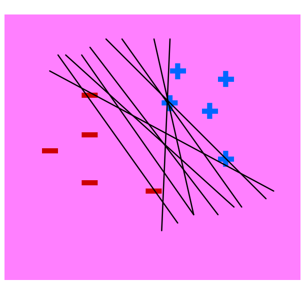


Problems with Perceptrons

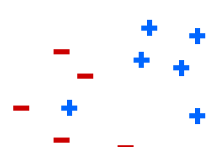
- Perceptron "goal": separate the training data

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

1. This may be an entire feasible space

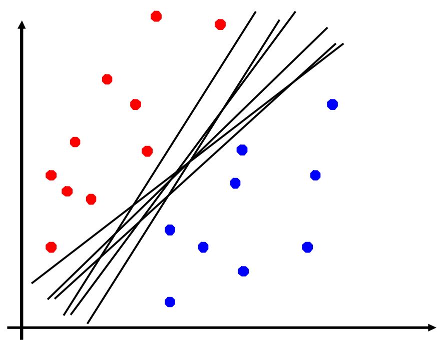


2. Or it may be impossible



Linear Separators

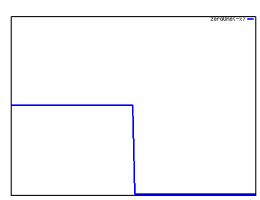
- Which of these linear separators is optimal?



Objective Functions

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

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

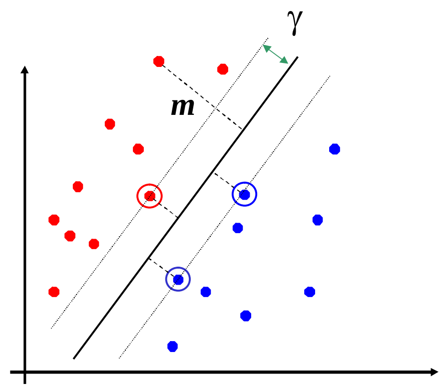


$$\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}^i) - \max_{\mathbf{y} \neq \mathbf{y}^i} \mathbf{w}^\top \mathbf{f}_i(\mathbf{y})$$

  - 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

Classification Margin (Binary)

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



Classification Margin

- For each example $\mathbf{x}_i$ and possible mistaken candidate $\mathbf{y}$, we avoid that mistake by a margin $\boldsymbol{m}_i(\mathbf{y})$ (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 $\boldsymbol{m}$

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

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

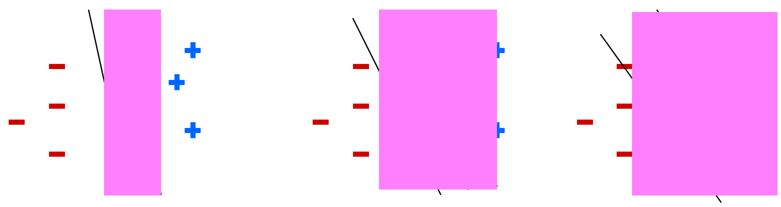
$$\forall i, \forall \mathbf{y} \quad \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}^i) \geq \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}) + \gamma \ell_i(\mathbf{y})$$

Maximum Margin

- Separable SVMs: find the max-margin w

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

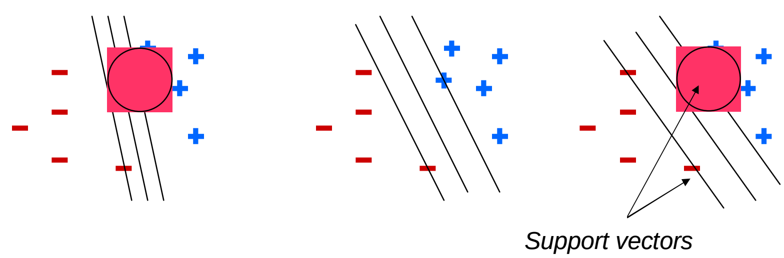
$$\forall i, \forall \mathbf{y} \quad \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}^i) \geq \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}) + \gamma \ell_i(\mathbf{y})$$



