What's the Perceptron Optimizing?

Machine Learning – CSE446
Carlos Guestrin
University of Washington

May 1, 2013

The Perceptron Algorithm [Rosenblatt '58, '62]

- Classification setting: y in {-1,+1}
- Linear model
  - Prediction: $\hat{y} = \text{sign}(w \cdot x)$
- Training: $w^{(0)} = 0$ or something smarter
  - Initialize weight vector:
  - At each time step:
    - Observe features: $x^{(t)} \leftarrow$ (page, user, ad)
    - Make prediction: $\hat{y} = \text{sign}(w^{(t)} \cdot x^{(t)})$
    - Observe true class:
      $y^{(t)} \leftarrow$ true label
    - Update model:
      - If prediction is not equal to truth, if make a mistake!

if $\hat{y} = y^{(t)}$
$w^{(t+1)} \leftarrow w^{(t)}$
else
$w^{(t+1)} \leftarrow w^{(t)} + y^{(t)} x^{(t)}$

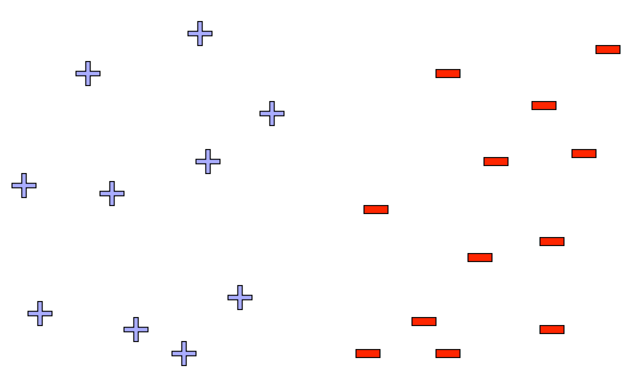
I made a mistake:
e.g. $y^{(t)} = +1$
$w^{(t)} \cdot x^{(t)} < 0$
but wanted > 0
what u $\max_u u \cdot x^{(t)}$?
$x^{(t)}$!!
by adding $x^{(t)}$ to w
I increase $w^{(t+1)} \cdot x^{(t)}$ the most

similarly when $y^{(t)} = -1$

What is the Perceptron Doing???

- When we discussed logistic regression:
  - Started from maximizing conditional log-likelihood
- When we discussed the Perceptron:
  - Started from description of an algorithm
- What is the Perceptron optimizing????

Perceptron Prediction: Margin of Confidence



Hinge Loss

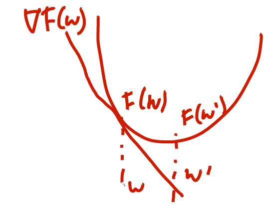
- Perceptron prediction:
- Makes a mistake when:
- Hinge loss (same as maximizing the margin used by SVMs)

Minimizing hinge loss in Batch Setting

- Given a dataset:
- Minimize average hinge loss:
- How do we compute the gradient?

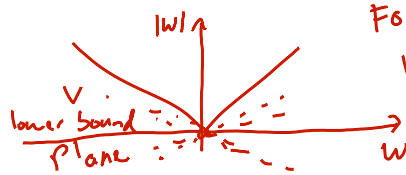
Subgradients of Convex Functions

- Gradients lower bound convex functions:



$$F(w') \geq F(w) + \nabla F(w)^T (w' - w)$$

- Gradients are unique at **w** iff function differentiable at **w**
- Subgradients: Generalize gradients to non-differentiable points:
  - Any plane v that lower bounds function:



For $|w|$ at 0: $v \in [-1, 1]$

$v \in \partial_w F(w)$ subgradient iff

$$F(w') \geq F(w) + v^T (w' - w)$$

Subgradient of Hinge

- Hinge loss:
- Subgradient of hinge loss:
  - If $y^{(t)} (w.\mathbf{x}^{(t)}) > 0$:
  - If $y^{(t)} (w.\mathbf{x}^{(t)}) < 0$:
  - If $y^{(t)} (w.\mathbf{x}^{(t)}) = 0$:
  - In one line:

