Online Learning
Perceptron Algorithm

Machine Learning – CSE546
Carlos Guestrin
University of Washington

October 23, 2013

Challenge 1: Complexity of Computing Gradients

$$w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta \left\{ -\lambda w_i^{(t)} + \sum_j x_i^j [y^j - \hat{P}(Y^j = 1 \mid \mathbf{x}^j, \mathbf{w}^{(t)})] \right\}$$

$\forall i$, cost is $O(Nk)$

if N is huge, we have a problem

→ SGD, looks at one data point at a time

Challenge 2: Data is streaming

- Assumption thus far: **Batch data**

  Have all data before you learn

- But, e.g., in click prediction for ads is a streaming data task:
  - User enters query, and ad must be selected:
    - Observe $\mathbf{x}^j$, and must predict y^j

  $R \to [\,\vdots\,] \to x^j \longrightarrow \hat{y}$ as to whether you'll click on ad

  (page, user, features of ad)

  $\to$ after, you reveal y^j

  - User either clicks or doesn't click on ad:
    - Label y^j is revealed afterwards
      - Google gets a reward if user clicks on ad, lose money if $\mathbb{1}(\hat{y} \neq y^j)$

  either you click or you don't + update model

  - Weights must be updated for next time: what's Δ?

  $w^{(t+1)} \leftarrow w^{(t)} + \Delta$

Online Learning Problem

- At each time step t:
  - Observe features of data point:
    - Note: many assumptions are possible, e.g., data is iid, data is adversarially chosen… details beyond scope of course

    $x^{(t)} \leftarrow$ (page, user, ad)

    $x^{(t)} = \begin{pmatrix} 1 \\ \text{page} \\ \text{user} \\ \text{ad} \end{pmatrix}$ ← constant

  - Make a prediction: $\hat{y} \leftarrow \text{sign}(w^{(t)} \cdot x^{(t)})$
    - Note: many models are possible, we focus on linear models
    - *For simplicity, use vector notation*

    $w_0^{(t)} + \sum_i w_i^{(t)} x_i^{(t)} \geq 0 \Rightarrow 1$, otherwise $\Rightarrow -1$

    $w^{(t)} \cdot x^{(t)} = \sum_{i=0}^{k} w_i^{(t)} x_i^{(t)} = w_0^{(t)} + \sum_i w_i^{(t)} x_i^{(t)}$

  - Observe true label:
    - Note: other observation models are possible, e.g., we don't observe the label directly, but only a noisy version… Details beyond scope of course

    Observe $y^{(t)}$ → clicked, → didn't click

    Mistake $\hat{y} \neq y^{(t)}$

  - Update model:

    $w^{(t+1)} \leftarrow w^{(t)} + \Delta^{(t)}$

Rosenblatt 1957

The Perceptron Algorithm [Rosenblatt '58, '62]

- Classification setting: y in {-1,+1}
- Linear model
  - Prediction: $\hat{y} = \text{sign}(w^{(t)} \cdot x^{(t)})$
- Training: $w^{(0)}$ e.g. 0 .. random motor settings
  - Initialize weight vector:
  - At each time step:
    - Observe features: $x^{(t)}$
    - 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 $\hat{y} = y^{(t)}$
$w^{(t+1)} \leftarrow w^{(t)}$
else $w^{(t+1)} \leftarrow w^{(t)} + y^{(t)} x^{(t)}$

$\text{sign}(w^{(t)} \cdot x^{(t)}) \neq y^{(t)}$

$\Rightarrow y^{(t)} w^{(t)} \cdot x^{(t)} < 0$

mistake

if $y^{(t)} = +1$

$w^{(t)} \cdot x^{(t)} < 0$

$\Rightarrow$ mistake

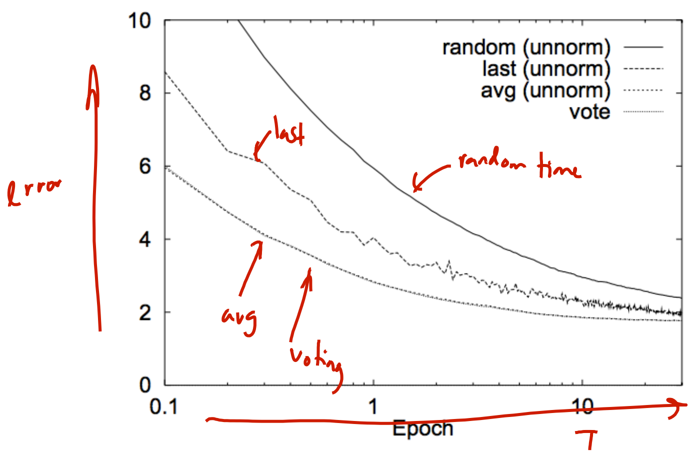
$\Rightarrow y^{(t)} w^{(t)} \cdot x^{(t)} < 0$