  - 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



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 $\gamma = 1$

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

Gamma to w

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

$$\mathbf{w} = \gamma u$$

$$\gamma = 1/||u||$$

$$\max_{||\gamma u||=1} 1/||u||^2$$
$$\forall i, \mathbf{y} \quad \gamma u^\top \mathbf{f}_i(\mathbf{y}^i) \geq \gamma u^\top \mathbf{f}_i(\mathbf{y}) + \gamma \ell_i(\mathbf{y})$$

$$\max_{||\gamma u||=1} 1/||u||^2$$
$$\forall i, \mathbf{y} \quad u^\top \mathbf{f}_i(\mathbf{y}^i) \geq u^\top \mathbf{f}_i(\mathbf{y}) + \ell_i(\mathbf{y})$$

$$\min_{||\gamma u||=1} ||u||^2$$
$$\forall i, \mathbf{y} \quad u^\top \mathbf{f}_i(\mathbf{y}^i) \geq u^\top \mathbf{f}_i(\mathbf{y}) + \ell_i(\mathbf{y})$$

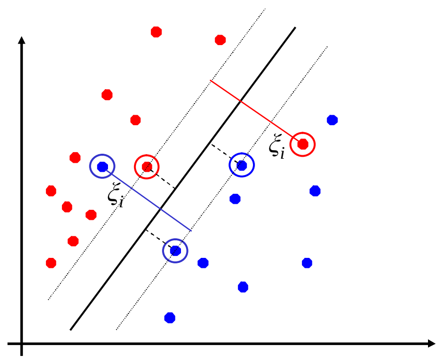
$$\min_{u} ||u||^2$$
$$\forall i, \mathbf{y} \quad u^\top \mathbf{f}_i(\mathbf{y}^i) \geq u^\top \mathbf{f}_i(\mathbf{y}) + \ell_i(\mathbf{y})$$

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

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

Soft Margin Classification

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



Maximum Margin

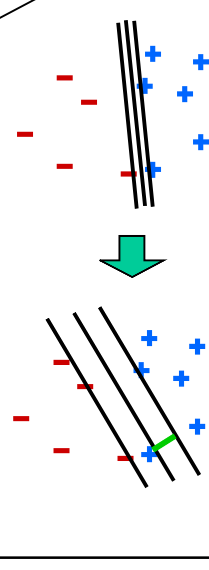
Note: exist other choices of how to penalize slacks!

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

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

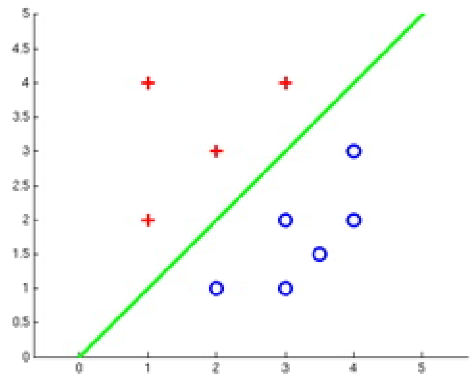
$$\forall i, \mathbf{y}, \quad \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}^i) + \xi_i \geq \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}) + \ell_i(\mathbf{y})$$

  - 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

Really, we should all stop calling this maximum entropy – it's multiclass logistic regression or a maximum likelihood log-linear model…

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

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

Make positive (numerator)

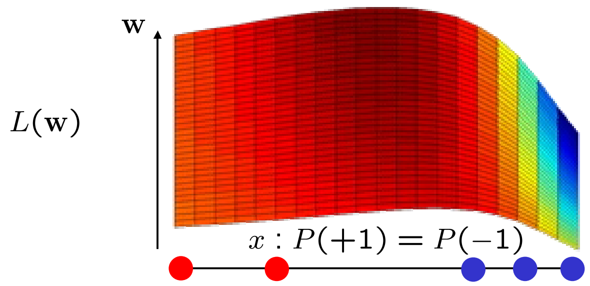
Normalize (denominator)