Subgradient Descent for Hinge Minimization

- Given data:
- Want to minimize:
- Subgradient descent works the same as gradient descent:
  - But if there are multiple subgradients at a point, just pick (any) one:

Perceptron Revisited

- Perceptron update:

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \mathbb{1}\left[y^{(t)}(\mathbf{w}^{(t)} \cdot \mathbf{x}^{(t)}) \leq 0\right] y^{(t)}\mathbf{x}^{(t)}$$

- Batch hinge minimization update:

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \eta\frac{1}{N}\sum_{i=1}^{N}\left\{\mathbb{1}\left[y^{(i)}(\mathbf{w}^{(t)} \cdot \mathbf{x}^{(i)}) \leq 0\right] y^{(i)}\mathbf{x}^{(i)}\right\}$$

- Difference?

What you need to know

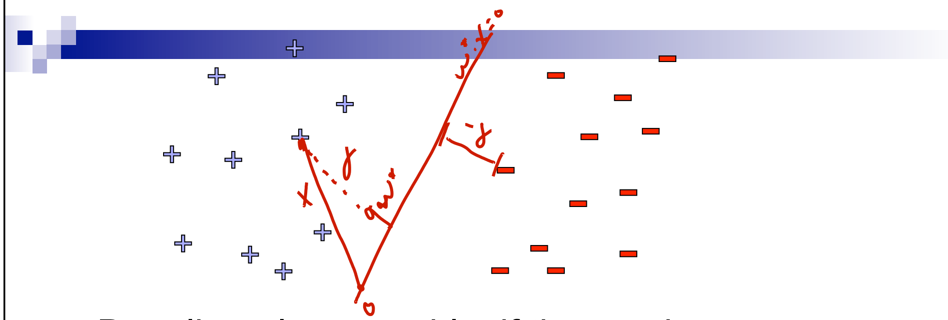
- Perceptron is optimizing hinge loss
- Subgradients and hinge loss
- (Sub)gradient decent for hinge objective

Kernels

Machine Learning – CSE446
Carlos Guestrin
University of Washington

May 1, 2013

Linear Separability: More formally, Using Margin



- Data linearly separable, if there exists
  - a vector $\exists w^*, \|w^*\|=1$
  - a margin $\gamma > 0$
- Such that all points are γ far or more from $w^* \cdot x = 0$

$\forall t$ if $y^{(t)} = +1$ $w^* \cdot x^{(t)} > \gamma$

$y^{(t)} = -1$ $w^* \cdot x^{(t)} < -\gamma$

$\Big\}$ $\underbrace{y^{(t)}\, w^* \cdot x^{(t)} > \gamma}_{\text{linearly separable, margin } \gamma}$

Perceptron Analysis: Linearly Separable Case

- Theorem [Block, Novikoff]:
  - Given a sequence of labeled examples: $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}) \ldots (x^{(T)}, y^{(T)})$

    examples need not be iid. or random..
  - Each feature vector has bounded norm:

    $\forall t \quad \|x^{(t)}\| \le R$ $\qquad$ w^* is unknown!
  - If dataset is linearly separable:

    $\exists w^*, \|w^*\|=1 \quad \forall t \quad y^{(t)}\, w^* \cdot x^{(t)} \geq \gamma, \text{ for } \gamma > 0$
- Then the number of mistakes made by the online perceptron on ~~this~~ any sequence is bounded by

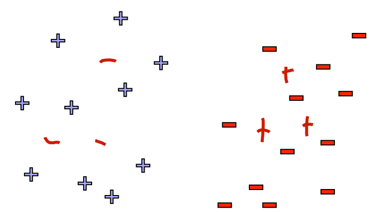
$$\left(\frac{R}{\gamma}\right)^2$$

wow!!

constant, doesn't depend on T or dimensionality of X !!