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

Fundamental Practical Problem for All Online Learning Methods: **Which weight vector to report?**

- Perceptron prediction: sign$(w \cdot x)$
- Suppose you run online learning method and want to sell your learned weight vector… Which one do you sell???
- Last one? $w^{(T)}$? ← very noisy
- Random time step? ← very noisy
- average!! $\hat{w} = \frac{1}{T+1} \sum_{t=0}^{T} w^{(t)}$
- Voting or more advanced avg, see readings

Choice can make a huge difference!!



[Freund & Schapire '99]

Mistake Bounds

- Algorithm "pays" every time it makes a mistake:

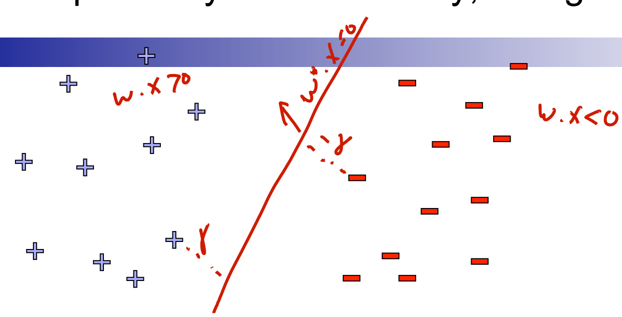
Loss function for this online setting is # mistakes

$\Rightarrow$ Google pays for every mistake

- How many mistakes is it going to make?

Mistake bound

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 at least γ away from $w.x = 0$

$\forall t$: if $y^{(t)} = +1$, $w^*.x^{(t)} \geq \gamma$; Linearly Separable:

$y^{(t)} = -1$, $w^*.x^{(t)} \leq -\gamma$; $y^{(t)}\ w^*.x^{(t)} \geq \gamma$

Perceptron Analysis: Linearly Separable Case

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

    not iid
  - Each feature vector has bounded norm: $\forall t\ ||x^{(t)}|| \leq R$
  - If dataset is linearly separable:

$\exists w^*,\ ||w^*|| = 1,\ \forall t\quad y^{(t)}\, w^* \cdot x^{(t)} \geq \gamma \quad$ for $\gamma > 0$

- Then the number of mistakes made by the online perceptron on any such sequence is bounded by

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

constant, doesn't grow with T!!

$a \cdot b \leq ||a||\,||b||$

Perceptron Proof for Linearly Separable case

Assume $||w^{(0)}|| = 0$

- Every time we make a mistake, we get gamma closer to w^*:
  - Mistake at time t: $w^{(t+1)} = w^{(t)} + y^{(t)} x^{(t)}$
  - Taking dot product with w^*: $w^* \cdot w^{(t+1)} = w^* \left(w^{(t)} + y^{(t)} x^{(t)}\right)$

    $= w^* \cdot w^{(t)} + \underbrace{y^{(t)} w^* x^{(t)}}_{\geq \gamma}$

    $w^* \cdot w^{(t+1)} \geq w^* \cdot w^{(t)} + \gamma$
  - Thus after m mistakes:

by induction $\ldots$ $w^* \cdot w^{(t+1)} \geq m\gamma$

- Similarly, norm of $w^{(t+1)}$ doesn't grow too fast:
  - $||\mathbf{w}^{(t+1)}||^2 = ||\mathbf{w}^{(t)}||^2 + \underbrace{2y^{(t)}(\mathbf{w}^{(t)} \cdot \mathbf{x}^{(t)})}_{\text{mistake: } < 0} + \underbrace{||\mathbf{x}^{(t)}||^2}_{\leq R^2}$

    $||w^{(t+1)}||^2 \leq ||w^{(t)}||^2 + R^2$
  - Thus, after m mistakes:

$||w^{(t+1)}||^2 \leq m R^2$

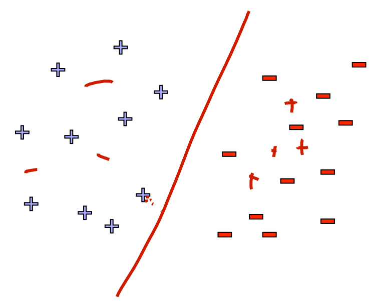
- Putting all together:

$m\gamma \leq w^* \cdot w^{(t+1)} \leq ||w^*||\,||w^{(t+1)}|| \leq \sqrt{m}\, R$

$\Rightarrow \quad m\gamma \leq \sqrt{m}\, R \quad \Rightarrow \quad m \leq \left(\frac{R}{\gamma}\right)^2$ Wow!!

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!
    - 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)

& if data is very very non-linearly separable

What you need to know

- Notion of online learning
- Perceptron algorithm
- Mistake bounds and proof
- In online learning, report averaged weights at the end

What's the Perceptron Optimizing?

Machine Learning – CSE546

Carlos Guestrin

University of Washington

October 23, 2013

What is the Perceptron Doing???

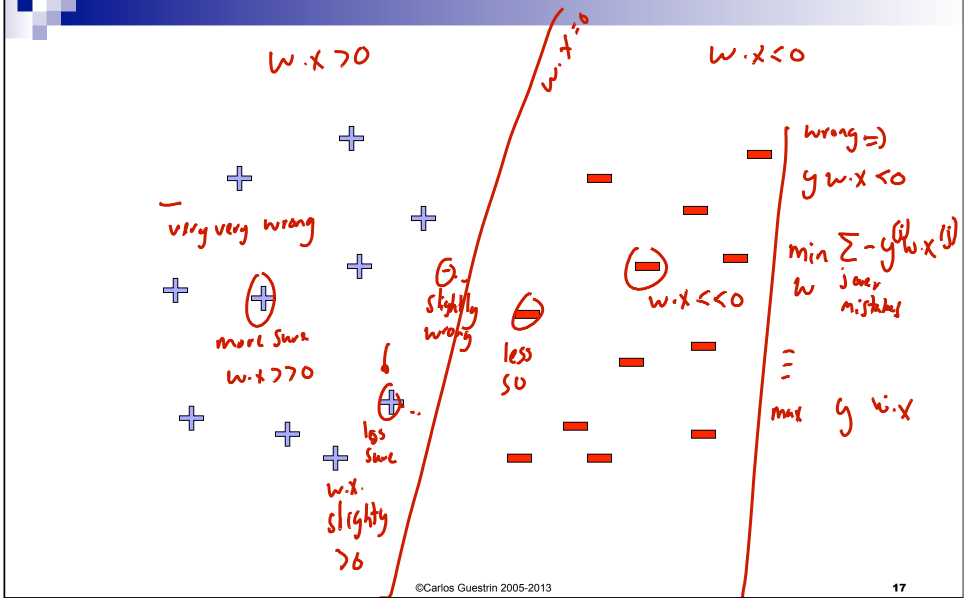
- When we discussed logistic regression:
  - Started from maximizing conditional log-likelihood

$$\max_w \log \prod_j P(y^{(j)} \mid x^{(j)}, w)$$

- When we discussed the Perceptron:
  - Started from description of an algorithm
- What is the Perceptron optimizing????

Perceptron Prediction: Margin of Confidence

$\min -f \Leftrightarrow \max f$



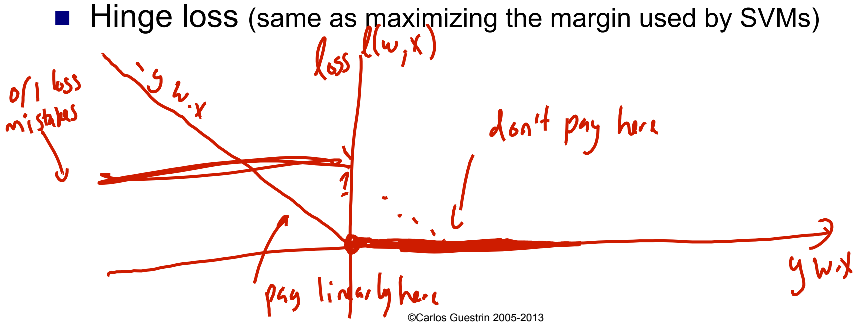
Hinge Loss

- Perceptron prediction: $\text{sign}(w \cdot x)$
- Makes a mistake when:

  $y\ w \cdot x < 0 \implies$

  $$\ell(w,x) = \begin{cases} 0 & \text{if } y\ w \cdot x \geq 0 \\ \cancel{1}\ -y\ w \cdot x & \text{otherwise } (y\ w \cdot x < 0) \end{cases}$$

- Hinge loss (same as maximizing the margin used by SVMs)



$(a)_+ \begin{cases} a & \text{if } a \geq 0 \\ 0 & \text{otherwise} \end{cases}$

Minimizing hinge loss in Batch Setting

- Given a dataset: $(x^1, y^1) \ldots (x^N, y^N)$
- Minimize average hinge loss:

$$\min_w \frac{1}{N} \sum_{j=1}^{N} \ell(w, x^j, y^j)$$

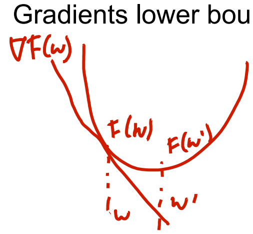
$$\begin{cases} 0 & \text{if } y^j w x^j \geq 0 \\ -y^j w x^j & \text{otherwise} \end{cases} = (-y^j w . x^j)_+$$

- How do we compute the gradient??

$\nabla_w \ell(w, x, y)$

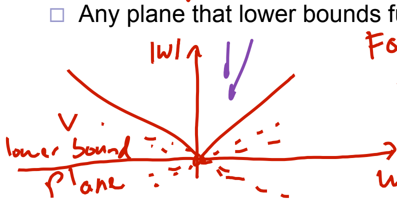
Subgradients of Convex Functions

- 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 that lower bounds function:



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

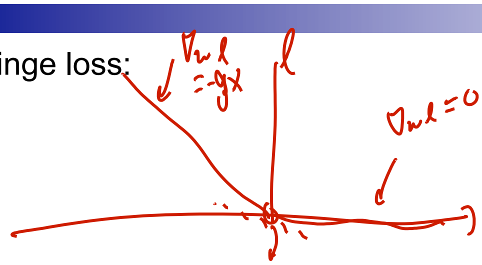
$v \in \partial_w F(w)$ subgradient

iff

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

Subgradient of Hinge

- Hinge loss:



- Subgradient of hinge loss:
  - If $y^{(t)}(w.\mathbf{x}^{(t)}) > 0$: $\nabla_w \ell = 0$
  - If $y^{(t)}(w.\mathbf{x}^{(t)}) < 0$: $\nabla_w \ell = -yx$
  - If $y^{(t)}(w.\mathbf{x}^{(t)}) = 0$: $\partial_w \ell = [-yx, 0]$ ← in subgradient descent, you can pick any of these, e.g. $-yx$
  - In one line:

$$\partial_w \ell(w,x,y) = \underbrace{\mathbb{1}(y\, w.x \le 0)}_{\text{mistake}}(-yx)$$

Subgradient Descent for Hinge Minimization

- Given data: $(x^1, y^1) \ldots (x^N, y^N)$

I want $\min_w$

- Want to minimize:

$$\frac{1}{N}\sum_{j=1}^{N} \ell(w, x^j, y^j) = \frac{1}{N}\sum_{j=1}^{N}\left(-y^j\, w.x^j\right)_+$$

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

$$w^{(t+1)} \leftarrow w^{(t)} - \underset{\text{step size}}{\eta} \sum_{j=1}^{N} \underbrace{\partial \ell(w^{(t)}, x^j, y^j)}_{\underbrace{\mathbb{1}(y^j w^{(t)}.x^j \le 0)}_{\text{mistake?}}(-y^j x^j)}$$

Perceptron Revisited

step size 1 if mistake add $y^{(t)} x^{(t)}$

- Perceptron update:

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

loss; sum over all points; mistake on point i?; add $y^i \cdot x^i$

- Batch hinge minimization update:

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

step size

- Difference? Perceptron as SGD for hinge loss minimization with step size constant $\eta = 1$, and no regularization

What you need to know

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