  - Maximize the (log) conditional likelihood of training data

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

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

Maximum Entropy Separators



$$\mathbf{f}(+1) = [1 \quad x \quad 0 \quad 0]$$
$$\mathbf{f}(-1) = [0 \quad 0 \quad 1 \quad x]$$
$$\mathbf{w} = [a \quad b \quad -a \quad -b]$$

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

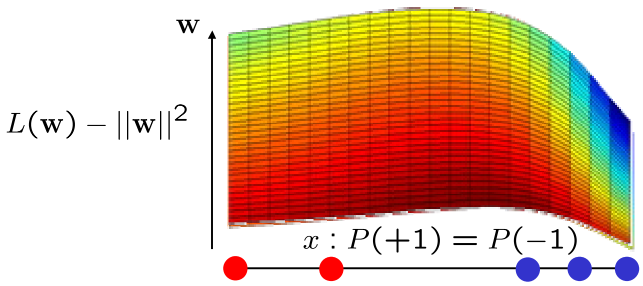
Maximum Entropy II

- 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)

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

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

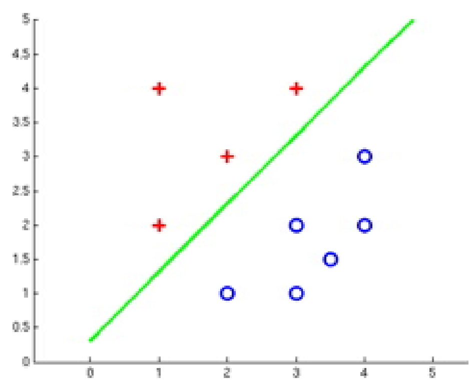
Maximum Entropy Separators



$$\mathbf{f}(+1) = [1 \quad x \quad 0 \quad 0]$$
$$\mathbf{f}(-1) = [0 \quad 0 \quad 1 \quad x]$$
$$\mathbf{w} = [a \quad b \quad -a \quad -b]$$

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

Maximum Entropy



Log-Loss

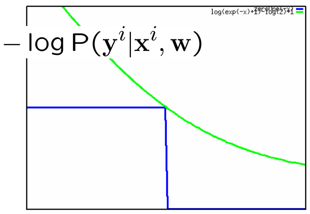
- If we view maxent as a minimization problem:

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

- This minimizes the “log loss” on each example

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

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

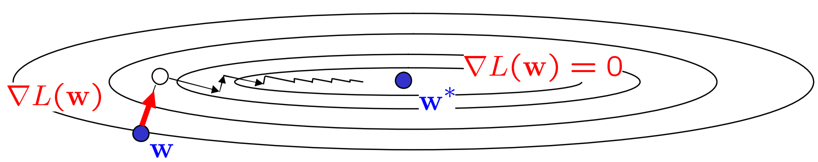


- One view: log loss is an *upper bound* on zero-one loss

Unconstrained Optimization

- The maxent objective is an unconstrained optimization problem

$L(\mathbf{w})$



- 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

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

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

Big weights are bad

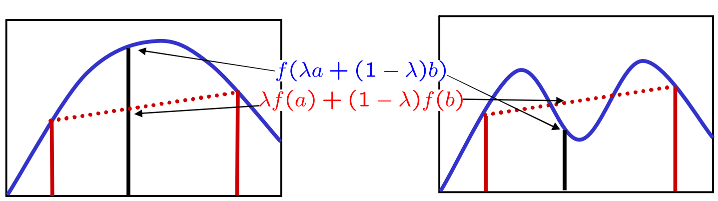
Expected count of feature n in predicted candidates

Total count of feature n in correct candidates

Convexity

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

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

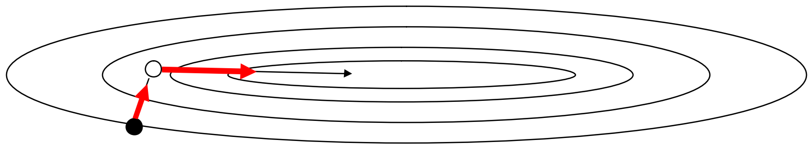


Convex Non-Convex

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

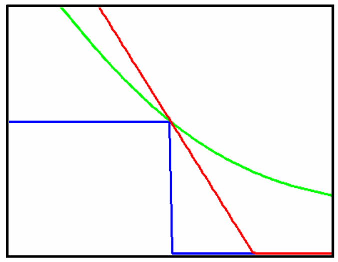
$$\min_{\mathbf{w}} \frac{1}{2}||\mathbf{w}||^2 - C\sum_i \left(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}^i) - \max_{\mathbf{y}} \left(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}) + \ell_i(\mathbf{y})\right)\right)$$