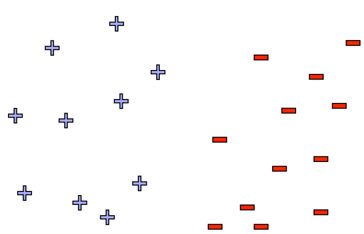
Beyond Linearly Separable Case

- Perceptron algorithm is super cool!
  - No assumption about data distribution!
    - Could be generated by an oblivious adversary, no need to be iid
  - Makes a fixed number of mistakes, and it's done for ever! ← $\left(\frac{R}{\gamma}\right)^2$
    - Even if you see infinite data
- However, real world not linearly separable
  - Can't expect never to make mistakes again
  - Analysis extends to non-linearly separable case
  - Very similar bound, see Freund & Schapire
  - Converges, but ultimately may not give good accuracy (make many many many mistakes)



we need features that make data as linearly separable as possible

What if the data is not linearly separable?



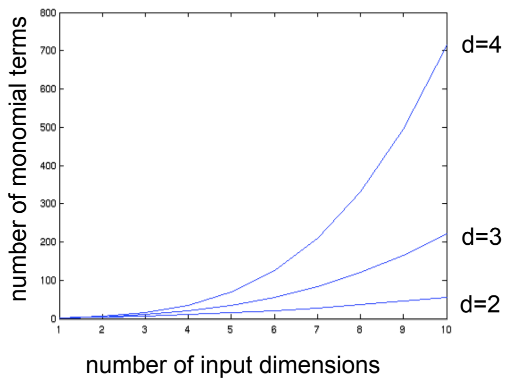
Use features of features of features of features….

$$\Phi(\mathbf{x}) : R^m \mapsto F$$

Feature space can get really large really quickly!

Higher order polynomials

$$\text{num. terms} = \binom{d+m-1}{d} = \frac{(d+m-1)!}{d!(m-1)!}$$



m – input features
d – degree of polynomial

grows fast!
d = 6, m = 100
about 1.6 billion terms

Perceptron Revisited

- Given weight vector $w^{(t)}$, predict point **x** by:
- Mistake at time *t*: $w^{(t+1)} = w^{(t)} + y^{(t)} x^{(t)}$
- Thus, write weight vector in terms of mistaken data points only:
  - Let $M^{(t)}$ be time steps up to *t* when mistakes were made:
- Prediction rule now:
- When using high dimensional features:

Dot-product of polynomials

$\Phi(\mathbf{u}) \cdot \Phi(\mathbf{v}) =$ polynomials of degree exactly d

Finally the Kernel Trick!!! (Kernelized Perceptron

- Every time you make a mistake, remember $(x^{(t)}, y^{(t)})$
- Kernelized Perceptron prediction for **x**:

$$\begin{aligned} \text{sign}(\mathbf{w}^{(t)} \cdot \phi(\mathbf{x})) &= \sum_{i \in M^{(t)}} \phi(\mathbf{x}^{(i)}) \cdot \phi(\mathbf{x}) \\ &= \sum_{i \in M^{(t)}} k(\mathbf{x}^{(i)}, \mathbf{x}) \end{aligned}$$

Polynomial kernels

- All monomials of degree d in O(d) operations:

$$\Phi(\mathbf{u})\cdot\Phi(\mathbf{v}) = (\mathbf{u}\cdot\mathbf{v})^d = \text{polynomials of degree exactly d}$$

- How about all monomials of degree up to d?
  - Solution 0:
  - Better solution:

Common kernels

- Polynomials of degree exactly d

$$K(\mathbf{u},\mathbf{v}) = (\mathbf{u}\cdot\mathbf{v})^d$$

- Polynomials of degree up to d

$$K(\mathbf{u},\mathbf{v}) = (\mathbf{u}\cdot\mathbf{v}+1)^d$$

- Gaussian (squared exponential) kernel

$$K(\mathbf{u},\mathbf{v}) = \exp\left(-\frac{||\mathbf{u}-\mathbf{v}||}{2\sigma^2}\right)$$

- Sigmoid

$$K(\mathbf{u},\mathbf{v}) = \tanh(\eta\mathbf{u}\cdot\mathbf{v}+\nu)$$

What you need to know

- Notion of online learning
- Perceptron algorithm
- Mistake bounds and proofs
- The kernel trick
- Kernelized Perceptron
- Derive polynomial kernel
- Common kernels
- In online learning, report averaged weights at the end