Hinge Loss

- Consider the per-instance objective:

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

- 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



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

Plot really only right in binary case

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

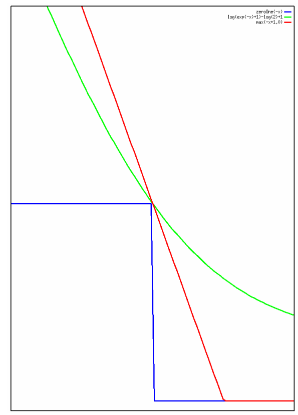
$$\sum_i step\left(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}^i) - \max_{\mathbf{y} \neq \mathbf{y}^i} \mathbf{w}^\top \mathbf{f}_i(\mathbf{y})\right)$$

- Hinge

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

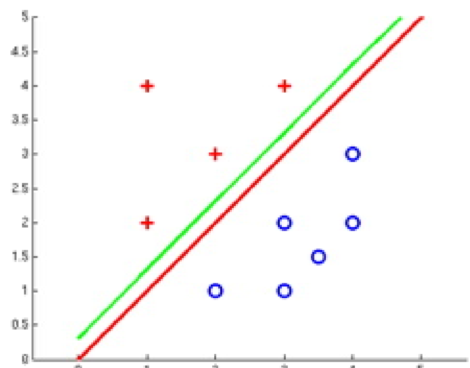
- Log

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



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

Separators: Comparison



Status Check

- We've covered:
  - Basics
  - What the perceptron does and how to train it
  - The max margin objective (but not how to optimize it)
  - The maximum entropy objective and how to optimize it
- Next:
  - "Dual classification" with perceptrons
  - Dual optimization, how to optimize SVMs
  - Kernel methods
  - Structured classification

Part II

- Kernels
  - Dual algorithms
  - Kernels and kernelization
- Structured classification
  - Structured inputs
  - Structured learning

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

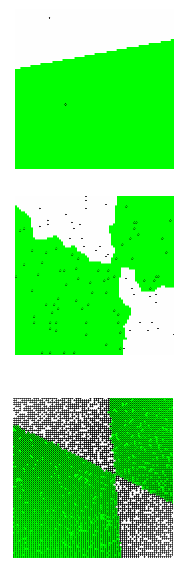
$$\text{sim}(x, y) = x^\top y = \sum_i x_i y_i$$

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…

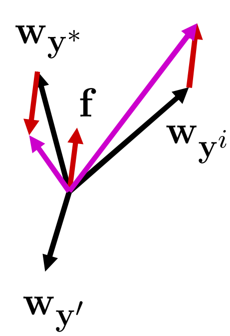
- Nearest neighbor-like approaches
  - Work with data through similarity functions
  - No explicit “learning”
- Linear approaches
  - Explicit training to reduce empirical error
  - Represent data through features
- Kernelized linear models
  - Explicit training, but driven by similarity!
  - Flexible,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

The Perceptron, Again

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

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

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



$$\mathbf{w} \leftarrow \mathbf{w} + \mathbf{f}_i(\mathbf{y}^i)$$
$$\mathbf{w} \leftarrow \mathbf{w} - \mathbf{f}_i(\mathbf{y}^*)$$
$$\mathbf{w} \leftarrow \mathbf{w} + \left(\mathbf{f}_i(\mathbf{y}^i) - \mathbf{f}_i(\mathbf{y}^*)\right)$$
$$\mathbf{w} \leftarrow \mathbf{w} + \boxed{\Delta_i(\mathbf{y})}$$

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 *i*
  - Try to classify x_i,

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

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

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

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

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

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

  - 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

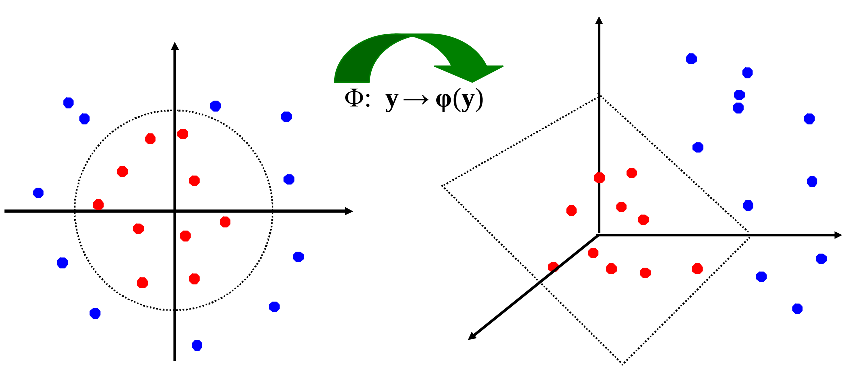
$$K(x, x') = (x \cdot x' + 1)^2$$

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

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

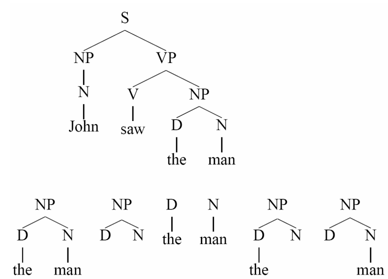
Non-Linear Separators

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



Some Structured Kernels

- PCFG Tree Kernels [Collins and Duffy 02]
  - Function of two trees
  - Measures the number of tree fragments in common (weighted by fragment size)
  - Computed by a dynamic program



- Dependency Tree Kernels [Culotta and Sorensen 04]

- Many more…

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…

Kernels vs. Similarity Functions

- Q: What does it take for a similarity function to be a kernel?

- A: It must satisfy some technical conditions:
  - Kernel matrix must be symmetric and positive semidefinite (e.g. self-similarity is high)
  - Note: making diagonal very large can sometimes suffice

Dual Formulation for SVMs

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

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

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

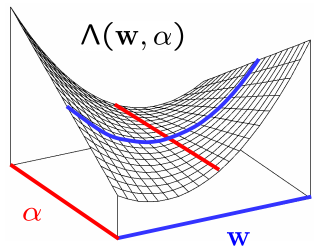
$$\min_{\mathbf{w}} \max_{\alpha \geq 0} \quad \Lambda(\mathbf{w}, \alpha) = \frac{1}{2}||\mathbf{w}||^2 - \sum_{i,\mathbf{y}} \alpha_i(\mathbf{y}) \left(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}^i) - \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}) - \ell_i(\mathbf{y})\right)$$

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

Lagrange Duality

- We start out with a constrained optimization problem:

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



- We form the Lagrangian:

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

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

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

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:

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

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

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

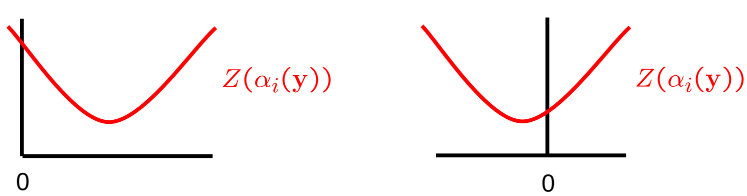
- So we have the Lagrangian as a function of only α's:

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

Coordinate Descent I

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

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



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

Coordinate Descent II

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

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

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

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

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

What are the Alphas?

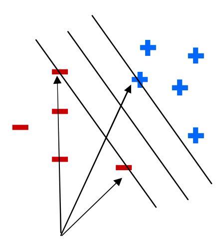
- Each candidate corresponds to a primal constraint

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

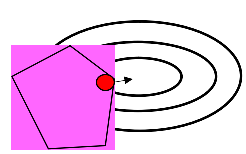
$$\forall i, \mathbf{y} \quad \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}^i) \geq \mathbf{w}^\top \mathbf{f}_i(\mathbf{y}) + \ell_i(\mathbf{y}) - \xi_i$$

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

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



Support vectors



Dual Linear Classifiers

- For SVMs and Perceptrons, we ended up with exactly the same rule:

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

- This form holds more generally (the Representer Theorem gives conditions)
  - Basically, components of the weight vector perpendicular to all training examples increase the norm of w without impacting training example scores
  - E.g. one can show that all weight vectors learned by maxent have the same form
  - So we could kernelize maxent as well…

Structured Prediction

- So far: talked about candidates y as if from a fixed set of labels
- In principle, nothing at all changes if y's have structure!
- In practice, big issues:
  - Perceptron: argmax is hard because |Y| is big

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

  - Maxent: expectations are hard because |Y| is big

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

  - Max margin: too many constraints / alphas because |Y| is big

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

Example: Parse Reranking

- If the candidate set |Y| is small, then no problem:
  - [Collins 02] Reranking with perceptron
  - [Charniak and Johnson 05] Reranking with maxent
- Start with the n-best outputs of a good base parser
  - Define tons of features on a parses y
  - No need to even make the feature local
- Learn classifier using only these candidates
  - Everything works exactly as we've discussed!

Example: Structured Perceptron

- Perceptron is nice: only need structured inference:

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

- Can do this search with a dynamic program provided the features used decompose in appropriate local ways [Collins 02]
- Only need to be able to do Viterbi search (and even approximate search can work). See also [Daume et al 06])
- Online margin methods like MIRA [Crammer and Singer 03] get some of the margin effect with similar requirements by updating minimally

Example: CRFs

- For maxent, we need feature expectations:

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

- We can sometimes calculate these expectations with, e.g., dynamic programs
- For sequences, conditional random fields (CRFs) do exactly this computations [Lafferty et al 01]
- Trees work fine as well [Johnson 01]
- *Much* other CRF work in the ACL community!
- Good property: CRFs put probabilities over candidates

Example: Structured Margin

- For maximum margin, things are harder:

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

- One option: only use constraints as you find you need them [Tsochantaridis et al 05]
- A better option: the alphas often decompose along dynamic programming structures [Taskar et al 03, 05]

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

Summary

- Basic feature-driven classification
  - Perceptron
  - Maximum entropy
  - Maximum margin
- Kernels and Structure
- Much, much more on this topic!

A VERY Few References

- Impossible to even start to list all the relevant work!
- Some texts:
  - Large list at: http://www.kernel-machines.org/books/
  - Classic: Tom Mitchell "Machine Learning," 1997.
  - Christopher Bishop, "Pattern Recognition and Machine Learning," 2007.
- Work directly cited in the tutorial:

  - Eugene Charniak and Mark Johnson, "Coarse-to-Fine n-Best Parsing and MaxEnt Discriminative Reranking," *ACL*, 2005.
  - Koby Crammer and Yoram Singer, "Ultraconservative Online Algorithms for Multiclass Problems," *Journal of Machine Learning Research*, 2003.
  - Michael Collins, "Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms," *EMNLP,* 2002.
  - Michael Collins and Nigel Duffy, "Convolution Kernels for Natural Language," *NIPS,* 2001.
  - James Henderson and Ivan Titov, "Data-Defined Kernels for Parse Reranking Derived from Probabilistic Models," *ACL*, 2005
  - Mark Johnson, "Joint and Conditional Estimation of Tagging and Parsing Models," *ACL* 2001.
  - John Lafferty, Andrew McCallum, and Fernando Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," *ICML*, 2001.
  - Ben Taskar, Dan Klein, Michael Collins, Daphne Koller and Chris Manning, "Max-Margin Parsing," *EMNLP,* 2004.
  - Ben Taskar, Carlos Guestrin and Daphne Koller "Max-Margin Markov Networks," *NIPS,* 2003.
  - Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun, "Large Margin Methods for Structured and Interdependent Output Variables," *JMLR*, 2005.